CENTRO UNIVERSITÁRIO UNISEB

TRABALHO DE CONCLUSÃO DE CURSO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÂO

RECONHECIMENTO AUTOMÁTICO DE PLACAS DE VEÍCULOS UTILIZANDO PROCESSAMENTO DIGITAL DE IMAGENS E INTELIGÊNCIA ARTIFICIAL

Guilherme Stéfano Silva de Souza

Paulo Henrique Passella

Orientador: Prof. Dr. Jean-Jacques G. S. De Groote

RIBEIRÃO PRETO

2011

Guilherme Stéfano Silva de Souza

Paulo Henrique Passella

RECONHECIMENTO AUTOMÁTICO DE PLACAS DE VEÍCULOS UTILIZANDO PROCESSAMENTO DIGITAL DE IMAGENS E INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao UniSEB COC de Ribeirão Preto, sob orientação do Prof. Dr. Jean-Jacques De Groote, para obtenção do grau de bacharel em Ciência da Computação.

Ribeirão Preto

2011

Aos nossos pais, irmãos e amigos por

estarem sempre ao nosso lado

AGRADECIMENTOS

Ao nosso orientador, Prof. Dr. Jean-Jacques De Groote, pela sua paciência e por compartilhar de sua experiência, tempo e disposição contribuindo para a conclusão deste trabalho.

Aos professores e ao coordenador do curso de Ciência da Computação do Centro Universitário UNISEB por passar todo o conhecimento necessário para a nossa formação acadêmica e profissional.

Guilherme: Aos nossos pais e irmãos por sempre estarem ao nosso lado, aos nossos amigos Christian Canalli, Leonardo Meloni e Felipe Miosso por nunca nos deixar desanimar durante todo o curso, ao grande amigo Rafael Ramos pelo companheirismo nos dias sofridos do desenvolvimento deste trabalho, aos amigos de longa data Marcos Vinícios, Atílio Renan e Rógério dos Santos Gaspar, ao Iron Maiden por sempre proporcionar inspiração e tranqüilidade nas noites de trabalho, e a todas as pessoas que não mencionamos aqui, mas sempre serão lembradas por serem importantes em nossas vidas.

Paulo: Aos nossos pais, pois sem eles eu não estaria aqui e a minha noiva, pela paciência de me esperar terminar este trabalho, e principalmente a Deus, pois estou aqui nesse momento.

“A felicidade às vezes é uma bênção,

mas geralmente é uma conquista”.

Paulo Coelho

Resumo

A proposta deste trabalho é o estudo de técnicas de Processamento Digital de Imagens e o desenvolvimento de um software, que utilizando essas técnicas, seja capaz de reconhecer automaticamente placas de veículos.

A metodologia adotada consiste na aplicação de filtros para o pré-processamento das imagens, a localização da placa do veículo utilizando segmentação, e o reconhecimento dos caracteres da placa. Para reconhecer os caracteres são utilizadas Redes Neurais Artificiais.

Para o desenvolvimento do trabalho foi também realizada uma pesquisa sobre as empresas que desenvolvem esse produto atualmente, e os modelos de placas.

Abstract

The purpose of this job is the study of digital image processing techniques, and the development of a software that using these techniques be able to automatically recognize license plates.

The methodology adopted consists of applying filters to pre-processing of images, the location of the license plate using segmentation, characters recognition of the license plates. To the characters recognizition artificial neural networks are used.

To the work development a research had been made about the companies that develop these products nowadays, and the models of license plates.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ALPR: *Automatic License Plate Recognition*

API: *Application Programming Interface*

BIN: Base de Índice Nacional

CAT: *Computerized Axial Tomography*

CBPF: Centro Brasileiro de Pesquisas Físicas

CONTRAN: Conselho Nacional de Trânsito

DCA: Departamento de Engenharia de Computação e Automação Industrial

DENATRAN: Departamento Nacional de Trânsito

dpi : *dots per inch*

FEEC: Faculdade de Engenharia Elétrica e de Computação.

IA: Inteligência Artificial

INSS: *International Neural Networks Society*

JPG: *Joint Photographic Group*

NIH: *National Institutes of Health*

NIMH: *National Institute of Mental Health*

OCR: *Optical character recognition*

PDI: Processamento Digital de Imagens

RAL: *Reichsausschuß für Lieferbedingungen*

RENAVAM: Registro Nacional de Veículos Automotores

RSB: *Research Services Branch*

SIAV: Sistema de Identificação Automática de Veículos

SRPLV: Sistemas de Reconhecimento da Placa de Licenciamento Veicular

UFF: Universidade Federal Fluminense

UFRJ: Universidade Federal do Rio de Janeiro

UNICAMP - Universidade Estadual de Campinas

WEKA: *Waikato Environment for Knowledge Analysis*

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

INTRODUÇÃO

Com a invenção do computador muitos processos trabalhosos foram simplificados, a vida do homem se tornou mais prática e também surgiram campos de estudo e necessidades que não existiam antes da invenção do mesmo.

Porém a necessidade de manipulação de imagens existia mesmo antes do surgimento do computador moderno, pois em 1920 o Sistema Bartlane, feito para que a indústria de jornais pudesse enviar fotos por um cabo submarino, foi como uma das primeiras aplicações técnicas de processamento digital de imagens. O desenvolvimento do processamento digital de imagens ocorreu com a invenção de computadores poderosos o suficiente para realizar tarefas de processamento de imagens no começo da década de 1960 (GONZALEZ; WOODS, 2010).

Com os avanços no processamento digital de imagens, o mesmo passou a ser aplicado em diversas áreas como na medicina em ressonâncias magnéticas, na agricultura para localização de doenças em plantações, na indústria para reconhecimento de placas de circuitos com peças faltantes, e na segurança pública, para reconhecimento de placas de veículos automotores, tema que pode ser aplicado também em engenharia de tráfego e que é o tema desse trabalho.

Com o constante aumento de número de veículos como é dito por GUINDO, THOMÉ e RODRIGUES (2002), a necessidade de um sistema capaz de reconhecer placas de veículos automotores é cada vez maior, como por exemplo, na engenharia de tráfego para conseguir dados rápidos e precisos, e com, isso podendo aumentar a eficiência e controle do tráfego.

A solução de reconhecimento automático de placas de veículos pode ser usada também em várias outras áreas como identificar veículos permitindo ou negando o acesso a áreas restritas em condomínios fechados, por exemplo, procurar e alertar a polícia em caso de roubo ou seqüestro, dentre outras aplicações.

Uma aplicação interessante para esse projeto, e que não foi encontrado nenhum registro da existência desse sistema, seria a utilização dessa tecnologia para achar um veículo perdido em um estacionamento, onde o usuário, por meio da numeração da placa de seu veículo, poderia ser informado em que setor do estacionamento se encontra seu carro.

Poderia também ser criada uma integração de um sistema desses com sites de relacionamentos como os atuais Twitter ou Facebook, por exemplo, onde se acessa um banco de

dados, alimentado por câmeras da cidade e nele pode-se encontrar onde está o carro de um amigo e com isso achar seu amigo.

Atualmente, como é descrito no artigo escrito por CONCI e MONTEIRO (2004) e também no artigo escrito por GUINDO, THOMÉ e RODRIGUES (2002), muitos dispositivos eletrônicos utilizados para aplicar multas na maioria dos estados não possuem um sistema que reconheça automaticamente os caracteres das placas dos veículos. Outro dispositivo em que o reconhecimento automático de placas seria útil é no controle eletrônico de estacionamentos, onde é emitido um recibo automaticamente, mas não é comum o controle de registro das placas de veículos ou identificação das placas dos veículos.

Tudo isso seria possível sem muito custo, pois com o desenvolvimento tecnológico e com o preço cada vez mais acessível das filmadoras, máquinas fotográficas digitais e WebCams , tornam-se viáveis utilizações de reconhecimento por imagens aplicadas à problemas do cotidiano como o reconhecimento de placas de veículos, segundo CONCI e MONTEIRO (2004).

Os sistemas de reconhecimento de placas de veículos automotores ou SRPLV como é descrito pelo DCA ou Departamento de Engenharia de Computação e Automação Industrial da Unicamp (2006), são compostos pelos seguintes itens, Sensor de presença, Câmera de vídeo, Computador e o *Software* que é o item mais importante, e é ele que será abordado nesse trabalho.

Assim como o SRPLV é dividido em vários itens, o software para reconhecimento também será dividido em vários itens, ou módulos. Essa forma de abordagem do problema pode ser vista nos artigos pesquisados como CONCI e MONTEIRO (2004), GUINDO, THOMÉ e RODRIGUES (2002) e na Dissertação de Mestrado de CARVALHO (2006).

Após analisar trabalhos, artigos e dissertações publicados sobre identificação de placas, o trabalho que será desenvolvido tem como objetivo o estudo de técnicas utilizadas no processamento digital de imagens e o desenvolvimento de um protótipo de um software no qual essas técnicas serão utilizadas para que esse sistema seja capaz de detectar de forma automática placas de veículos automotivos em qualquer lugar de uma imagem.

Esse sistema será desenvolvido procurando levar em consideração os temas que foram propostos, os temas que sugeridos como futuros trabalhos, os problemas ocorridos em alguns trabalhos, e tentar de alguma forma unir os mesmos.

O Sistema de Reconhecimento de Placas que será desenvolvido, também será dividido em módulos, essa divisão será feita de forma semelhante ao utilizado no protótipo feito pelo DCA da

Unicamp (2006). As técnicas e metodologias utilizadas serão explicadas no Capítulo de Desenvolvimento.

Além de desenvolver esse protótipo, o trabalho tem como objetivo apresentar um estudo sobre os tipos de placas existentes, suas especificações e as empresas que desenvolvem esses sistemas atualmente.

No capítulo 1 apresentamos um estudo sobre técnicas de processamento de imagens, no capítulo 2 são apresentadas as especificações das placas de automóvies, os trabalhos realizados e empresas que desenvolvem produtos relacionados ao tema, no capítulo 2 apresentamos um estudo sobre redes neurais, no capítulo 3 apresentamos como foi feito o desenvolvimento do sistema e no capítulo final são apresentadas as conclusões e resultados.

Capítulo 1. Processamento Digital de Imagens

Este capítulo aborda o tema Processamento Digital de Imagens citado na introdução, mostrando seu lado histórico, sua definição, as principais técnicas utilizadas, e em quais áreas é utilizado.

1.1. Origens do PDI e suas aplicações

O processamento digital de imagens atualmente está presente nas mais diversas áreas, como medicina, agricultura, segurança, geologia, cartografia, astronomia entre outros. Mas nem sempre foi possível aplicar as técnicas de PDI em todas essas áreas, isso se tornou possível somente depois do desenvolvimento de computadores potentes que suportariam tais aplicações.

Essa seção é dedicada a explicar como surgiu o Processamento Digital de Imagens e em quais áreas está sendo aplicado atualmente.

1.1.1. Perspectiva histórica

Uma das primeiras aplicações técnicas de processamento de imagens foi com a utilização do sistema Bartlane em 1920. O sistema foi feito para que a indústria de jornais pudesse enviar fotos por um cabo submarino (Cabo Bartlane) de Londres para Nova York, o que reduziu de uma semana para menos de três horas o tempo necessário para transportar uma foto pelo oceano. O sistema codificava as imagens para transmissão, e depois reconstituía a imagem no recebimento (GONZALEZ; WOODS, 2010).

Após o recebimento, a imagem era impressa por uma impressora telegráfica, a Figura 1 foi transmitida dessa forma. Mas ocorriam alguns problemas com o uso desse sistema quanto à melhora da qualidade visual dessas primeiras figuras digitais. Esse método de impressão foi substituído em 1921 por uma técnica de reprodução fotográfica através de fitas perfuras.

Figura 1 - Imagem produzida através do sistema Bartlane
FONTE: GONZALEZ; WOODS, 2010, p. 2

Esses primeiros sistemas conseguiam decodificar imagens em cinco níveis de cinza, aumentando para 15 em 1929.

Mesmo esses exemplos envolvendo imagens digitais, os mesmos não podem ser considerados processamento digital de imagens, pois não foram utilizados computadores em seu processamento. O começo do processamento digital de imagens ocorreu no começo da década de 1960 com a invenção de computadores poderosos o suficiente para realizar tarefas de processamento de imagens.

O uso de processamento digital de imagens em fotos tiradas de sondas espaciais teve início em 1964 no Jet Propulsion Laboratory (Pasadena, Califórnia), quando figuras da Lua tiradas pela sonda espacial *Ranger 7* foram processadas para correção de vários tipos de distorções de imagem como é mostrado na Figura 2.

Figura 2 - Primeira foto tirada da Lua capturada pelo *Ranger 7*
FONTE: GONZALEZ; WOODS, 2010, p. 3

1.1.2. Áreas de aplicação

As utilidades do processamento digital de imagens são muitas, e é isso que será discutido resumidamente nessa seção.

> Hoje em dia não, existe praticamente mais nenhuma área de empreendimento técnico que não seja impactada de uma forma ou de outra pelo processamento digital de imagens (GONZALEZ; WOODS, 2010, p.5).

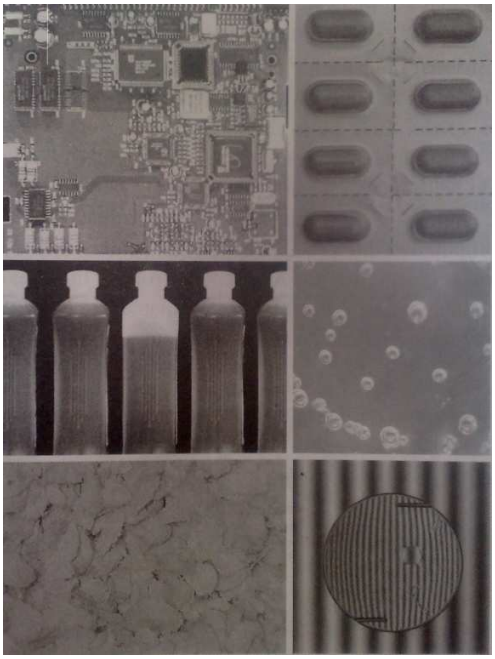
Nesse trecho citado Gonzales e Woods estão enfatizando a importância do Processamento digital de imagens que, com a sua evolução dos computadores, passou a ser usada nas mais diversas áreas utilizando vários tipos de raios do espectro eletromagnético além dos raios que são visíveis ao ser humano. Por exemplo, na medicina a detecção de doenças ósseas, como infecções ou tumores, é feita uma injeção de isótopos radioativos que emitem raios gama em um paciente, esses raios são coletados por detectores que, por fim, montam a imagem.

Ainda na área de medicina outro importante exemplo utilizando raios X é o CAT ou tomografia axial computadorizada onde cada CAT é uma “fatia” do paciente e, à medida que o paciente vai se movendo, são geradas várias fatias que, unidas, formam uma imagem 3D do paciente. Os raios X podem ser usados também na indústria onde uma placa de circuito impresso pode ser analisada em busca de falta de componentes ou com trilhas defeituosas.

Na agricultura, para detecção de fungos no milho são utilizadas técnicas de Processamento Digital de Imagens usando imagens da banda ultravioleta, banda que também é útil na astronomia.

Na indústria farmacêutica uma cartela de comprimidos pode ser inspecionada em busca de compridos faltantes utilizando raios da banda visível, o que pode ser visto na Figura 3(b). Ainda na banda de luz visível, na área de segurança pública pode ser citada a leitura automática de placas de veículos automotores que é utilizada para monitoramento e controle de tráfego (GONZALEZ; WOODS, 2010).

a b



c d

e f

Figura 3 - Exemplos de alguns produtos em que o processamento digital de imagens é utilizado em sua fabricação. (a) Circuito de uma controladora de CD-ROM. (b) Cartela de Comprimidos. (c) Garrafas. (d) Bolhas de ar em um produto de plástico transparente. (e) Cereais. (f) Imagens de um implante intra-ocultar

FONTE: GONZALEZ; WOODS, 2010, p. 11

1.2. O que é processamento digital de imagens.

O processamento digital de imagens é utilizado de duas formas: melhora das informações visuais para a interpretação humana e o processamento de dados de imagens para a percepção automática por máquinas. Até que a imagem seja transformada em alguma dessas formas ela precisa passar por uma série de passos, e para saber quais são esses passos fundamentais, é necessário saber o que é considerado processamento digital de imagens.Serão esses assuntos que serão abordados nessa seção,ou seja, o que pode ser considerado processamento digital de imagens e quais são os passos fundamentais para o processamento digital de imagens.

1.2.1. Definição de Processamento digital de imagens.

> Não existe um acordo geral entre os autores em relação ao ponto em que o processamento de imagens termina e outras áreas relacionadas, como a análise de imagens e a visão computacional, começam. (GONZALEZ; WOODS, 2010, p.1).

Gonzales e Woods ao escreverem esse trecho mostram como a definição do que é processamento digital de imagens pode variar segundo o autor, que pode definir processamento digital de imagens como sendo uma disciplina na qual tanto a entrada quanto a saída devem ser imagens, o que torna o cálculo da intensidade média de uma imagem uma operação não considerada processamento digital de imagens.

Mas existe um paradigma que pode ser utilizado de forma bastante útil levando em consideração três tipos de processos computacionais: processos de nível baixo, médio e alto. O processo de nível baixo envolve operações de pré-processamento, como reduzir o ruído, o realce de contraste e aguçamento de imagens, nesse processo a entrada e saída são imagens. O processo de nível médio envolve tarefas como separar a imagem em regiões ou objetos, classificar e descrever esses objetos, nesse nível a entrada é uma imagem e a saída um objeto extraído dessa imagem. E por fim o processamento de nível alto que é “dar sentido” ao objeto reconhecido.

1.2.2. Passos fundamentais em processamento digital de imagens.

Irão ser abordados nessa seção os passos necessários no processamento digital de imagens, para que a compreensão seja mais fácil será utilizada a aplicação de reconhecimento de placas de veículos, que é proposta nesse projeto, o como exemplo. A Figura 4 ilustra quais são esses passos fundamentais

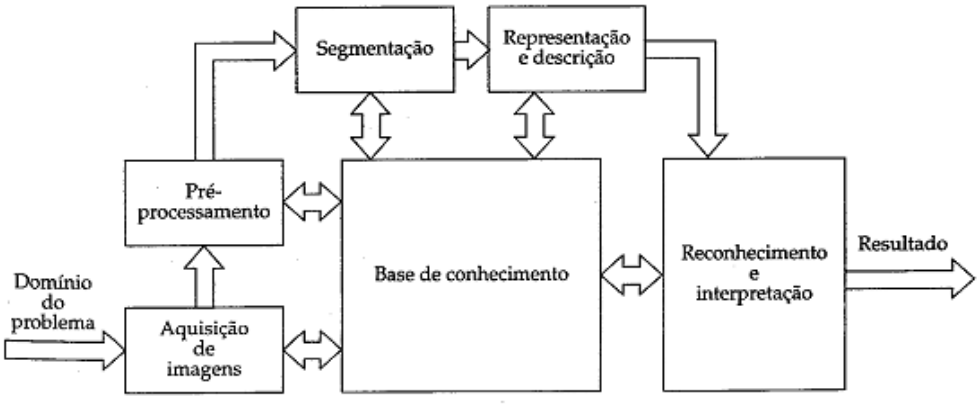


Figura 4 - Passos fundamentais em processamento digital de imagens

FONTE: GONZALES; WOODS, 2000, p.4

A *aquisição de imagens* é o primeiro passo onde a imagem é adquirida, onde a energia eletromagnética (luz) é captada por algum dispositivo, que podem ser câmeras fotográficas, filmadoras, aparelhos de raios x ou scanners. No caso desse trabalho será utilizado um banco de dados de fotos geradas por câmeras digitais.

Após a aquisição da imagem o próximo passo é o *pré-processamento*, que tem a função de melhorar a imagem aumentando as chances de sucesso para os processos seguintes, como técnicas de realce de imagens para retirar ruído, por exemplo. Nesse trabalho dentre as técnicas de realce serão utilizados filtros para ter uma melhor visualização da placa e dos caracteres, como um filtro de realce de contrastes ou um filtro de limiarização, e um para isolamento de regiões que contenham as informações procuradas como detecção de bordas, por exemplo.

O próximo passo é a *segmentação* que tem a tarefa de dividir a imagens em objetos de maior interesse. No caso do problema de reconhecimento de placas de veículos a segmentação pode ser dividida em duas partes, uma que separe a placa do veículo e uma que separe os seus caracteres.

Com a imagem segmentada, os agrupamentos de pixels segmentados devem ser *representados e descritos* de uma forma apropriada para o processamento computacional. Na *representação* de uma região deve se optar pelas escolhas: (1) representar as características externas da região (fronteira), usada quando a preocupação é a forma do objeto, (2) representar pelas características internas da região (pixels que compõe a região), usada para cor ou textura. Em algumas aplicações essas representações coexistem que é o caso do reconhecimento de caracteres de uma placa de automóvel onde são usados algoritmos baseados na forma da borda e também propriedades internas. Após representar é necessário *descrever* a região, onde se procura extrair características de interesse para discriminação entre classes de objetos. No reconhecimento de caracteres, por exemplo, buracos e concavidades são características que auxiliam na diferenciação entre partes do alfabeto.

O estágio de *reconhecimento e interpretação* envolve atribuir um rótulo ao objeto baseado na informação do descritor (por exemplo, “Placa”) que é o reconhecimento. Após o reconhecimento, a *interpretação* envolve a atribuição de significado ao conjunto de objetos reconhecidos. No caso do reconhecimento de placas de veículos, a letra “c” na placa recebe o rótulo “c” e o conjunto de três letras e quatro números é interpretada como uma placa de veículo.

A *base de conhecimento* é utilizada para se ter um conhecimento prévio sobre a imagem, por exemplo, a possível posição de um objeto na imagem, ou os possíveis erros que podem estar na imagem. No reconhecimento de placas de veículos, por exemplo, a placa do carro sempre vai estar no canto inferior da imagem (GONZALES; WOODS, 2000).

1.3. Fundamentos da Imagem Digital

Essa seção aborda os conceitos de aquisição de imagens, o modelo de formação de uma imagem, os conceitos de amostragem e quantização e resolução espacial e de intensidade.

1.3.1. Modelo de formação da imagem

Uma imagem pode ser definida como uma função $f(x,y)$, onde x e y são as coordenadas espaciais e cada par de coordenadas (x,y) é chamada de *intensidade* ou *nível de cinza* da imagem, o que está sendo mostrado na Figura 5. A função $f(x,y)$ pode ser caracterizada por dois

componentes: (1) a quantidade de iluminação da fonte que incide na cena da imagem, e (2) a quantidade de iluminação refletida pelos objetos na cena. Esses componentes são expressos por *i*(*x*,*y*) e *r*(*x*,*y*) e são chamados de *iluminação* e *refletância* como é mostrado na Figura 6. Essas duas funções combinadas resultam em *f*(*x*,*y*).

Figura 5 - Representação de uma imagem digital com seus eixos *x* e *y* e a convenção do ponto inicial para (*x*, *y*)

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 19

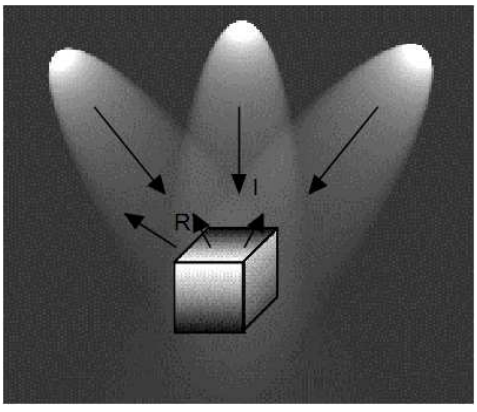


Figura 6 - Os componentes iluminação (i) e refletância(r) de uma imagem

FONTE: MARQUES FILHO,VIEIRA NETO; 1999; p. 20

1.3.2. Amostragem e quantização

> Para ser adequada para o processamento computacional, uma função $f(x,y)$ precisa ser digitalizada tanto espacialmente quanto em amplitude. A digitalização das coordenadas espaciais (x, y) é denominada *amostragem* e a digitalização da amplitude é chamada *quantização* em níveis de cinza (GONZALES; WOODS, 2000, p.21).

Gonzales e Woods explicam nesse trecho que para que um dispositivo consiga gerar uma imagem digital é necessário converter os dados contínuos para dados digitais o que envolve os processos de amostragem e quantização.

A idéia de *amostragem* e *quantização* é ilustrada pela Figura 7, nela existe uma imagem contínua onde os valores das coordenada x e y e da amplitude serão convertidos para o formato digital.

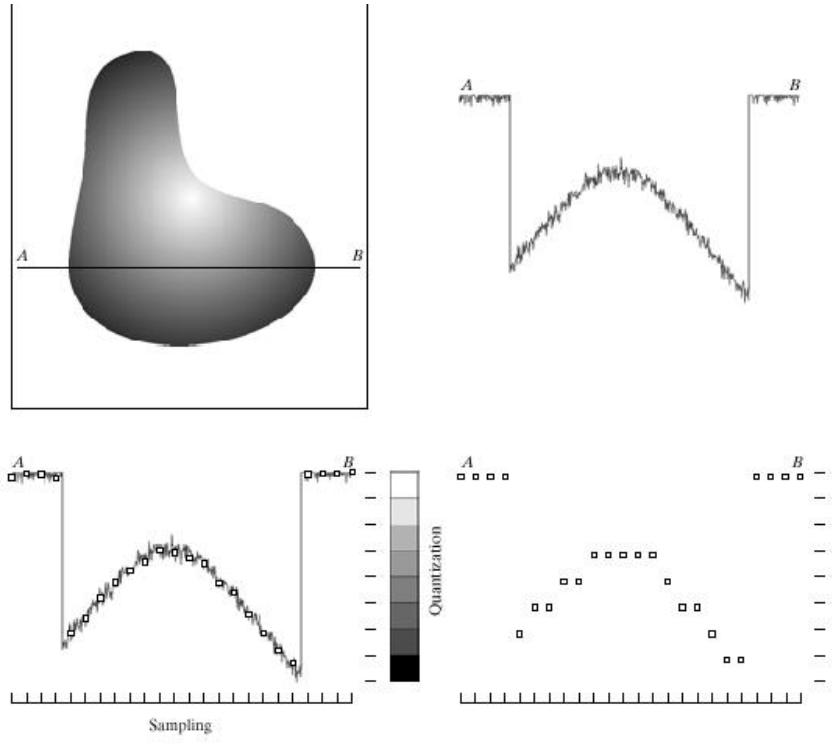


Figura 7 - Produzindo uma imagem digital, (1) Imagem contínua. (2) Linha de varredura de A a B na imagem contínua utilizada para os conceitos de amostragem e quantização. (3) Amostragem e Quantização. (4) Linha de varredura digital

FONTE: (GONZALES; WOODS, 2010, p.34).

Um fator que é importante ser mostrado no tema de *amostragem* e *quantização* é que a qualidade da imagem está relacionada com o número de amostras e de níveis de intensidade, como mostra a Figura 8.

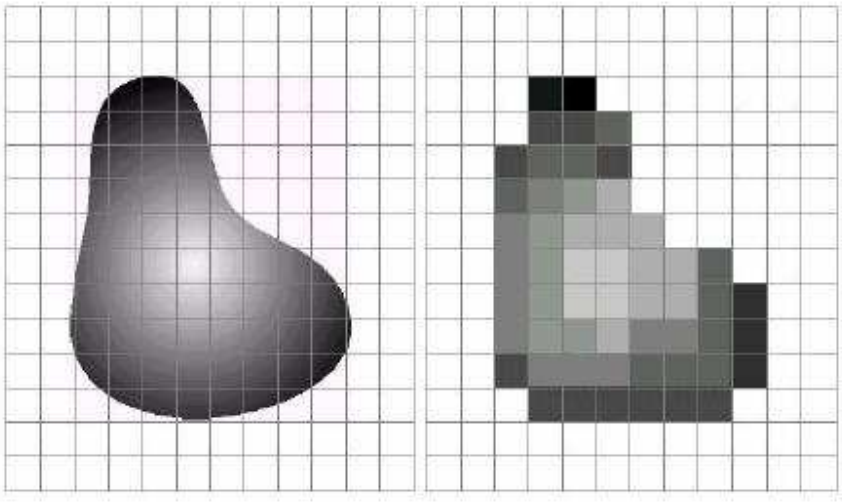


Figura 8 - Imagem contínua projetada para uma matriz e depois convertida para o formato digital através da amostragem e quantização

FONTE: GONZALES; WOODS, 2010, p.35

1.3.3. Resolução Espacial e de Intensidade

A *resolução espacial* é a medida do menor detalhe que pode ser percebido em uma imagem. Essa resolução pode ser expressa de várias formas, sendo que as mais comuns são pares de linhas por unidade de distância e pontos (pixels) por unidade de distância. Um exemplo de quando uma imagem é expressa por pares de linha por unidade de distância seria quando se diz que uma imagem tem 100 pares de linhas por cada *mm* da imagem. A medida pontos por unidade de distância é utilizada na impressão por editoras e indústrias gráficas. Nos Estados Unidos essa medida é conhecida como *dots per inch*(pontos por polegada) ou dpi (GONZALES; WOODS, 2010, p.35).

Porém o tamanho sozinho não faz sentido, dizer que uma imagem possui 1.024*x*1.024 pixels não é muito útil se não se sabe qual a *resolução de intensidade* dessa imagem. A *resolução de intensidade* refere-se aos níveis de cinza ou níveis de cores da imagem. Para se definir o nível de cinza é definido o número de bits reservados para cada pixel como 8 bits, por exemplo.

Tendo explicado a *resolução espacial e de intensidade*, pode se perceber que quando alterados esses valores a imagem sofre mudanças. A diminuição dos níveis de cinza pode causar problemas na imagem como sulcos em áreas de intensidade constante ou parcialmente constante, o que é chamado de “falso contorno”. A Figura 9 mostra a diminuição de *intensidade*.

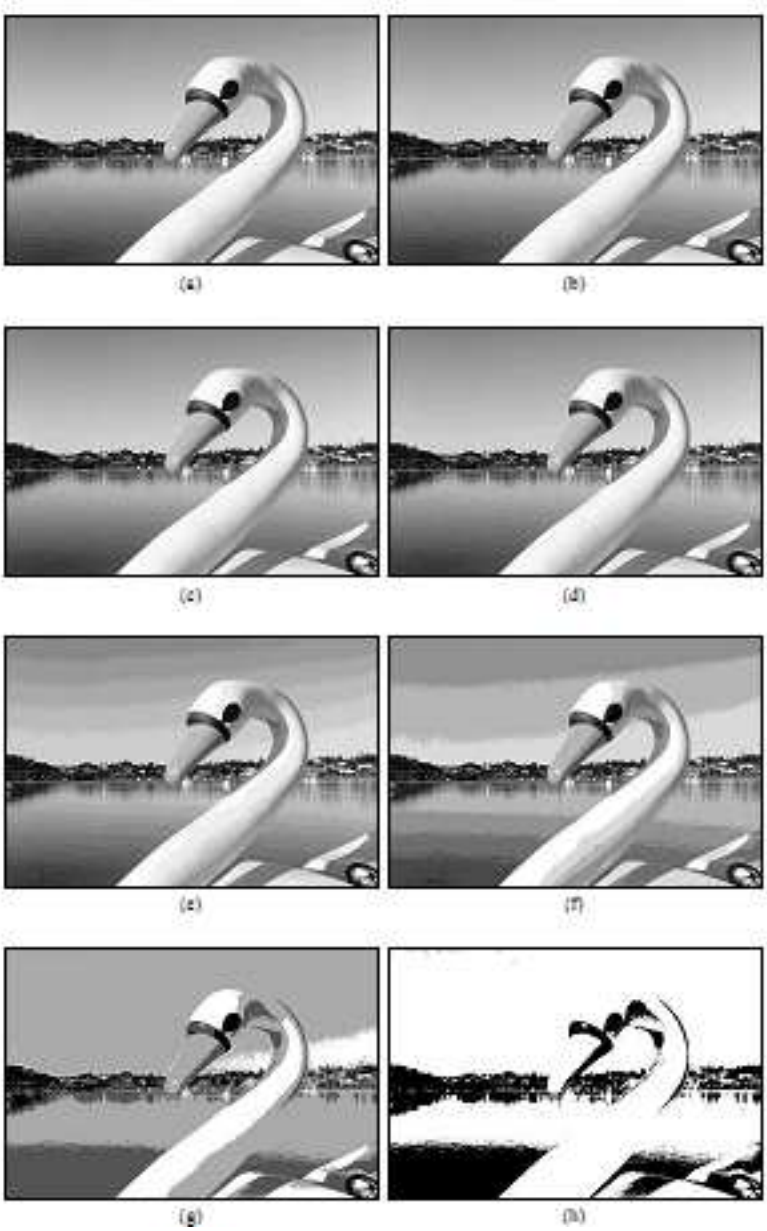


Figura 9 - Efeito do número de níveis de cinza na qualidade de uma imagem 442 x 299 com 256, 128, 64, 32, 16, 8, 4 e 2 níveis de cinza, respectivamente imagem

FONTE: MARQUES FILHO,VIEIRA NETO; 1999; p. 24

A diminuição da *resolução espacial*, que na verdade é diminuir a matriz de pixels da imagem pode causar degradações deixando a imagem com um efeito quadriculado. A Figura 10 mostra a diminuição da *resolução espacial*.

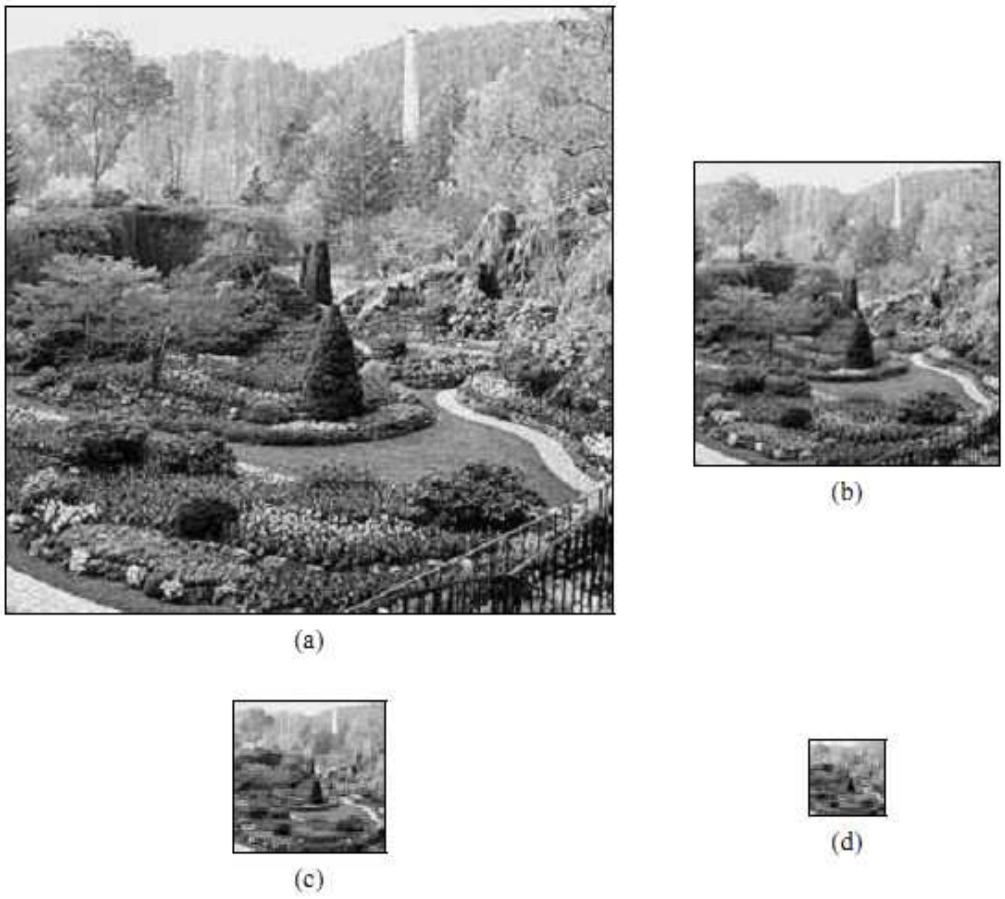


Figura 10 - Diminuição da resolução espacial da imagem
FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 23

.

1.3.4. Conectividade

A conectividade entre pixels é um importante conceito usado para estabelecer limites de objetos e componentes de regiões em uma imagem. Para saber se dois pixels estão conectados, é preciso determinar se eles são adjacentes, ou seja, fazem divisa seguindo algum critério. Para isso é verificado se seus níveis de cinza satisfazem a um determinado critério de similaridade.

O conceito de conectividade depende o conceito de vizinhança entre os pixels. Existem três tipos de vizinhança de 4, vizinhos diagonais e vizinhança de 8.

- Vizinhança de 4: Um pixel *p*, de coordenadas *(x,y)* , tem 4 vizinhos horizontais e verticais, cujas coordenadas *são (x+1, y), (x-1, y), (x, y+1) e (x, y-1).* Essa vizinhança é designada *N4(p).*
- Vizinhos diagonais: os quatro vizinhos diagonais de *p* são os pixels de coordenadas *(x-1, y-1), (x-1, y+1), (x+1, y-1) e (x+1, y+1).* Essa vizinhança é designada *Nd(p).*
- Vizinhança de 8: é a união das duas outras vizinhanças.Definida como $N8(p) = N4(p) \cup Nd(p)$

  A Figura 11 ilustra os tipos de vizinhança:

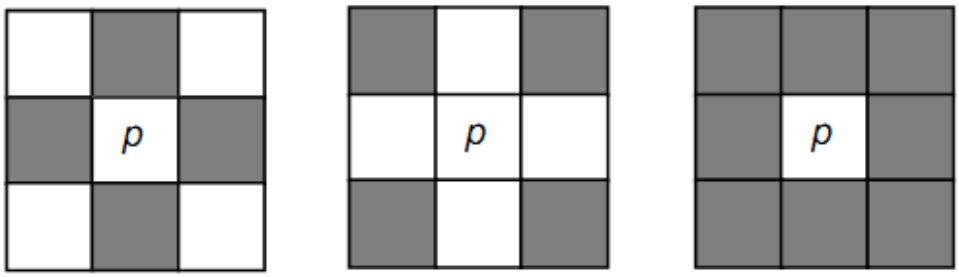


Figura 11 - Conceitos de 4-vizinhança, vizinhança diagonal e 8-vizinhança

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 26

Já sabendo o conceito de vizinhança, um exemplo de conectividade seria uma imagem binária, onde os pixels podem assumir os valores 0 e 1, dois pixels podem ser 4-vizinhos, mas somente serão considerados 4-conectados se possuírem o mesmo valor.

Os tipos de conectividade são:

- 4-conectividade: dois pixels *p* e *q* com valores de tom de cinza contidos em V são 4 - conectados se $q \in N4(p)$.
- 8-conectividade: dois pixels *p* e *q* com valores de tom de cinza contidos em V são 8-conectados se $q \in N8(p)$.
- m-conectados (conectividade mista): dois pixels *p* e *q* com valores de tom de cinza contidos em V, são m-conectados se:

  (i) $q \in N4(p)$ *ou*

(ii) $q \in Nd(p)$ e $N4(p) \cup N4(q) = \emptyset$.

A conectividade mista é uma modificação da 8-conectividade para eliminar os múltiplos caminhos que surgem com 8-conectividade. Um exemplo de conectividade mista pode ser vista na Figura 12(c), sendo que a Figura 12 (b) é uma imagem utilizando a 8-conectividade, repare que a conectividade mista retirou as conexões redundantes. (MARQUES FILHO; VIEIRA NETO, 1999).

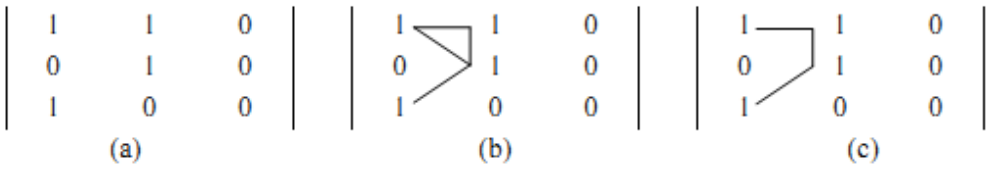


Figura 12 - (a) Segmento de imagem binária, (b) 8-vzinhos do pixel central, (c) m-vizinhos do pixel central
FONTE: (MARQUES FILHO; VIEIRA NETO, 1999, p. 26).

1.4. Realce de imagens

As técnicas de realce de imagens são muito importantes para o processamento digital de imagens, essa seção tem como objetivo mostrar quais são as principais técnicas que serão utilizadas nesse trabalho.

Apesar de importantes, essas técnicas dependem do problema em que serão utilizadas, e por isso precisam ser bem compreendidas e estudadas antes de serem efetivamente aplicadas, pois uma técnica usada por engano pode acabar atrapalhando o processamento digital de imagens, como é citado nesse trecho do livro de MARQUES FILHO e VIEIRA NETO (1999).

> O principal objetivo das técnicas de realce de imagens é processar uma certa imagem de modo que a imagem resultante seja mais adequada que a imagem original para uma aplicação específica. Desta afirmativa decorrem duas importantes conclusões:
>
> 1. A interpretação de que o resultado é mais adequado, ou não, normalmente é subjetiva e depende de conhecimento prévio do observador a respeito das imagens analisadas.
> 2. As técnicas de realce de imagens a serem estudadas neste capítulo são por natureza orientadas a um problema que se deseja resolver. Logo, não existem técnicas

> capazes de resolver 100% dos problemas que uma imagem digital possa apresentar, como também nem sempre uma técnica que produz bons resultados para imagens biomédicas adquiridas através de um tomógrafo computadorizado apresentará desempenho satisfatório se aplicada a uma imagem contendo uma impressão digital, por exemplo (MARQUES FILHO; VIEIRA NETO, 1999, p. 83).

O realce de imagens pode ser classificado em duas categorias: técnicas de filtragem espacial e as técnicas de filtragem no domínio da freqüência. As técnicas de filtragem espacial trabalham diretamente sobre a matriz de pixels que é a imagem digitalizada, normalmente utilizando operações de convolução com máscaras. As técnicas que atuam no domínio da freqüência se baseiam na modificação da transformada de Fourier da imagem. Existem técnicas de filtragem que combinam ambas as abordagens. Essa seção irá abordar somente as técnicas de filtragem espacial.

O trecho citado abaixo mostra como é definida a técnica de filtragem espacial no livro de MARQUES FILHO e VIEIRA NETO (1999).

> As técnicas de filtragem no domínio espacial são aquelas que atuam diretamente sobre a matriz de pixels que é a imagem digitalizada. Logo, as funções de processamento de imagens no domínio espacial podem ser expressas como:
>
> $g(x, y) = T [f(x, y)]$
>
> onde: $g(x,y)$ é a imagem processada, $f(x,y)$ é a imagem original e T é um operador em f, definido em uma certa vizinhança de (x,y) (MARQUES FILHO; VIEIRA NETO, 1999, p. 83).

1.4.1. Histograma.

O histograma de uma imagem pode ser considerado um conjunto de números que indicam a quantidade de cada nível de cinza que existe na imagem. Esses valores são normalmente colocados em um gráfico de barras fornecendo para cada nível de cinza sua respectiva quantidade na imagem. Através do histograma podemos verificar se uma imagem possui um nível de brilho e contraste adequado, assim podendo dizer se essa imagem é muito clara ou escura. A Figura 13 mostra o exemplo de uma imagem e seu histograma. A Tabela 1 mostra um exemplo de histograma e a Figura 14 um gráfico montado com base nessa tabela. (MARQUES FILHO; VIEIRA NETO, 1999).

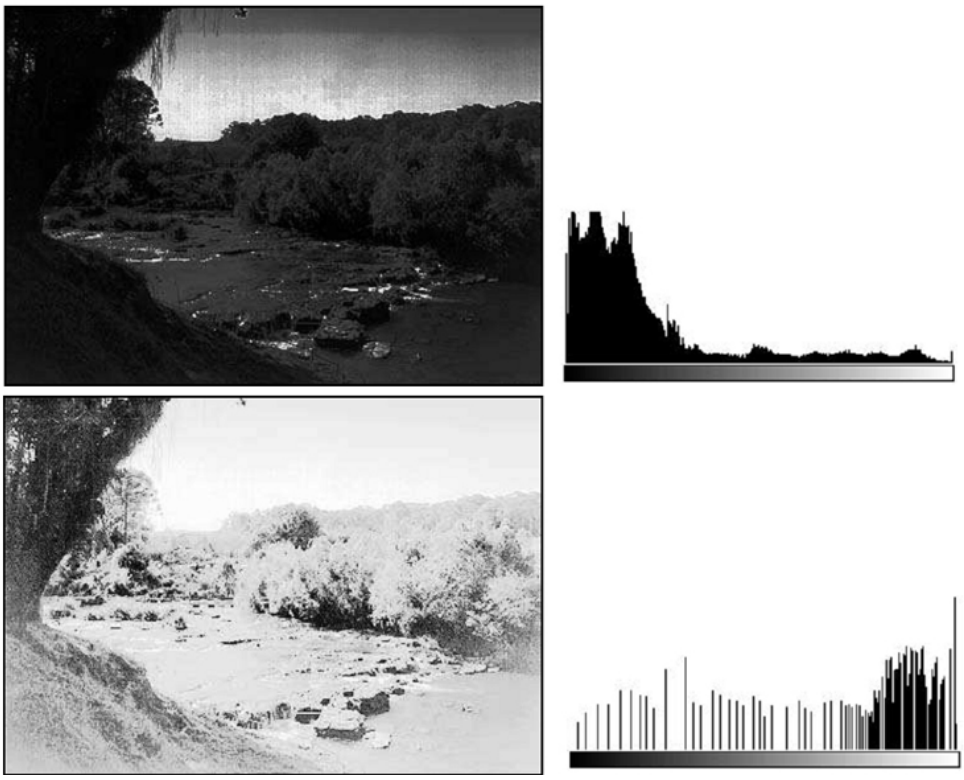


Figura 13 - Imagens e seus respectivos histogramas

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 58

Tabela 1- Exemplo de histograma

Nível de cinza	Número de Pixels
0	1120
1/7	3214
2/7	4850
3/7	3425
4/7	1995
5/7	784
6/7	541
1	455
Total	16384

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 56

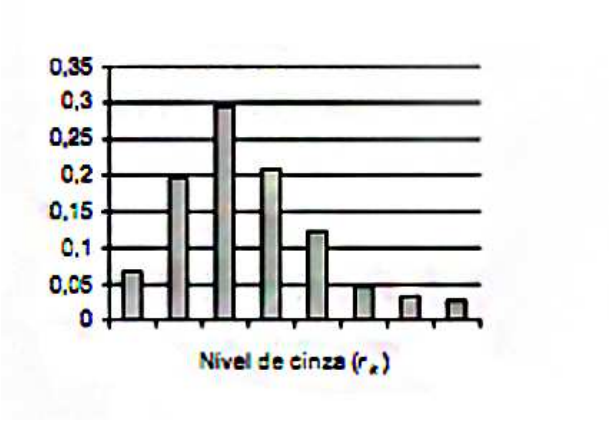


Figura 14 - Exemplo de histograma para imagem com oito níveis de cinza.
FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 56

1.4.2. Equalização de histograma.

A equalização de histograma procura redistribuir os valores de tons de cinza dos pixels de uma imagem, fazendo com que o número de pixels de qualquer nível de cinza seja praticamente o mesmo, obtendo assim um histograma uniforme. A equalização de uma imagem pode ser feita da seguinte forma:

Dada uma imagem de n x m Pixels e *g* níveis de cinza. Sendo o ideal de pixels dado pela formula:

$$I = (n \times m)/g$$

A equalização pode ser realizada então fazendo:

$$q = max\left\{0, ARRED\left(\frac{\sum_{j=0}^{k} nj}{I}\right) - 1\right\} 0 \leq k \leq g$$

Onde: g= níveis de cinza da imagem velha.

q=níveis de cinza da imagem equalizada

Para um exemplo será utilizada uma imagem com 30 pixels e 10 níveis de cinza, o seu histograma e a equalização podem ser representados pela Tabela 2.

Tabela 2 - Exemplo de equalização.

g	n	∑n	q
0	1	1	0
1	9	10	2
2	8	18	5
3	6	24	7
4	1	25	7
5	1	26	8
6	1	27	8
7	1	28	8
8	2	30	9
9	0	30	9

1.4.3. Convolução com Máscaras.

A convolução com máscara é utilizada em inúmeras operações de processamento digital de imagens como no filtro passa alta, passa baixa, média, mediana e outros tipos de filtragens. Na convolução, uma matriz de pequenas dimensões chamada máscara ou janela, é espelhada tanto na horizontal quanto na vertical de uma imagem, percorrendo todos os pontos da mesma deslocando-se ao longo de cada linha e entre as várias linhas, da direita para a esquerda, de cima para baixo, até ter processado o último elemento da matriz imagem. Cada vez que a máscara é espelhada em uma região da imagem, os valores dos pixels dessa região são multiplicados pelos valores da máscara, esses valores são somados e o resultado será o valor do pixel central da máscara. A Figura 15 mostra a convolução sendo aplicada no primeiro pixel de uma imagem. O resultado final de toda a operação será armazenado em uma matriz de mesmas dimensões que a imagem original (MARQUES FILHO; VIEIRA NETO, 1999).

Abaixo um exemplo de convolução:

Seja a imagem dada por:

$$\begin{bmatrix} 5 & 8 & 3 & 4 & 6 & 2 & 3 & 7 \\ 3 & 2 & 1 & 1 & 9 & 5 & 1 & 0 \\ 0 & 9 & 5 & 3 & 0 & 4 & 8 & 3 \\ 4 & 2 & 7 & 2 & 1 & 9 & 0 & 6 \\ 9 & 7 & 9 & 8 & 0 & 4 & 2 & 4 \\ 5 & 2 & 1 & 8 & 4 & 1 & 0 & 9 \\ 1 & 8 & 5 & 4 & 9 & 2 & 3 & 8 \\ 3 & 7 & 1 & 2 & 3 & 4 & 4 & 6 \end{bmatrix}$$

E seja a máscara a seguir:

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 1 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$

A operação de convolução bidimensional produzirá como resultado a matriz:

$$\begin{bmatrix} 20 & 10 & 2 & 26 & 23 & 6 & 9 & 4 \\ 18 & 1 & -8 & 2 & 7 & 3 & 3 & -11 \\ 14 & 22 & 5 & -1 & 9 & -2 & 8 & -1 \\ 29 & 21 & 9 & -9 & 10 & 12 & -9 & -9 \\ 21 & 1 & 16 & -1 & -3 & -4 & 2 & 5 \\ 15 & -9 & -3 & 7 & -6 & 1 & 17 & 9 \\ 21 & 9 & 1 & 6 & -2 & -1 & 23 & 2 \\ 9 & -5 & -25 & -10 & -12 & -15 & -1 & -12 \end{bmatrix}$$

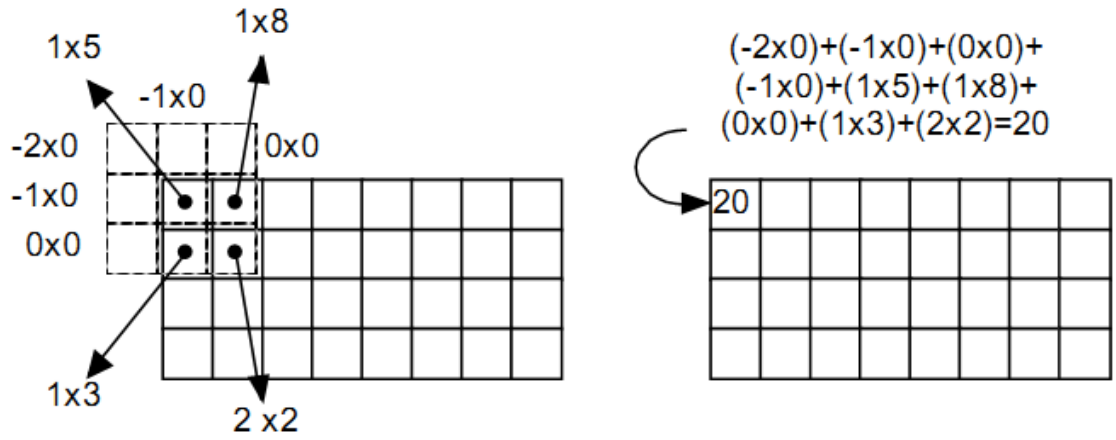


Figura 15 - Primeiro pixel da imagem tendo seu valor alterado pela operação de convolução
FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 36

1.4.4. Limiarização (*Thresholding*).

A limiarização é uma técnica usada para separar as regiões de uma imagem, com essa técnica é possível separar o fundo de uma imagem de um determinado objeto. Como a limiarização produz uma imagem binária como saída, o processo também pode ser denominado como binarização. A limiarização pode ser aplicada convertendo os pixels cujos tons de cinza são maiores ou iguais a um valor de limiar (*T*). A Figura 16 ilustra o funcionamento da limiarização. Por exemplo, se uma imagem tiver 255 tons de cinza e for aplicada a operação limiarização com limiar 128 nessa imagem, todos os pixels com tons menores que 128 receberão o valor 0 e os pixels com tons maiores receberão o valor 255 (MARQUES FILHO; VIEIRA NETO, 1999).

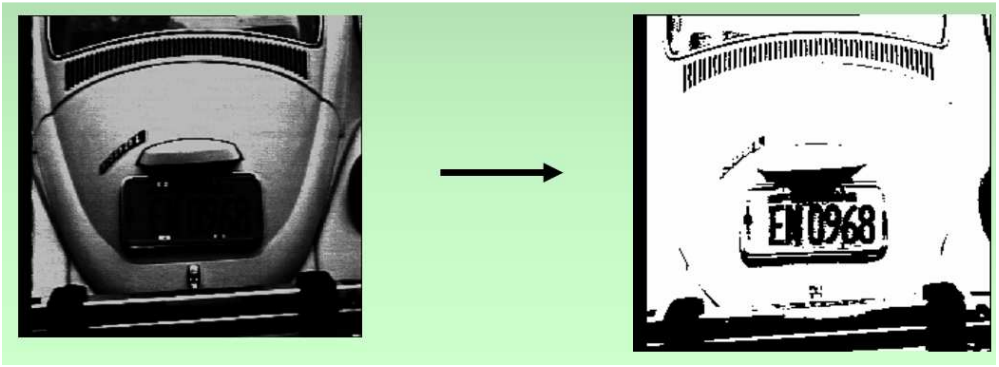


Figura 16 - Imagem original à esquerda e após operação de limiarização à direita

1.4.5. Suavização de imagens no domínio espacial

Os filtros de suavização são utilizados para borramento e redução de ruído. O objetivo da suavização é tornar os valores dos pixels de uma imagem mais homogêneos, assim se o nível de cinza de um pixel for muito diferente dos pixels de sua vizinhança seus valores serão alterados. Com a suavização utilizada no pré-processamento, pequenos detalhes da imagem são removidos antes da extração de objetos (grandes) e também são conectadas pequenas descontinuidades em linhas ou curvas.

Na suavização pode ser utilizada simplesmente a média dos pixels contidos na vizinhança de uma máscara de filtragem, essa técnica pode ser chamada de *filtro de média* ou *filtro passa-baixa*.

Um modo simples de implementar um *filtro de média* é construir uma máscara 3 x 3 com todos seus coeficientes iguais a 1, dividindo o resultado da convolução por um fator de normalização, neste caso igual a 9. A Figura 17(a) mostra uma máscara 3 x 3 , enquanto as Figuras 17(b) e 17(c) ilustram o mesmo conceito, aplicado a máscaras 5 x 5 e 7 x 7. Ao escolher o tamanho da máscara é necessário levar em consideração que quanto maior a máscara, maior o grau de borramento da imagem resultante como é ilustrado na Figura 18 (MARQUES FILHO; VIEIRA NETO, 1999).

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{25}\begin{bmatrix}1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{49}\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1 & 1 & 1 & 1\end{bmatrix}$$

(a) (b) (c)

Figura 17: Máscaras para cálculo de média: (a)3x3; (b)5x5; (c)7x7

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 85

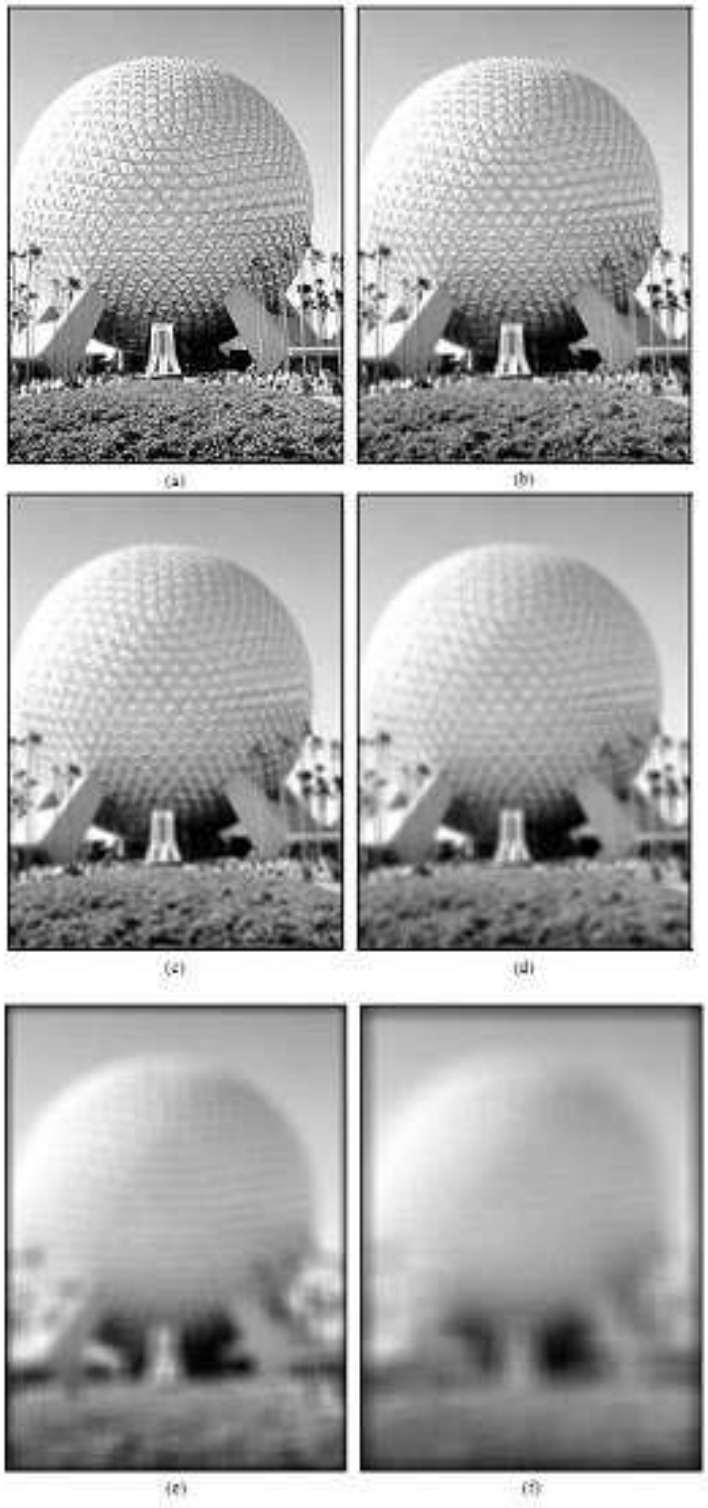


Figura 18 - (a) Imagem original; (b)-(f) resultados da aplicação do filtro da média com máscara de dimensões n x n, n = 3, 5, 7, 17, 317x7

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 87

1.4.6. Filtro passa-alta

O objetivo da filtragem *passa-alta* é realçar detalhes finos na imagem removendo as partes homogêneas e deixando passar os pixels com tons de cinza diferentes de sua vizinhança.

Esse tipo de filtragem também é utilizado para detecção de bordas, o que será comentado na próxima Seção 1.5. A Figura 19 mostra o filtro *passa-alta* básico com uma máscara 3 x 3 projetando uma máscara com pixel central positivo e todos seus oito vizinhos negativos. A soma algébrica dos coeficientes desta máscara é zero, o que faz com que as regiões homogêneas de uma imagem fiquem com valor zero ou um valor muito baixo. A Figura 19 ilustra a aplicação da máscara da Figura 20 em uma imagem.

$$\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figura 19 - Passa-alta básico

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 96

(a) (b)

Figura 20 - (a) Imagem original; (b) imagem após filtragem passa-alta com a máscara da Figura 19

1.5. Morfologia matemática

> O principio básico da morfologia matemática consiste em extrair as informações relativas à geometria e à topologia de um conjunto desconhecido (uma imagem), pela transformação através de outro conjunto completamente definido, chamado elemento estruturante (MARQUES FILHO; VIEIRA NETO, 1999, p. 139).

Como dito no trecho acima através da comparação da imagem original com outra menor denominada elemento estruturante a morfologia matemática é utilizada para extrair informações de uma imagem, e pode ser aplicada em várias áreas do processamento digital de imagens, como realce, filtragem, segmentação, detecção de bordas, esqueletização e afinamento. A seguir serão mostradas algumas técnicas que envolvem morfologia matemática, sendo a dilatação e a erosão a base para a maioria das operações de morfologia matemática.

1.5.1. Erosão

Sendo *A* e *B* conjuntos de Z^2, a erosão de *A* por *B*, indicada por $A \ominus B$, é definida como:

$$A \ominus B = \{z|(B)_z \subseteq A\}$$

Essa equação indica que a erosão de *A* por *B* é o conjunto de todos os pontos *z* de forma que *B*, transladado por z, está contido em *A*. O conjunto *B* é considerado o elemento estruturante (GONZALEZ; WOODS, 2010). Em outras palavras a erosão consiste basicamente no deslocamento linear de cada pixel de coordenadas *(X,Y)* na horizontal e/ou vertical do elemento estruturante sobre uma imagem, tal que ao deslocar o elemento estruturante sobrem somente os pixels da imagem original que estejam totalmente encaixados no elemento estruturante. A Figura 21 mostra exemplos de erosão, e a Figura 22 mostra a erosão em uma imagem com caracteres.

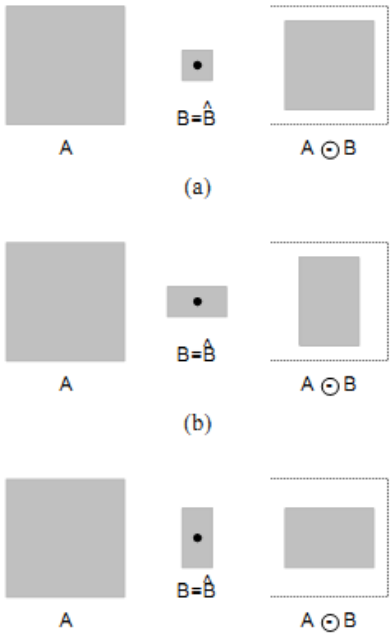


Figura 21 - Exemplo de Erosão

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 143

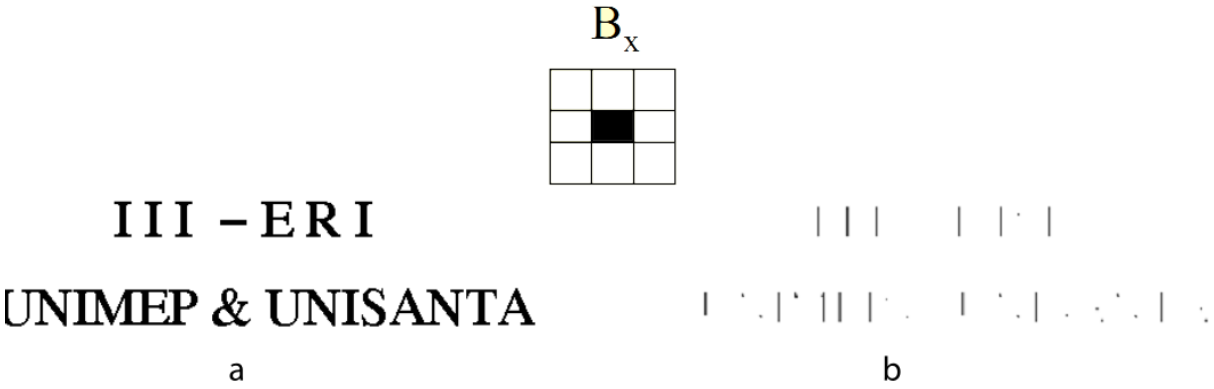


Figura 22 - Exemplo de erosão utilizando o elemento estruturante B_x em uma imagem com caracteres.

1.5.2. Dilatação

Sendo *A* e *B* conjuntos de Z^2, a dilatação de *A* por *B*, indicada por $A \oplus B$, é definida como:

$$A \oplus B = \{z | (B)_z \cap A \neq \emptyset\}$$

A dilatação de *A* por *B* é, então o conjunto de todos os deslocamentos, *z*, de forma que *B* e *A* se sobreponham pelo menos por um elemento (GONZALEZ; WOODS, 2010). Sendo *B* o elemento estruturante e *A* uma imagem, outra forma de explicar a dilatação seria o deslocamento linear de cada pixel de coordenadas (X,Y) na horizontal e/ou vertical do elemento estruturante sobre uma imagem, onde os pixels que são interceptados pelo elemento estruturante são acesos. A Figura 23 mostra um exemplo de dilatação, e a Figura 24 a dilatação em uma imagem com caracteres.

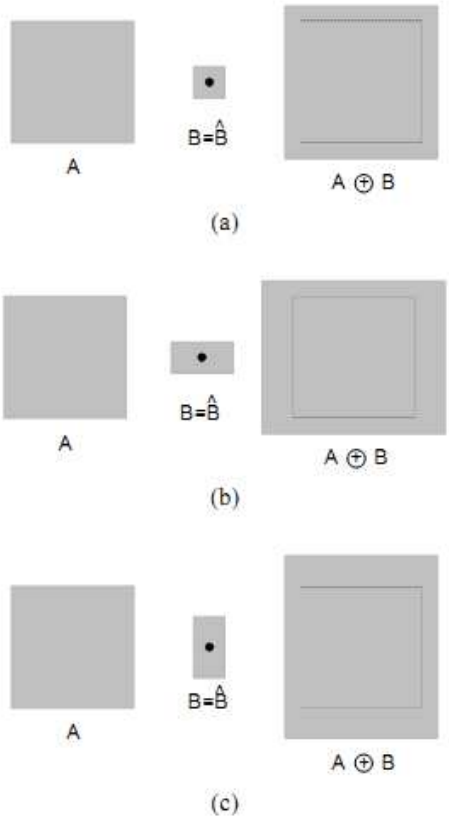


Figura 23 - Exemplo de dilatação

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 142

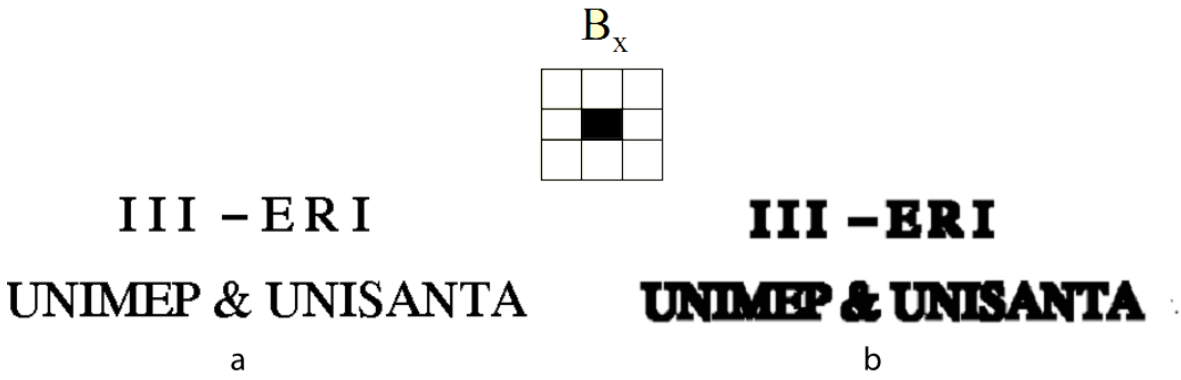


Figura 24 - Exemplo de dilatação utilizando o elemento estruturante B_x em uma imagem com caracteres.

1.5.3. Abertura e Fechamento

Como visto nas seções anteriores a dilatação expande os objetos da imagem enquanto a erosão encolhe. Outras duas operações importantes da morfologia matemática são a abertura e o fechamento. A abertura em geral suaviza o contorno de uma imagem e elimina saliências finas, enquanto o fechamento funde descontinuidades estreitas e alongadas, elimina pequenos buracos e preenche as lacunas em um contorno.

A abertura de um conjunto *A* por um elemento estruturante *B*, denotada $A \circ B$ é definida como:

$$A \circ B = (A \ominus B) \oplus B$$

Isso significa que a abertura nada mais é que uma erosão seguida de uma dilatação. A Figura 25 mostra um exemplo de abertura utilizando um elemento estruturante circular.

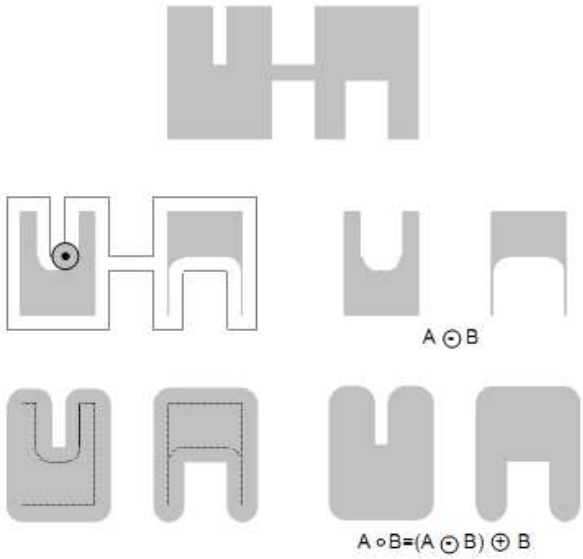


Figura 25 - Exemplo de abertura utilizando um elemento estruturante circular
FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 145

Já o fechamento, é uma dilatação seguida de uma erosão, como pode ser visto na Figura 26 um exemplo de fechamento utilizando um elemento estruturante circular. O fechamento de um conjunto A por um elemento estruturante *B*, denotado $A \bullet B$ é definido como:

$$A \bullet B = (A \oplus B) \ominus B$$

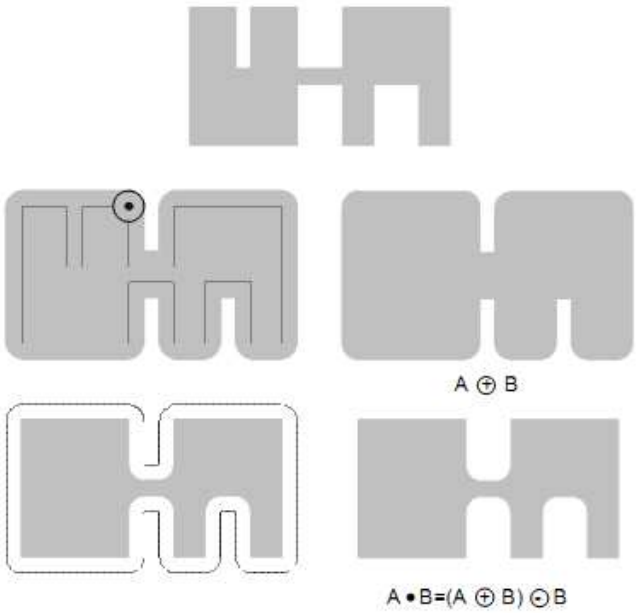


Figura 26 - Exemplo de fechamento utilizando um elemento estruturante circular
FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 145

1.5.4. Esqueletização

A esqueletização é um pré-processamanto, preparando a imagem para outros processamentos. O esqueleto morfológico tem como objetivo o afinamento, encontrando a estrutura interna de determinado objeto. O uso de todos os pixels de um objeto encarece a sua classificação no processo de identificação. Uma maneira de amenizar os custos de processamento é procurar um conjunto de características únicas, ou forma geométrica, que pode ser usado para identificar o objeto, representando sua estrutura básica ou esqueleto (MIRANDA, 2006). A Figura 27 mostra um exemplo de esqueletização.

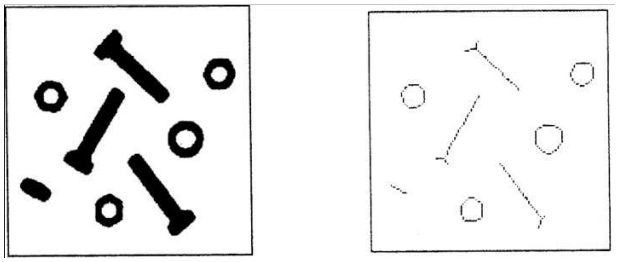


Figura 27 - Exemplo de esqueletização da imagem
FONTE: MIRANDA, 2006, p. 226

O esqueleto por afinamento pode ser representado pela equação:

$$S_n\,(A) = (A \ominus nE)$$

que representa a n-ésima erosão do objeto *A* pelo elemento estruturante *E*.

O algoritmo apresentado para a esqueletização é o algoritmo de Stentiford. Ele utiliza uma série de máscaras 3x3 mostradas na Figura 28. Quando há um casamento entre a máscara e os pixels da imagem, então o pixel central é atribuído um valor branco, conforme o algoritmo:

1. Encontrar um pixel *(i, j)* onde os pixels da imagem se encaixem na máscara M1 (Figura 29).
2. Se o pixel central não for um ponto terminal, e tiver conectividade 1, marcar este pixel para ser retirado.
3. Repetir os passos 1 e 2 para todos os pixels que se encaixem na máscara M1.
4. Repetir os passos 1,2 e 3 para as máscaras M2, M3 e M4.

5. Retirar os pixels marcados, mudando seu valor para branco.
6. Se algum pixel foi retirado no passo 5, repetir todos os passos anteriores , senão parar.

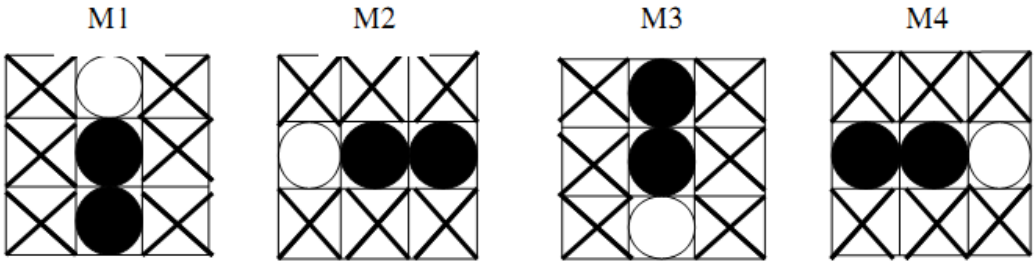


Figura 28: Máscaras para o algoritmo de esqueletização
FONTE: MIRANDA, 2006, p 223

Os pixels especificados nas máscaras como preto e branco devem corresponder a pixels de mesma cor na imagem. O valor X indica lugares onde a cor do pixel não importa. A imagem deve ser percorrida buscando a coincidência com cada formato de máscara. Essas máscaras devem ser percorridas na imagem da seguinte forma:

M1 – da esquerda para a direita e de cima para baixo;
M2 – de baixo para cima e da esquerda para a direita;
M3 – da direita para a esquerda e de baixo para cima;
M4 – de cima para baixo e da direita para a esquerda;

1.6. Segmentação de imagens

A segmentação é um passo importante para conseguir extrair atributos de uma imagem. Esse passo é responsável por subdividir uma imagem em regiões que a compõe, mas o nível de detalhe depende do problema a ser resolvido.

A segmentação de imagens não é uma tarefa simples, sendo umas das tarefas mais difíceis no processamento digital de imagens. A precisão da detecção determina o sucesso ou o fracasso final dos procedimentos de análise computadorizada.

Os algoritmos de segmentação podem baseados em mudanças bruscas de intensidade, como as bordas e divisão de imagens em regiões que sejam semelhantes de acordo com um critério pré-definido. A segmentação também pode ser alcançada com a combinação de métodos de categorias diferentes (GONZALEZ; WOODS, 2010).

1.6.1. Detecção de pontos isolados

O filtro *passa-alta* básico comentado na seção anterior pode ser utilizado para detecção de pontos isolados, pois essa formulação simples mede as diferenças entre um pixel e sua vizinhança de 8. Assim a intensidade de um ponto isolado será muito diferente da sua vizinhança e portanto, será facilmente detectável por essa máscara (GONZALEZ; WOODS, 2010).

1.6.2. Detecção de linhas

Em uma imagem podem existir várias linhas, orientadas a 0º, +45º, -45º, e 90º, essas linhas podem ser detectadas com uma máscara específica. A Figura 29 ilustra essas máscaras. Nessas máscaras a soma de seus coeficientes também é zero, como na detecção de pontos isolados, indicando uma resposta nula em áreas de intensidade constante (GONZALEZ; WOODS, 2010).

$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix} \text{(a)} \qquad \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} \text{(b)}$$

$$\begin{bmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{bmatrix} \text{(c)} \qquad \begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \text{(d)}$$

Figura 29 - Máscaras para detecção de linhas sendo (a) Horizontal; (b) Vertical; (c)-45º; (d)+45º

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 37

1.6.3. Detecção de bordas

Segundo MARQUES FILHO e VIEIRA NETO (1999) o tema detecção de bordas (*edge detection*) é um tema em aberto, pois é um desafio para os pesquisadores da área de Processamento de Imagens há muitos anos.

A borda pode ser definida como fronteira entre duas regiões cujos níveis de cinza predominantes são diferentes. PRATT (1991) define uma borda de luminosidade como uma descontinuidade na luminosidade de uma imagem.

Exemplos de máscaras de detecção de bordas são os operadores de Roberts, Sobel, Prewitt e Freis-Chen, mostrados na Tabela 3. A Figura 30 ilustra o realce de bordas utilizando essas máscaras.

Tabela 3 - Máscaras para detecção de bordas

Operador	*Vertical*	*Horizontal*
Roberts	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Sobel	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Prewitt	$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Frei-Chen	$\frac{1}{2+\sqrt{2}}\begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{2+\sqrt{2}}\begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$

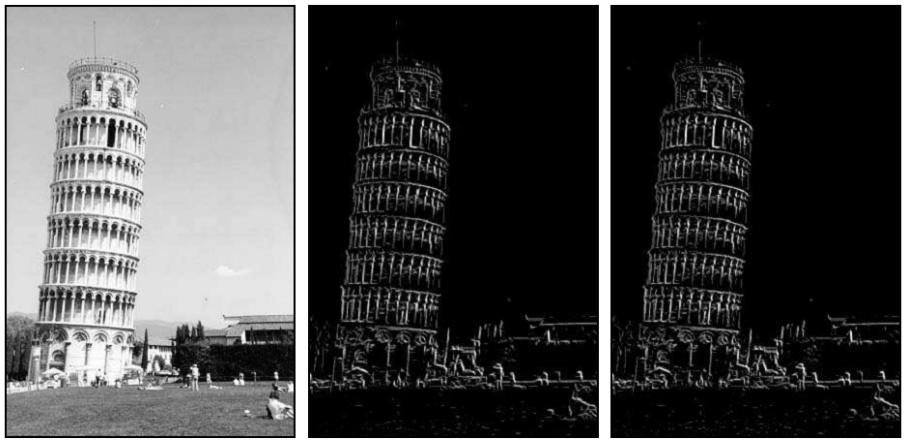


Figura 30- Imagem original, realce de bordas utilizando os operadores de Prewitt horizontal e vertical e realce de bordas utilizando os operadores de Sobel horizontal e vertical respectivamente

FONTE: MARQUES FILHO; VIEIRA NETO, 1999, p. 38

Capítulo 2. Sistemas de Reconhecimento de placas de Veículos

Esse capítulo tem o objetivo de mostrar o que é um Sistema de Reconhecimento de Placa de Veículos, quais são as medidas utilizadas e tipos de placas utilizados atualmente e quais empresas prestam esse tipo de serviço.

2.1. Sistema de Placas de Identificação de Veículos

As placas de identificação de veículos no Brasil são emitidas pelos departamentos de trânsito (DETRAN) e de cada unidade da Federação. Em 15 de março de 2007 foi criada a resolução 231 pelo CONTRAN, especificando vários detalhes das placas veiculares. Essa seção irá abordar os modelos de placas e suas especificações.

2.1.1. Modelos de Placas

De acordo com a resolução 231 de 15 de março de 2007, cada veículo deverá conter duas placas, uma placa dianteira e uma placa traseira, contendo 7 caracteres, sendo eles 3 letras, variando de A a Z e permitindo repetição , e 4 dígitos, esses sendo na base decimal, variando de 0 a 10, e tanto traseiras, quanto dianteiras deverão conter uma tarja com o nome do município e a unidade de federação.

As placas excepcionalizadas deverão conter, gravados nas tarjas ou em espaço correspondente na própria placa, os seguintes caracteres:

- veículos oficiais da União: B R A S I L;
- veículos oficiais das Unidades da Federação: nome da Unidade da Federação;
- veículos oficiais dos Municípios: sigla da Unidade da Federação e nome do Município.

As placas dos veículos automotores pertencentes a Missões Diplomáticas, Repartições Consulares, Organismos Internacionais, Funcionários Estrangeiros Administrativos de Carreira e aos Peritos Estrangeiros de Cooperação Internacional deverão conter as seguintes gravações

estampadas na parte central superior da placa (tarjeta), substituindo-se a identificação do Município:

a) CMD, para os veículos de uso dos Chefes de Missão Diplomática;

b) CD, para os veículos pertencentes ao Corpo Diplomático;

c) CC, para os veículos pertencentes ao Corpo Consular;

d) OI, para os veículos pertencentes a Organismos Internacionais;

e) ADM, para os veículos pertencentes a funcionários administrativos de carreira estrangeiros de Missões Diplomáticas, Repartições Consulares e Representações de Organismos Internacionais;

f) CI, para os veículos pertencentes a perito estrangeiros sem residência permanente que venham ao Brasil no âmbito de Acordo de Cooperação Internacional.

De acordo com a revista Super Interessante (2004) em 1990 foi definido que os caracteres das placas, tanto as letras quanto os números estão relacionados com o local de emplacamento do mesmo, faixas de placas por estado, como visto na Tabela 4 abaixo. Porém podem-se encontrar placas com a faixa e o estado “trocados”, isso acontece quando se emplaca um veículo e o proprietário troca de localidade, então é trocado apenas o nome do município e do estado.

Tabela 4 - Faixas de placas por estado

Estado	Série Inicial	Série Final
Paraná	AAA-0001	BEZ-9999
São Paulo	BFA-0001	GKI-9999
Minas Gerais	GKJ-0001	HOK-9999
Maranhão	HOL-0001	HQE-9999
Mato Grosso do Sul	HQF-0001	HTW-9999
Ceará	HTX-0001	HZA-9999
Sergipe	HZB-0001	IAP-9999
Rio Grande do Sul	IAQ-0001	JDO-9999
Distrito Federal	JDP-0001	JKR-9999
Bahia	JKS-0001	JSZ-9999
Pará	JTA-0001	JWE-9999
Amazonas	JWF-0001	JXY-9999
Mato Grosso	JXZ-0001	KAU-9999
Goiás	KAV-0001	KFC-9999
Pernambuco	KFD-0001	KME-9999
Rio de Janeiro	KMF-0001	LVE-9999
Piauí	LVF-0001	LWQ-9999
Santa Catarina	LWR-0001	MMM-9999
Paraíba	MMN-0001	MOW-9999
Espírito Santo	MOX-0001	MTZ-9999
Alagoas	MUA-0001	MVK-9999
Tocantins	MVL-0001	MXG-9999
Rio Grande do Norte	MXH-0001	MZM-9999
Acre	MZN-0001	NAG-9999
Roraima	NAH-0001	NBA-9999
Rondônia	NBB-0001	NEH-9999
Amapá	NEI-0001	NFB-9999
Goiás	NFC-0001	NGZ-9999

Na resolução 231 de 15 de março de 2007, existe uma tabela de cores identificando quais os modelos de placas de veículos existentes no país e suas respectivas cores como é mostrado na Tabela 5. A Tabela 6 mostra os códigos de cores RAL usados nas placas.

Tabela 5 - Cores das Placas

CATEGORIA DO VEÍCULO	**COR**	
	PLACA E TARJETA	
	FUNDO	**CARACTERES**
Particular	Cinza	Preto
Aluguel	Vermelho	Branco
Experiência/Fabricante	Verde	Branco
Aprendizagem	Branco	Vermelho
Coleção	Preto	Cinza
Oficial	Branco	Preto
Missão Diplomática	Azul	Branco
Corpo Consular	Azul	Branco
Organismo Internacional	Azul	Branco
Corpo Diplomático	Azul	Branco
Organismo Consular/Internacional	Azul	Branco
Acordo Cooperação Internacional :	Azul	Branco
Representação	Preto	Dourado

Tabela 6 - Codificação das Cores

COR	**CÓDIGO RAL**
CINZA	7001
VERMELHO	3000
VERDE	6016
BRANCA	9010
AZUL	5019
PRETA	9011

2.1.2. Especificações técnicas das placas

De acordo com a resolução 231 de 15 de março de 2007 as dimensões das placas dos veículos devem ter uma largura de 400 milímetros e altura de 130 milímetros como mostra a Figura 31.

Figura 31 - Dimensões de uma placa de veículo

FONTE: Contran; resolução 231 de 15 de março de 2007; p. 9

Para motocicletas as placas de identificação devem ter uma largura de 187 milímetros e altura de 136 milímetros conforme mostra a Figura 32.

Figura 32 - Dimensões de uma placa de motocleta

FONTE: Contran; resolução 231 de 15 de março de 2007; p. 9

Os caracteres das placas também devem seguir um padrão, sendo 63 milímetros de altura com espessura de 10 milímetros para veículos, e 42 milímetros de altura e 6 milímetros para motocicletas. A Tabela 7 mostra a altura dos caracteres em milímetros para veículos e a Tabela 8 para motocicletas.

Tabela 7 - A altura dos caracteres em milímetros para veículos

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
54	44	44	43	40	40	45	45	10	36	49	40	54	47	45	44	51	46	46
T	U	V	W	X	Y	Z	1	2	3	4	5	6	7	8	9	0		
44	45	49	49	49	47	40	18	36	37	40	36	36	36	38	36	36		

Tabela 8 - A altura dos caracteres em milímetros para motocicletas

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
36	30	30	30	27	27	30	30	6	25	33	27	36	32	30	30	35	31	31
T	U	V	W	X	Y	Z	1	2	3	4	5	6	7	8	9	0		
30	30	33	33	33	32	27	12	24	25	27	24	24	24	26	24	24		

Os caracteres seguem o padrão da Figura 33 de fonte usando a fonte Mandatory.

Figura 33 - Padrão dos caracteres na fonte Mandatory
FONTE: Contran; resolução 231 de 15 de março de 2007; p. 5

2.2 .Conceitos de um Sistema de Reconhecimento de Placas de Veículos

Essa seção busca mostrar o que é um Sistema de Reconhecimento de Placas, quais são seus componentes, seus passos fundamentais, suas utilidades, quais pesquisas realizadas para a identificação de placas e quais empresas desenvolvem o sistema.

2.2.1. Componentes de um Sistema de Reconhecimento de Placas de Veículos

Como é descrito pelo DCA da Unicamp (2006) um Sistema de Reconhecimento de Placas de Veículos, ou SRPLV como é chamado, é composto pelos seguintes itens:

- Sensor de presença (capacitivo, indutivo, etc);
- Câmera de vídeo (analógica ou digital); cabo para transmissão de sinal de vídeo da câmera ao computador;
- Computador (com porta USB ou IEEE 1394 para câmeras digitais; ou com placa digitalizadora de vídeo para câmeras analógicas);
- Software (programa de computador) de SRPLV.

De todos esses itens o principal componente deste sistema é o software. Pois é ele que identifica o veículo através da imagem obtida pela câmera de vídeo.

O funcionamento desses itens em conjunto segundo o DCA da Unicamp (2006) começa quando um veículo passa pelo sensor de presença, então a câmera de vídeo captura a imagem e envia ao computador. O próximo item a entrar em ação é o *software*, processando a imagem do veículo em busca da placa de licenciamento veicular. O *software* primeiro corrige eventuais distorções da imagem essa etapa pode ser chamada de pré-processamento, em seguida separa os caracteres da placa, etapa que pode ser chamada de segmentação, e identifica cada caractere.

O *software* , após o reconhecimento dos caracteres, para que possa buscar informações do condutor, ou alguma situação ilegal do veículo poderia estar interligado com o sistema RENAVAM, que segundo o DETRAN(1990) é um sistema composto por uma base central (BIN) e pelos cadastros informatizados de todos os estados, interligados através de uma rede nacional de computadores, com o objetivo de integrar informações sobre veículos, tornando-as disponíveis em todo o território nacional, através da ligação da BIN às bases de dados estaduais.

2.2.2. Quais são as aplicações de um Sistema de Reconhecimento de Placas de Veículos

Esse sistema pode ter muitas aplicações e será citado o que algumas instituições que já fizeram esse projeto e algumas empresas que vendem esse produto descrevem como sendo uma área em que esse produto pode ser aplicado.

Segundo o DCA da Unicamp (2006) as principais áreas de aplicação são:

- **Fiscalização:** infração de trânsito e sinistro em outro veículo.
- **Controle de acesso:** identificar cada veículo na entrada e saída em estacionamentos, identificação do veículo em praças de pedágio para posterior cobrança da tarifa, autorizar a entrada de veículos em locais de restrito acesso.
- **Monitoramento do Tráfego:** calcular o tempo e velocidade do veículo no percurso, detectar quando um veículo para em um trecho da rodovia acionando o serviço de socorro.
- **Identificação de Veículo Roubado**: comunicar ao órgão competente o roubo de um veículo.

A empresa SOFTVAIRES utiliza esse sistema também em balanças eletrônicas, onde a pesagem e o monitoramento de possíveis irregularidades na frota de veículos podem ser úteis às administradoras de rodovias nacionais. Já a empresa TECNIMA inclui o sistema na área de controle de frotas e de cargas, monitorando a circulação de containeres e ônibus urbanos através da identificação automática de seqüências de números e/ou letras afixados nos mesmos, independentemente de suas dimensões.

2.2.3. Passos no processamento de um Sistema de Reconhecimento de Placas de Veículos

O processamento em um Sistema de Reconhecimento de Placas de Veículos pode ser dividido em vários módulos. O sistema LooKar do DCA da Unicamp (2006) é dividido em 5 módulos :

- **Pré-processamento:** converte as imagens em escalas de cinza utilizando o sistema de cores YIQ.
- **Localização da Placa:** Utiliza as propriedades de cor e contraste junto com parte da geometria dos caracteres, para encontrar as regiões candidatas a terem a placa de licenciamento veicular
- **Enquadramento da Placa:** retirar partes do veículo que vieram junto da placa na etapa de *Localização da Placa*, e correto enquadramento da placa e a correção da distorção de perspectiva.
- **Separação dos Caracteres:** enquadrar cada caractere para que o mesmo seja reconhecido no módulo de “Reconhecimento dos Caracteres”.
- **Reconhecimento dos Caracteres:** classificar e comparar o caractere a ser reconhecido com todas as amostras.

2.2.4. Pesquisas realizadas para a identificação de placas

No artigo de CONCI e MONTEIRO (2004) o objetivo é o reconhecimento de uma placa em si, sem a necessidade de encontrá-la na foto. A abordagem desse problema é dividida da seguinte forma: primeira parte onde é realizado o pré-processamento utilizando técnicas de binarização e erosão, segunda parte que procura dar rótulos aos caracteres utilizando segmentação dos caracteres da placa através de uma verificação da vizinhança e a última parte que busca dar significado aos caracteres utilizando um banco de dados para comparação das características extraídas nas outras etapas, esse banco de dados deve ser inicializado previamente com essas características.

Já a Dissertação de Mestrado de CARVALHO (2006) define o reconhecimento de placas em três principais módulos: localização da placa, segmentação dos caracteres, e reconhecimento dos caracteres. No primeiro módulo é feita uma busca das regiões que representam a placa. No segundo módulo são segmentados os caracteres. No terceiro módulo com cada um dos caracteres obtidos se faz o reconhecimento dos mesmos. Porém o trabalho de CARVALHO (2006) enfatiza a "localização de placas" que segundo ele é considerada como o estágio mais crucial no sistema de RP e uma que vez que a placa foi encontrada, o resultado pode ser alimentado no segundo e terceiro módulo. Nesse trabalho é utilizada a técnica de Morfologia Matemática que segundo ele se diferencia de outras técnicas onde o principal problema são as características usadas que dependem das diferenças de intensidade entre a placa e as cores do carro e que, portanto não são estáveis, sendo gravemente influenciadas por iluminação, orientação da câmera, ou mudanças na cor do carro.

Em seu sistema CARVALHO (2006) utiliza a plataforma MATLAB em paralelo com a linguagem Python com o toolbox pymorph, deixando como proposta a realização de um estudo para otimizar as rotinas e a implementação em uma linguagem de mais baixo nível e também que é necessário a integração com algum trabalho de Reconhecimento de Caracteres de Automóveis .

A utilização de redes neurais para localizar os caracteres é enfatizada no trabalho de MUNIZ (2007) que cria um sistema em DELPHI 7 onde a localidade da placa é recortada manualmente para que seus caracteres alfanuméricos sejam identificados automaticamente. A sua aplicação também é dividida em fases, a primeira delas é o pré-processamento, onde as técnicas de realce aplicadas são: passar a imagem colorida para tons de cinza, redução da imagem que

consiste em diminuir a quantidade de pixels de uma imagem sem que haja perda significativa da informação nela contida, binarização e vetorização que são aplicados de forma simultânea, onde a binarização é utilizada para diminuição das escalas de cinza da imagem e a vetorização onde matriz da imagem resultante da binarização é transformada em um vetor para que possa ser utilizado pelas redes neurais.

Após o pré-processamento é aplicado o conceito de redes neurais, para que seja localizado na região selecionada da imagem algum padrão, ou seja, algum caractere da imagem com base no treinamento que foi realizado. O treinamento consiste em adaptar os pesos, gerados aleatoriamente, até que sejam obtidas respostas corretas em todos os padrões.

Como proposta de um futuro trabalho MUNIZ (2007) comenta que é necessário um algoritmo que localize a placa em uma imagem e retire desta, automaticamente, os caracteres para servir de entrada na rede.

O artigo publicado por GUINDO, THOMÉ e RODRIGUES (2002) procura fazer todas as etapas do processamento de imagens dividindo elas em: localização da região da placa, seleção e extração da região da placa, segmentação dos caracteres da placa, extração das características dos caracteres segmentados e reconhecimento dos caracteres. Nesse artigo GUINDO, THOMÉ e RODRIGUES (2002) se deparam com vários problemas como a segmentação de caracteres nem sempre conseguir separar todos os caracteres em sete arquivos distintos devido a baixa qualidade das imagens utilizadas, outro problema foi que os resultados obtidos se mostraram bastante abaixo do desejado, devido há alguns problemas nas imagens utilizadas que são: baixa qualidade geral das fotos à disposição, dificuldades como iluminação, ruído gerado pelo ambiente (chuva, neblina, etc.) , problemas com as câmeras (falta de foco, localização inadequada, ângulo de visão impróprio, etc.).Outro fator para os baixos resultados foram as confusões que ocorreram mais freqüentes entre os caracteres “B”, “D” e “O” e entre os dígitos “0” e “8”.Isso aconteceu porque o exterior desses caracteres, seguindo a fonte utilizada na fabricação das placas, tem um desenho muito semelhante.

No artigo de SOUZA *et al* (2006) é desenvolvido um trabalho de otimização do algoritmo de análise de variação tonal para a localização de placas automotivas em tempo real. A variação tonal procura localizar a “impressão digital” da placa fazendo uma varredura seqüencial na imagem.

Como se pode observar, os problemas que foram encontrados nos trabalhos ou as propostas que foram sugeridas para próximos trabalhos pelos autores são complementados uns pelos outros, como por exemplo, CARVALHO (2006) sugere um trabalho para completar o seu que é a segmentação e reconhecimento dos caracteres, essa é a proposta de MUNIZ (2007) e CONCI e MONTEIRO (2004).

2.2.5. Empresas que desenvolvem o sistema

Atualmente existem algumas empresas que desenvolvem o Sistema de Reconhecimento de Placas de Veículos. Mas também existem instituições, além das empresas, que desenvolveram o sistema ou projetos sobre o assunto. A Tabela 9 mostra alguns Sistemas de Reconhecimento de Placas de Veículos com base na pesquisa feita pelo DCA da Unicamp (2006).

Tabela 9 - Sistemas de Reconhecimento de Placas de Veículos

Instituição/Empresa	Sistema/Projeto	Nacionalidade
UNICAMP	LooKar	Brasil
Motorola	ALPR	Estados Unidos
PIPS Technology	ALPR products and services	Estados Unidos
Softvaires	SRPV	Brasil
Compuletra	VistoriaPro	Brasil
PONFAC	Sistema Leitor de Placas PONFAC	Brasil
Tecnima	EVA	Brasil
UFRGS	SIAV 2.0	Brasil
CBPF	SIAV	Brasil
Verde Tecnologia	Família VD-100 (VD-110 e VD-120)	Brasil
HTS	SeeCar	Israel
Adaptive Recognition Hungary	Carmen	Hungria

Capítulo 3. Inteligência Artificial

3.1. Considerações inicias

Há algum tempo o homem procurou maneiras de desenvolver uma máquina que possa “aprender”. Com isso chegou-se em um modelo, esse chamado de rede neural, esse inspirado em nosso próprio cérebro, esse que seja capaz de “imitar” um cérebro em fase de aprendizado.

Podemos considerar essa fase quando nos deparamos com algo novo, algo que não tínhamos conhecimento anterior, algo que teremos que nos dar.

Imagine uma criança em fase de aprendizado, onde você apresenta formas básicas para a mesma, como por exemplo, um quadrado, um triangulo e uma estrela. A criança de primeiro não vai saber “reconhecer” as formas a ela mostrada, então em um processo repetitivo, seu cérebro é “treinado”.

Este treinamento consiste em mostrar uma forma para a criança e “rotular” a mesma, seguindo os seguintes passos:

1 – Mostra o quadrado
2 – Fala o nome da forma

Este procedimento é repetido e após uma determinada quantidade de treinamentos, é faz a pergunta, “Que forma é esta?”. Com isso a criança lhe dará a resposta, de forma correta ou de forma incorreta. Com base a na resposta, treine a criança novamente, este procedimento é repetido até que a taxa de acerto seja satisfatória.

Este mesmo procedimento é aplicado a uma rede neural artificial, já que o intuito de uma rede neural artificial é “imitar” uma rede neural natural.

De acordo com HAYKIN (1999), o cérebro humano é um sistema não linear, paralelo e altamente complexo. O mesmo executa operações altamente complexas e um espaço de tempo muito curto.

De acordo com HAYKIN (1999), uma rede neural artificial é uma representação tecnológica de outras disciplinas, como matemática, neurociência, estatísticas, física.

3.2. Introdução

O que é inteligência artificial?

É uma ciência feita de maquinas inteligente, especialmente programas de computadores inteligentes, similar a usar computadores para entender a inteligência humana (MCCARTHY, 2007).

3.2.1. Conceito de Rede Neural

De acordo com HAYKIN (1999), Uma rede neural é um processador massivo paralelo distribuído feito de simples unidades de processamentos que tem uma propensão para armazenar conhecimentos experimentais e disponibilizá-los para futuro uso.

3.3. Histórico

De acordo com Muniz (2007), o conceito de redes neurais teve inicio em 1943, com McCulloch e Pitts, que sugeriram a construção de uma máquina, esta baseada no funcionamento de um cérebro humano.

Em 1949, Donald Hebb, autor do livro “The Organization of Behavior”, propôs uma lei para a maquina de McCulloch e Pitts.

Snark foi o primeiro computador a simular uma rede neural, o mesmo ajustava seus pesos de forma automática, mas não foi utilizado para nada de grande importância.

Em 1958 foi criado o Mark I Perceptron, na Universidade de Cornel por Frank Rosenbalt. O mesmo era um sistema linear, e foi desenvolvido para resolver problemas primeiramente para reconhecimento de caracteres. Em 1960, Franck Rosenbalt escreveu um livro, princípios de Neurodinâmica, contendo sua idéia sobre a modelagem do cérebro humano.

Em 1959 Bernard Widrow inventou o Adaline, que vem de “Adaptive Linear Neuron”, neurônio linear adaptivo, que funciona como um perceptron, classificando sistemas lineares.

Com esses avanços, foram aparecendo previsões de maquinas como o cérebro humano, sendo assim, a esta área de pesquisa acabou perdendo credito. Somente em 1983 as pesquisas na área começaram a ganhar créditos novamente, com pesquisas mais discretas e com resultados mais satisfatórios.

Em 1987, em São Francisco, foi feita a primeira conferencia de redes neuraus, a IEEE International Conference on Neural Networks, assim formando a INSS.

3.4. Perceptron

O perceptron é uma unidade que aprende conceitos e, por exemplo, pode dar respostas como verdadeiro ou falso, por aprendizado repetitivo, como a criança com formas, onde você apresenta a forma e treina até que a mesma mostre a resposta esperada. É uma unidade que representa um neurônio humano. A Figura 34 mostra a representação de um neurônio humano.

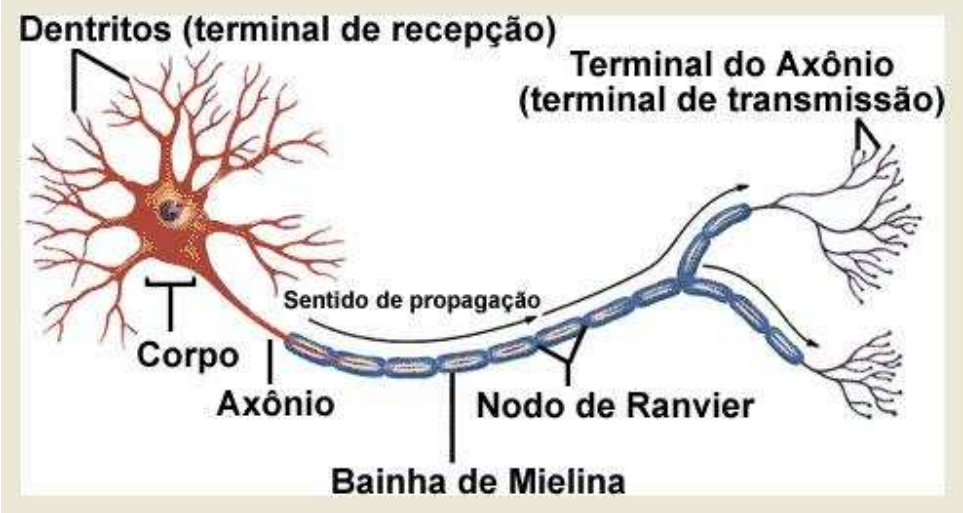


Figura 34 - Neurônio Humano

FONTE: Disponivel em: http://www.paraibavip.com.br/rotativo_ver.php?id=59; Acessada em 30/10/2011

Como pode ser visto o neurônio é composto das seguintes partes: dentritos, corpo, axônio, bainha de Mielina, nodo de Ranvier e terminal de axônio.

O perceptron faz parte de um sistema que serve para realizar classificações, mais precisamente distinguindo classes diferentes.

O perceptron tem uma representação do neurônio humano em uma rede neural artificial, e pode ser representada pela Figura 35.

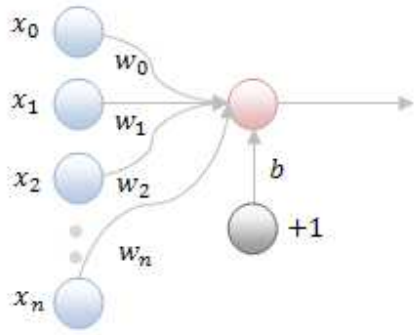


Figura 35: Representação de um perceptron na rede neural.

FONTE: http://www.generation5.org/content/1999/perceptron.asp; Acessado em 30/10/2011

De acordo com TONSIG (2000), O perceptron é uma estrutura lógico-matemática que procura simular a forma, o comportamento e as funções de um neurônio biológico, assim sendo, os dentritos foram substituídos por entradas, cujas ligações com o corpo celular artificial são realizadas através de elementos chamados de peso. Os estímulos captados pelas entradas são processados pela função soma, e o limiar de disparo do neurônio biológico foi substituído pela função de transferência.

De acordo com TONSIG (2000), Todo o conhecimento de uma rede neural está armazenado nas sinapses, ou seja, nos pesos atribuídos as conexões entre neurônios. De 50 a 90% do total de dados deve ser separado para o treinamento da rede neural, dados estes escolhidos aleatoriamente, a fim de que a rede “aprenda” as regras e não “decore” exemplos. O restante dos dados só é apresentado a rede neural na fase de testes, a fim de que ela possa deduzir corretamente o inter-relacionamento entre os dados.

O que TONSIG (2000) quer dizer é que se mostrarmos todas as amostras para a rede, a mesma pode acabar decorando, e não aprendendo.

Como pode ser visto, o perceptron possui vários terminais, que chamamos de atributos, e cada atributos possui um peso, que é a peso do atributo na resposta final do perceptron.

Também pode ser visto que existe um limitador. O perceptron pode ser representado com a seguinte função matemática:

$$Net = \sum_{i=0}^{n} xi\ wi + b$$

$$f(x) = \{\frac{x \geq 0\ \rightarrow 1}{x < 0\ \rightarrow 0}\}$$

Como pode ser visto, existe uma função limitadora, caso o resultado seja >= a 0, a resposta final será 1 e caso contrário, será zero.

3.5. Treinamento

O treinamento de uma rede neural de uma única camada consiste em alterar os pesos dos perceptrons, para alterar esses pesos a rede é trinada com varias amostras, é como treinar uma criança com as formas básicas, podem-se realizar vários treinamentos, até que a rede nos traga a resposta correta, ou seja, quando a mesma nos trazer a reposta de forma correta, isso quer dizer que os pesos deste perceptron estão ajustados.

De acordo com RUSSEL e NORVIG (1995), o perceptron é treinado para responder a vários atributos (entradas) e reponder verdadeiro ou falso, e que a regra de aprendizado converge para uma solução de tempo finito, caso a solução exista.

A regra de aprendizado é dada pelas equações:

$$W(x) = W(x) + (T - A) * E(x)$$
$$b = b + (T - A)$$

Sendo que, *W* é o vetor de pesos. *E* é o vetor dos atributos, *T* é a resposta correta, *A* é a saída atual da rede e *b* é o bias.

O treinamento é realizado da seguinte maneira, um conjunto de amostras é apresentado à rede onde cada atributo é associado a um peso, e a rede é treinada, se a resposta da rede for a

resposta esperada, é passada a próxima amostra, caso a resposta não seja a esperada, os pesos e o bias são reajustadas, esse processo deve ser feito até a rede apresentar um resultado satisfatório.

Em um treinamento uma passagem inteira por toda a rede é chamada de ciclo

3.5.1. Padrões

Quanto maior a quantidade de amostras distintas melhor é o resultado, com isso a rede pode ser mais treinada, chegando aos pesos ideais. A Figura 36 mostra a representação gráfica da letra “A”, esta representação dividida em uma matriz serve para podermos ver como a rede é treinada.

Nessa matriz a rede considera cada pixel, um perceptron, e cada matriz uma amostra.

O pixel [0,0] está em branco, significando que este pixel, no caso da rede neural, não tem “importância”, fazendo com que o mesmo tenha um peso menor em relação ao pixel [4,2] que é onde existe uma parte do caractere “A”, ou seja, este perceptron, tem mais “importância” que o outro perceptron, assim, ele possui um peso maior.

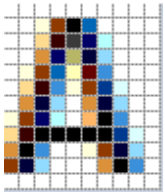


Figura 366- Representação grafica da letra A dividida em pixels

3.5.2. Energia de Treinamento

De acordo com Muniz (2007), durante a fase de treinamento, após cada alteração de pesos, é feito o cálculo da energia, sendo esta dada pela formula:

$$E = \frac{1}{2}\sum_{\mu=1}^{n}(t^{u} - \sum_{t=1}^{m} S_t^{\mu}\, W_t)^2$$

Sendo *n*, o número de padrões que estão sendo usados no treinamento, *t* representa a saída correta, *m* é o número de entradas que cada padrão possui, *S* é o valor de cada entrada *m* e *W* é o valor de cada peso.

Com esse cálculo é possível saber em que estado se encontra o treinamento. Quanto menor é este valor, maior é o percentual de acerto da rede com relação ao conjunto de treinamento.

3.5.3. Algoritmo

Após apresentar uma amostra para a rede, a mesma apresenta um resultado, caso o resultado seja o esperado, outra amostra é apresentada a rede, e caso o resultado não seja o esperado, deve-se reajustar os pesos das entradas. O ajuste dos pesos é realizado pela seguinte equação:

$$W_m^{i+1} = W_m^i + \eta \,.\sum_{\mu=1}^{m}[(t^\mu - T^\mu).S_m^i]$$

Onde:

i é o passo.

η é a taxa de aprendizado.

T é a saída da rede.

Como pode ser vista na fórmula, a taxa de aprendizado é a velocidade do passo. É um ajuste fino, onde quando menor for a taxa de treinamento, melhor vão ser os pesos, com isso uma maior taxa de acerto será obtida, porém, a velocidade será muito menor, em contrapartida, caso a taxa de aprendizado seja grande, pode-se não chegar a um peso que satisfaça a maioria das amostras.

3.6. Considerações Finais

Neste capitulo foi apresentado a parte de inteligência artificial, desde a história até o funcionamento de uma rede neural de uma única camada, esta o suficiente para poder realizar o reconhecimento de caracteres. Com este capitulo é possível iniciar o desenvolvimento de software para o reconhecimento dos caracteres.

Capítulo 4. Desenvolvimento

Neste capítulo serão mostradas as técnicas utilizadas para o desenvolvimento do aplicativo. O desenvolvimento ocorreu em quatro fases, como é mostrado na Figura 37.

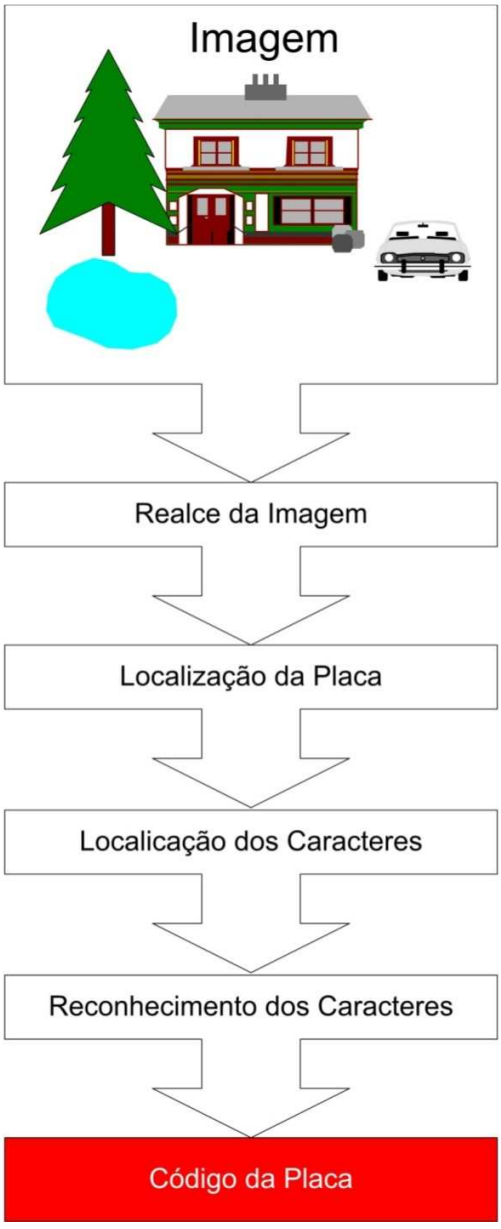


Figura 37- Diagrama dos módulos

Funções de cada módulo:

- Realce: utilização de filtros como escalas de cinza, passa-alta, binarização e esqueletização.
- Localização da Placa: encontrar a região que contenha a placa.
- Localização dos caracteres: localizar os objetos que possam ser possíveis caracteres para que os mesmos sejam reconhecidos no módulo de "Reconhecimento dos Caracteres".
- Reconhecimento dos Caracteres: dar significado aos objetos que foram encontrados no módulo anterior.

4.1. Base de dados

Para desenvolver esse trabalho foi utilizada uma base de dados, pois dispor de uma quantidade significativa de imagens é essencial para esse tipo de pesquisa. Todos os testes foram feitos em uma base contendo carros e caminhões parados em frente a cancelas de cobrança. As imagens foram obtidas do trabalho do Projeto de Reconhecimento de Placas de Veículos Brasileiros desenvolvido por ALBUQUERQUE (2006).

Ao todo são utilizadas 100 imagens dessas 25 não possuem placas ou as placas não estão em condições de serem analisadas, pois possuem muitos desgastes, também existem imagens com muita luminosidade ofuscando totalmente os caracteres ou estão focalizadas pela metade cortando alguns caracteres. Das 75 imagens restantes três estão desgastadas e 12 são de cor vermelha. Todas as imagens estão em escalas de cinza e são do formato JPG.

4.2. Tecnologias utilizadas

Nessa seção serão mostradas as ferramentas utilizadas assim como a linguagem escolhida onde será falado um pouco de cada uma delas.

4.2.1. Java e sua biblioteca para imagens

A linguagem de programação escolhida foi a linguagem Java, por ser orientada a objetos, pela sua capacidade de portabilidade e por ser uma linguagem considerada nova, ou seja, não irá deixar de ser utilizada tão cedo. Também pelo fato de estar fortemente interligada com os dispositivos móveis com aplicativos J2ME, ou Android onde o desenvolvimento para o sistema operacional é feito na linguagem Java, proporcionando uma fácil migração desse projeto para um dispositivo móvel. Outro motivo pela escolha da linguagem Java é a facilidade do uso de classes nativas para a manipulação de imagens, e a existência de várias APIs para a manipulação de imagens, OCR e inteligência artificial. Alguns exemplos de APIs são ImageJ e JAI para manipulação de imagens , Tesseract e JavaOCR para OCR e Weka para inteligência artifical.

A principal classe para manipulação de imagens na linguagem Java é a BufferedImage, que possui toda uma estrutura para trabalhar com imagens em escalas de cinza ou coloridas. Um exemplo básico dessa estrutura está sendo mostrado na Figura 38.

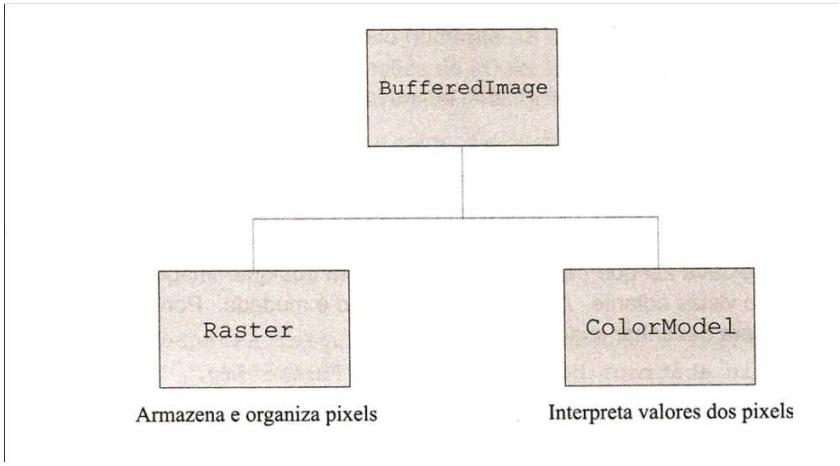


Figura 38 - Estrutura básica da classe *BufferedImage*
FONTE: MIRANDA, 2006, p. 38

Como pode ser visto no desenho, a BufferedImage possui duas classes internas, uma para armazenar dos pixels (Raster), e outra para interpretar os valores de cada pixel(ColorModel).

4.2.2. API Image J

O ImageJ é uma API de processamento digital de imagens para a linguagem Java. Ele foi uma adaptação do NIH imagem, que foi um software de processamento de imagens de domínio público para o Macintosh. Ele foi desenvolvido no departamento *Research Services Branch* (RSB) do *National Institute of Mental Health* (NIMH), parte do *National Institutes of Health* (NIH) que fica no estado de Maryland nos Estados Unidos. Ele foi substituído pelo ImageJ que roda em Macintosh, Linux e Windows. Essa API possui uma grande variedade de recursos como filtros de detecção de bordas, binarização, esqueletização, ferramentas de histograma e equalização. Nesse projeto a API ImageJ foi utilizada para fazer a esqueletização das imagens.

Além de uma API o ImageJ também é uma ferramenta de processamento de imagens com interface gráfica que pode ser instala ou ser executada direto do arquivo .jar .

A classe ImagePlus representa uma imagem no ImageJ, ela herda a classe ImageProcessor que é uma classe abstrata que fornece métodos para trabalhar os dados da imagem (BAILER,2006).

4.3. Realce

Essa seção fala sobre as técnicas utilizadas no pré-processamento da imagem e como essas técnicas são utilizadas. Essas técnicas não estão sendo utilizadas somente antes da localização da placa, mas também, antes da localização dos caracteres e também durante a localização da placa que é dividia em duas partes, como será explicado nas próximas seções.

4.3.1. Filtro passa-alta

O filtro passa-alta escolhido para ser utilizado no sistema foi o Sobel, por ser o que obteve os melhores resultados, deixando as retas mais grossas e assim a placa em maior evidência comparada com o os outros filtros de detecção de bordas como está sendo mostrado na Figura 39.

(a)

(b)

(c)

Figura 39 - Detecção de bordas de uma imagem com uma placa veicular, sendo Prewitt (a), Roberts (b) e Sobel (c).

O algoritmo utilizado na filtragem de detecção de bordas foi baseado no livro de Miranda (2006), onde é mostrado como realizar uma convolução de uma máscara 3x3 utilizando as bibliotecas do Java para processamento digital de imagens.

Após a realização de alguns experimentos foi constatado que a localização da placa seria dificultada utilizando o filtro de detecção de bordas inteiro, pois os carros com pára-choque que possuem uma grande quantidade de linhas horizontais confundiam a localização da placa mesmo utilizando o passo seguinte que é a esqueletização. Então o algoritmo foi alterado para fazer somente a filtragem vertical deixando a placa em maior destaque em comparação com o restante da imagem, como é exibido na Figura 40. O código desenvolvido se encontra no Apêndice A.

Figura 40 - Filtro Sobel Vertical

4.3.2. Binarização

Foram tentados três métodos para binarizar a imagem, um utilizando um limiar fixo que é o tom de cinza 125 implementado utilizando as bibliotecas do Java para processamento digital de imgens, outro utilizando um método interno da API ImageJ e um útimo também utilizando as bibliotescas do Java para processamento digital de imagen. O que obteve o melhor resultado foi o método da API ImageJ. É importante ressaltar que foi tentado acessar os métodos mais avançados de limiarização no ImageJ, como Otsu ou Huang, porém so foi possível implementar o algoritmo padrão do ImageJ. Isso ocorreu pela complexidade da API e a falta de materiais falando sobre o assunto. A Figura 41 mostra quais os métodos disponíveis na ferramenta, lembrando que é pela ferramenta e não API, pois não foram encontrados materias mostrando como utilizá-los através da API.

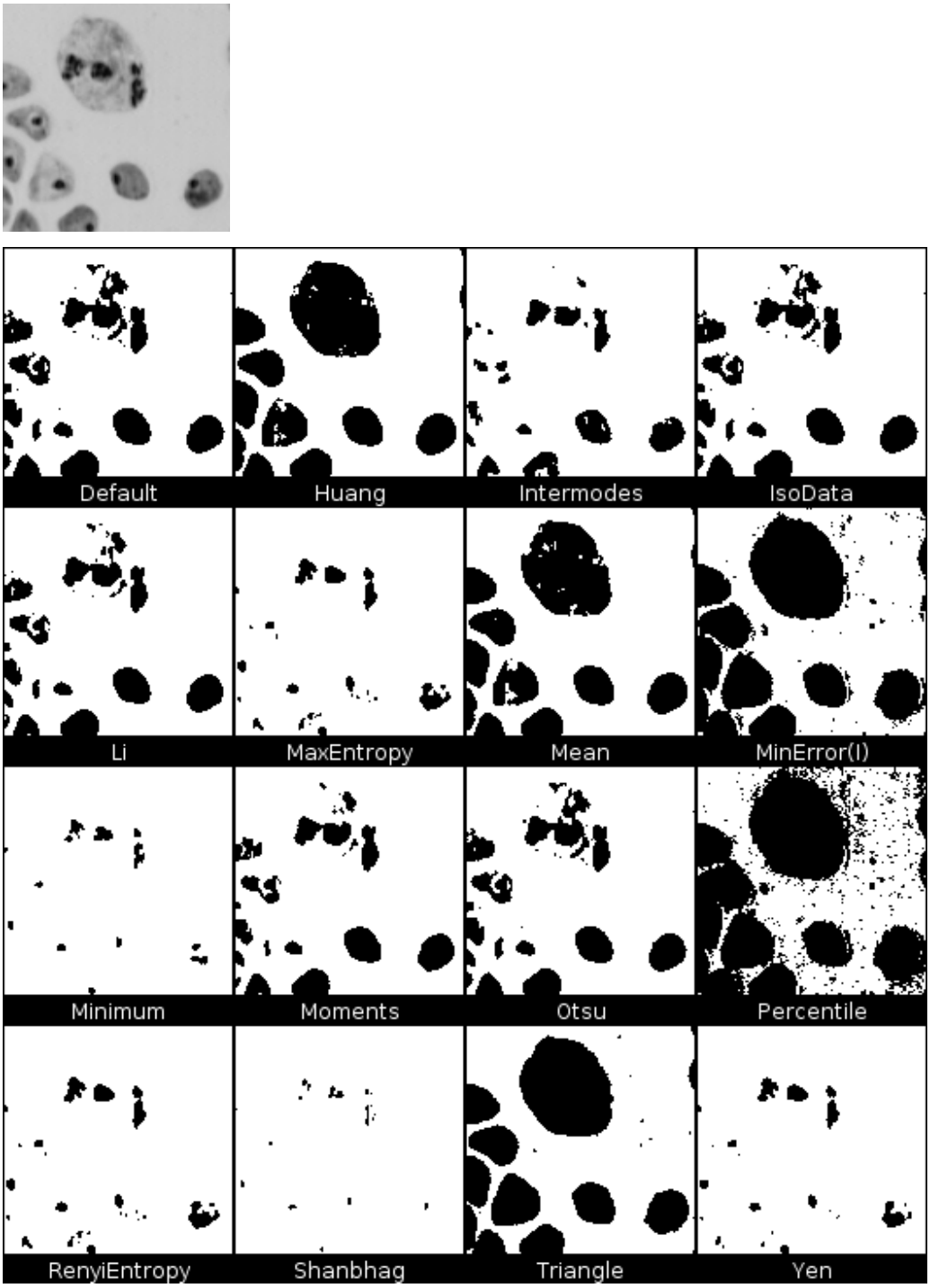


Figura 41 – Imagem original e a suas limiarizalções utilizando a ferramenta ImageJ

Segundo LANDINI(2011) o algoritmo utilizado como padrão no ImageJ é o algoritmo IsoData ou Iterativo Intermeans.

O Apêndice B mostra os algoritmos para binarização. A Figura 42 mostra a binarização utilizando a API ImageJ.

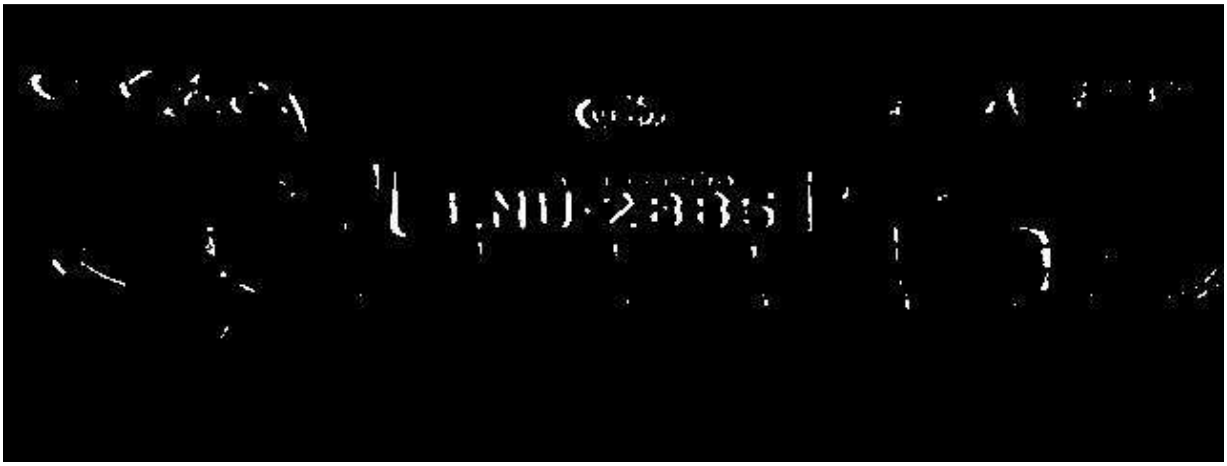


Figura 42 - Binarização da imagem após a aplicação do filtro Sobel vertical

4.3.3. Esqueletização

A esqueletização é o próximo passo depois do filtro Sobel e da binarização. Ela é útil no processo de localização da placa, pois mesmo com o filtro vertical, a localização da placa ainda é difícil devido aos objetos como o farol do carro ou símbolos como a marca do carro, que podem ser confundidos com uma placa. A idéia da esqueletização é fazer todas as linhas da imagem ficarem com a mesma espessura.

Neste trabalho deixar todas as linhas com a mesma espessura é essencial para a localização da placa, pois como será mostrado na próxima seção, a localização da placa será feita percorrendo uma máscara do tamanho da placa que procura na imagem o local onde a máscara se encaixe e que possua a maior quantidade de pixels brancos. Se a esqueletização não for aplicada quando um farol for encontrado e tiver as linhas mais grossas que uma placa após a filtragem passa-alta vertical, e esse farol tiver as mesmas dimensões da placa, então ele será confundido com uma placa, mas se nessa mesma imagem for aplicada a esqueletização então o problema será resolvido, pois a máscara irá se encaixar na região da placa. Um exemplo da esqueletização é mostrado na Figura 43.

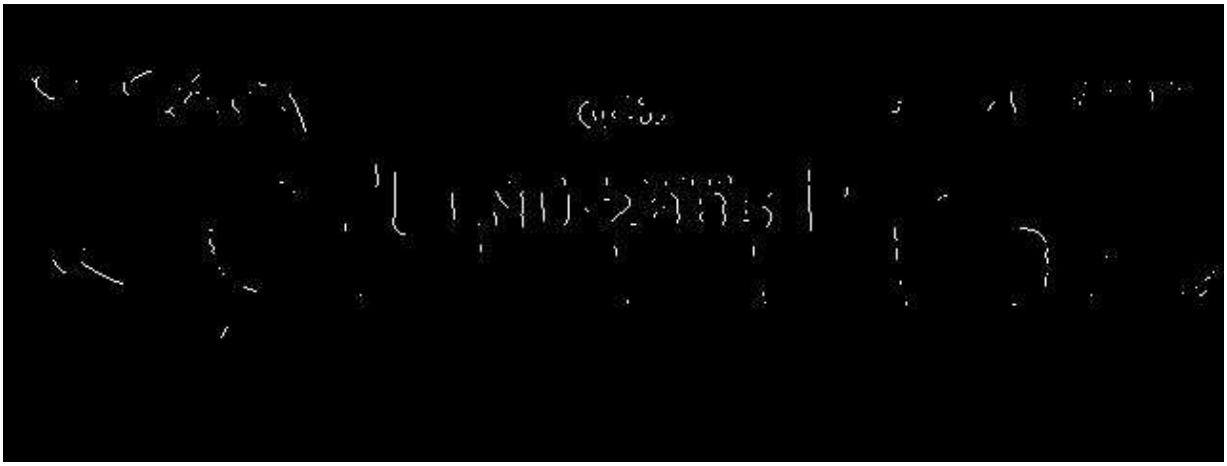


Figura 43 - Esqueletização de uma imagem contendo uma placa veicular após a binarização.

O algoritmo para esqueletização utiliza a API ImageJ. O Apêndice C mostra o código desenvolvido para realizar a esqueletização. No algoritmo é realizada a filtragem passa-alta vertical, depois é realizado uma binarização da imagem através da API ImageJ , que possui um método interno para binarização, então é realizada a esqueletização da imagem.

4.3.4. Equalização

A equalização foi implementada tanto para ser utilizada no pré-processamento para localização da placa quanto na localização dos caracteres. A idéia de utilização era fazer com que as placas muito claras ou muito escuras ficassem em uma tonalidade mediana. A equalização se mostrou eficiente em alguns casos, mas em outros ela piorou a imagem fazendo a placa perdesse detalhes, e por isso foi descartada das técnicas utilizadas para localização das placas e dos caracteres. A Figura 44 mostra um exemplo de imagem que perdeu detalhes utilizando a esqualização. O Apêndice D mostra os algoritmos e classes utilizadas para a equalização.

(a)

(b)

Figura 44 - Imagem normal (a). Placa perdendo detalhes após equalização (b).

4.4. Localização da Placa

Essa seção mostra quais foram os métodos usados para a localização da placa, não só os métodos que funcionaram, mas também os métodos que não deram certo. A localização da placa é dividida em duas etapas, uma que localiza um ponto dentro da placa e um que delimitada a região de busca a partir desse ponto e depois que localiza a placa.

4.4.1. Encontrar Ponto Dentro da Região da Placa

A primeira etapa para localizar a placa é a encontrar um ponto dentro da região da placa. Para isso foram testados vários métodos antes de se chegar a uma solução que apresentasse bons resultados, que é utilizar com pré-processamento o Filtro Sobel vertical e depois a esqueletização da imagem. Foi utilizada uma idéia básica em todos os métodos que é percorrer a imagem com

uma máscara um pouco maior que o tamanho médio da placa fazendo a média dos níveis de intensidade, e depois disso é procurado o pixel com maior intensidade na imagem. Como as placas não estão todas com o mesmo tamanho no banco de imagens, apesar de serem imagens de uma cancela onde a distância é praticamente a mesma para todos os veículos, foi escolhida uma máscara com largura de 240 por uma altura de 40 pixels, que é um tamanho um pouco maior que média das placas.

O primeiro método testado foi utilizar somente o filtro de detecção de bordas Sobel para depois encontrar o pixel com maior nível de cinza. O resultado não foi muito preciso, pois havia muitos detalhes na imagem e os pixels com maior intensidade foram localizados muitas vezes em pára-choques e faróis de carros. A Figura 45 mostra um exemplo da utilização desse método.

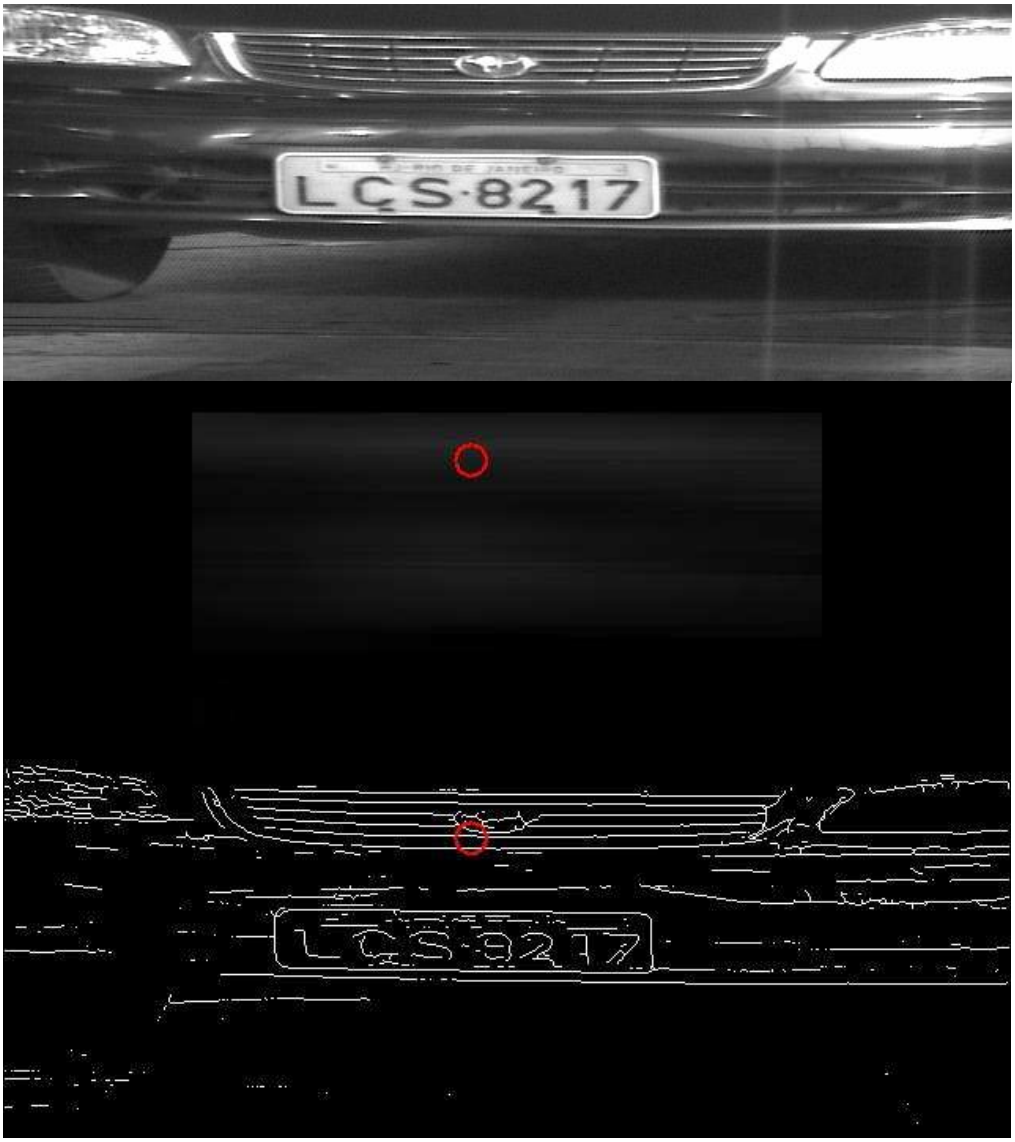
(a)

(b)

(c)

Figura 45 - Imagem normal (a). Média da imagem utilizando o filtro Sobel (b). Imagem com filtro Sobel e ponto com maior intensidade encontrado o pára-choque (c)

O segundo método testado foi utilizar o filtro Sobel, e a esqueletização da imagem, para que as linhas mais grossas como em faróis ou pára-choques não atrapalhassem na localização do pixel de maior intensidade. Mas o resultado também não foi satisfatório, pois devido à grande quantidade de linhas horizontais em alguns pára-choques o pixel de maior intensidade ainda não foi localizado na placa. A Figura 45 mostra um exemplo.



(a)

(b)

(c)

Figura 46 - Imagem normal (a). Média da imagem utilizando o filtro Sobel e esqueletização (b). Imagem com filtro Sobel e esqueletização e ponto com maior intensidade encontrado o pára-choque (c)

O terceiro método foi utilizar o filtro Sobel, mas somente a matriz vertical. A idéia de usar somente a matriz vertical é para não ter problemas com as linhas verticais se sobressaindo sobre a

placa. Esse método obteve resultados muito bons, mas ainda teve alguns problemas para encontrar a placa em alguns casos onde o símbolo da marca do automóvel atrapalhou na localização como é mostrado na Figura 47.

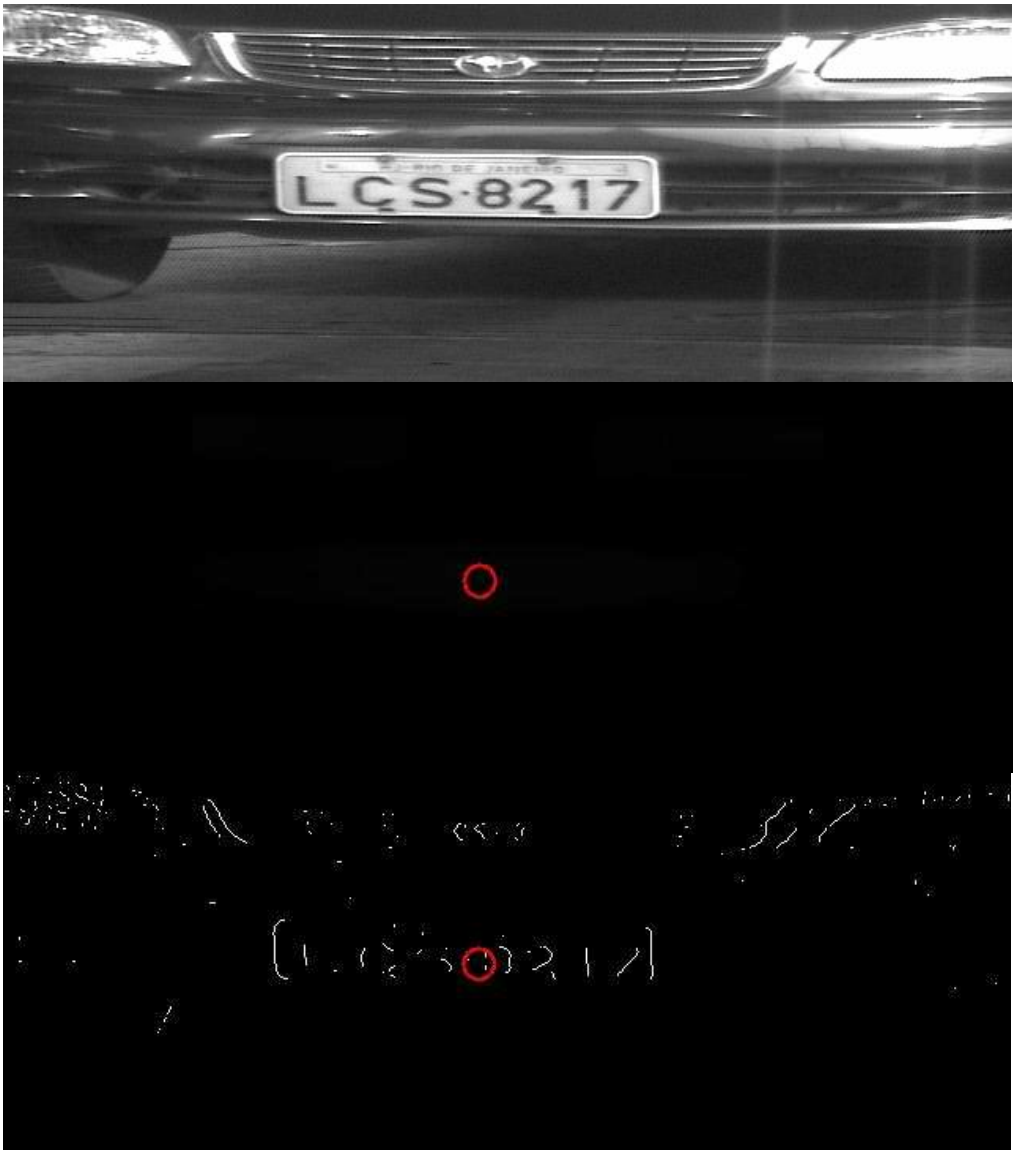
(a)

(b)

(c)

Figura 47 - Imagem normal (a). Média da imagem utilizando o filtro Sobel somente com a máscara vertical (b). Imagem com filtro Sobel somente com a máscara vertical e ponto com maior intensidade encontrado as linhas verticais do pará-choque (c)

O quarto método foi utilizar o filtro Sobel, com a matriz vertical, e em seguida a esqueletização. Esse método obteve os melhores resultados comparado com os outros dois realmente localizando um ponto dentro da placa. A Figura 47 mostra um exemplo. O código utilizado para realizar a delimitação da região da placa se encontra no Apêndice E.



(a)

(b)

(c)

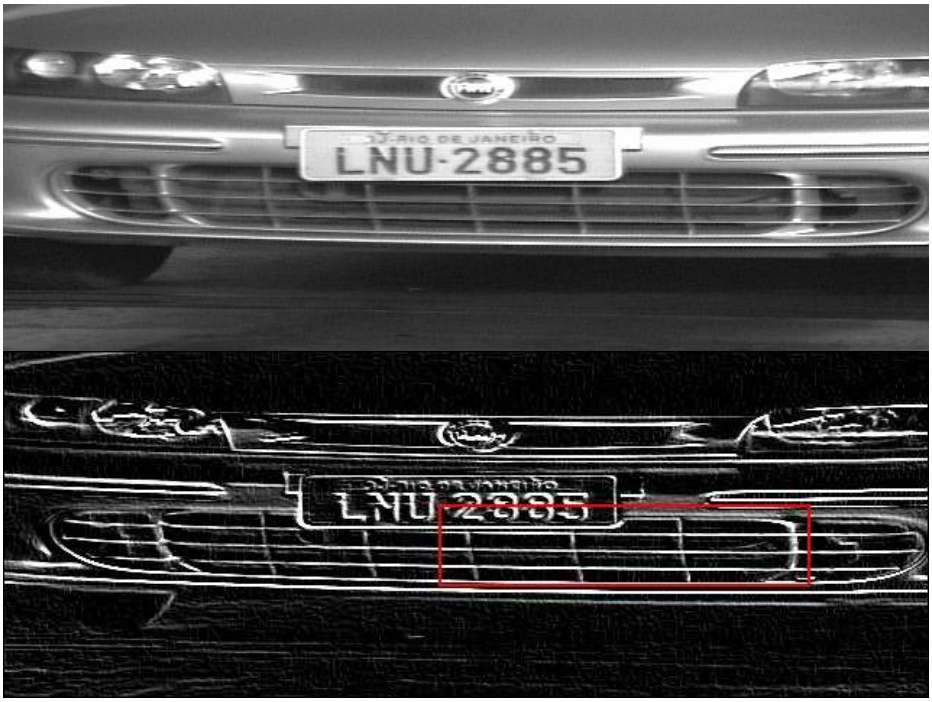
Figura 48 - Imagem normal (a). Média da imagem utilizando filtro Sobel somente com a matriz vertical e esqueletização (b). Imagem com filtro Sobel somente com a matriz vertical e esqueletização e ponto com maior intensidade encontrado no centro da placa (c)

O método que obteve o melhor resultado foi que utiliza filtro sobel vertical combinado com a esqueletização, portanto foi o empregado no sistema.

4.4.2. Localização da Região da placa

Após encontrar um possível ponto dentro da região da placa, a sua localização é feita delimitando uma região ao redor do ponto encontrado. Nessa janela criada foram testados vários métodos para encontrar a placa, varrendo a imagem com uma máscara e calculando a somatória da máscara de cada pixel, onde existir a maior somatória é o local onde a placa está. Foram utilizadas duas máscaras, uma oca com as bordas tendo o valor 1 e o seu interior 0 e uma com todos seus valores iguais a 1. Foram tentados junto com essas máscaras a esqueletização e o filtro Sobel alta. O tamanho da placa utilizado nesse método foi o mesmo utilizado no método de delimitação da região da placa. Após encontrar o ponto com maior somatória, o tamanho da placa é redimensionado em 5 pixels em largura e altura para que não seja descartada alguma parte da placa que fique fora do alcance da máscara , então essa região é salva em uma nova imagem. O algoritmo utilizado nesse método se encontra no Apêndice F.

O primeiro método testado foi passar um filtro Sobel na imagem utilizando as duas matrizes. Após esse pré-processamento é procurado o ponto onde uma máscara oca com borda de 4 pixels possui maior somatória. Esse método não se mostrou muito eficaz, pois a taxa de erro foi muito grande, e as linhas horizontais do pára-choque confundiram a localização da máscara assim como na etapa de localização de um ponto dentro da placa. A Figura 49 mostra um exemplo.



(a)

(b)

Figura 49 - Imagem normal (a). Placa parcialmente localizada utilizando uma máscara oca com borda de 4 pixels após a utilização do filtro Sobel

O segundo método testado foi passar um filtro Sobel na imagem utilizando as duas matrizes e depois a esqueletização. A esqueletização foi para tentar afinar as linhas do pára-choque do carro fazendo com que a máscara tenha mais facilidade para encontrar a placa do carro. Também foi utilizada uma máscara com borda de 4 pixels, porém os resultados não foram muito satisfatórios. A Figura 50 mostra um exemplo.

(a)

(b)

Figura 50 - Imagem normal (a). Placa parcialmente localizada utilizando uma máscara oca com borda de 4 pixels após a utilização do filtro Sobel e esqueletização

O terceiro método foi realizar uma filtragem Sobel com a matriz vertical e depois a esqueletização seguindo a idéia da etapa de localização de um ponto dentro da placa. Após esse pré-processamento é procurado o ponto onde uma máscara oca com borda de 15 pixels possui maior somatória. Foi utilizada uma máscara com as bordas mais espessas, pois com os testes foi constatado que quanto maior as bordas melhores são os resultados. Esse método obteve bons resultados, porém grande parte das placas foram encontradas deixando os caracteres fora do centro da máscara, o que pode dificultar na localização dos caracteres dependendo do método escolhido. A Figura 51 mostra um exemplo desse método.

(a)

(b)

Figura 51 - Placa localizada com os caracteres fora do centro da máscara utilizando uma máscara oca de 15 pixels após a utilização do filtro Sobel com matriz vertical e esqueletização (a). Após localização, máscara desenhada na imagem original(b)

Por fim, foi tentado um último método utilizando uma máscara com todos os pixels tendo o seu valor igual a 1 e o pré-processamento na imagem utilizando Sobel com a matriz vertical e depois a esqueletização. Os resultados mostraram serem superiores aos outros métodos testados, pois a maioria das placas foram localizadas no local correto e tendo os caracteres no centro. Esse método foi o utilizado no sistema final por ser mais simples, não necessitando da esqueletização , e possuir maior taxa de acerto. A Figura 52 mostra um exemplo.

(a)

(b)

Figura 53 - Placa localizada utilizando uma máscara com todos os valores iaguais a 1 após a utilização do filtro Sobel com matriz vertical e esqueletização(a). Após localização, máscara desenhada na imagem original(b)

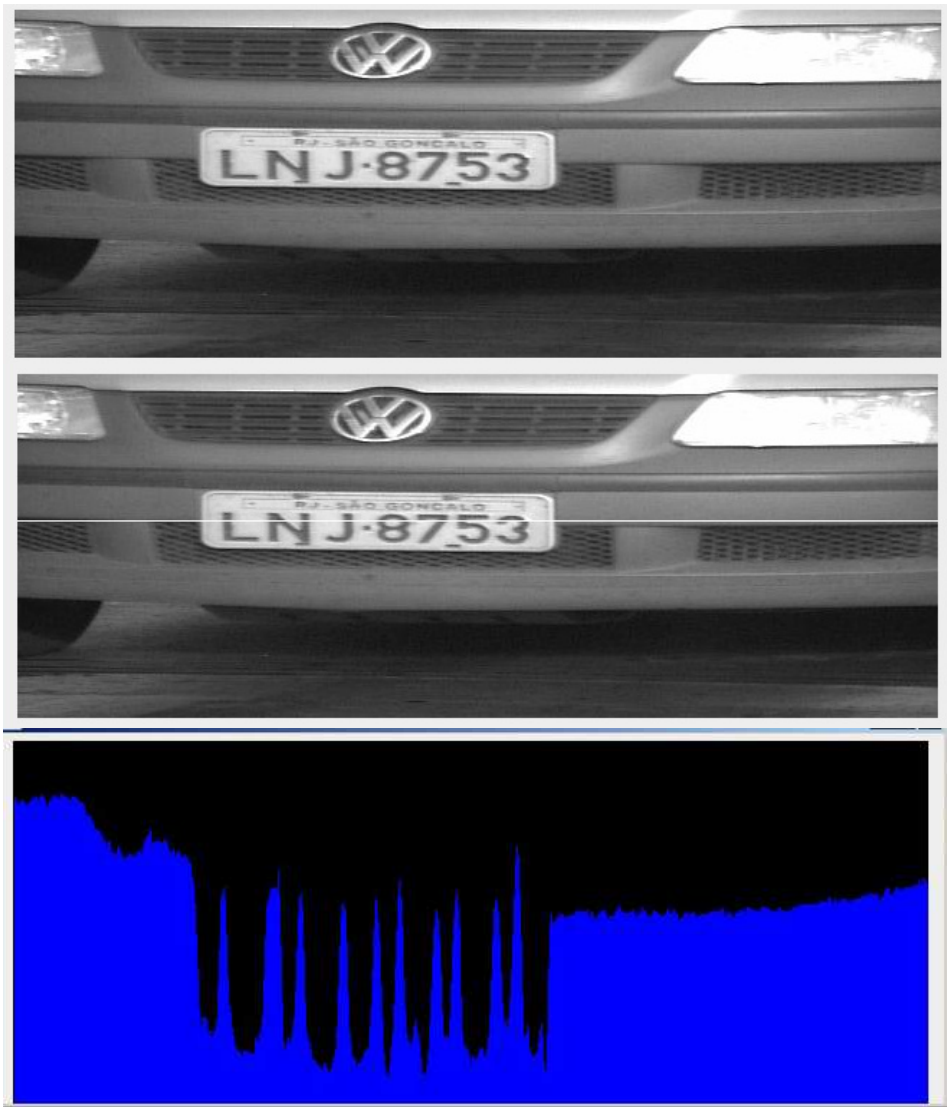
Delimitar a região da placa antes de procurar a placa é útil, pois delimitando a região são eliminados faróis e símbolos da imagem que podem ser confundidos com a placa o que não ocorreria se fosse utilizado o método de reconhecimento sem a delimitação.

4.5. Localização dos Caracteres

Essa seção mostra os métodos que funcionaram e também os que não funcionaram para a localização dos caracteres. Após a localização da placa, é necessário encontrar os caracteres. A região da placa que foi localizada é salva em uma imagem, em cima dessa nova imagem é feita a localização dos caracteres, que foi feita utilizando duas etapas, uma que delimita a região possível onde os caracteres se encontram e outra que segmenta os caracteres.

4.5.1. Delimitação da região dos caracteres

A delimitação da região dos caracteres foi feita utilizando a mesma idéia do algoritmo de análise de variação tonal ou assinatura da placa do artigo de SOUZA *et al* (2006). A variação tonal procura localizar a “impressão digital” da placa fazendo uma varredura seqüencial na imagem. A Figura 53 mostra como é a assinatura de uma placa ao traçar uma reta no meio da placa.



(a)

(b)

(c)

Figura 53 - Imagem original (a). Imagem com uma reta traça no meio da placa (b). Variações da região onde a reta foi traçada, sendo que quanto mais alto o pico mais escuro é o seu valor

No algoritmo implementado no projeto a região onde os caracteres estão é delimitada buscando a região onde existe a maior quantidade de transições na imagem. Para encontrar essa região, a imagem é percorrida no sentido vertical. Nessa etapa não foi utilizado nenhum pré-processamento. Para selecionar os pixels que estão na região da transição foi feito um algoritmo que procura a cada 5 pixels, pixels que possuam uma diferença de intensidade de 60 tons de cinza entre o pixel anterior e o posterior a esse espaço de 5 pixels. Os pixels que tiverem a diferença de intensidade só serão aceitos se estiverem entre o maior e menor espaço possível entre dois caracteres. No algoritmo o maior espaço utilizado foi de 38 pixels e o menor foi de 18. Além dos pixels terem que estar entre esses espaços mínimos e máximos com uma diferença de 60 níveis de cinza a cada 5 pixels, existe uma quantidade máxima e mínima de pixels por linha da imagem. No algoritmo está sendo utilizado o valor de máximo de 28 pixels por linha e mínimo de 7 pixels por linha. Após encontrar os pixels são traçadas duas retas, uma sob o valor do pixel de menor valor em y e outro sobre o pixel de maior valor em y. As retas são traçadas com uma taxa de erro para cima e para baixo para evitar que as retas sejam traçadas em cima dos caracteres ou mesmo percam parte deles.

A Figura 54 mostra a seqüência de passos para delimitar a região dos caracteres. O primeiro passo é localizar a placa, em seguida é analisada a variação de cada linha da placa localizada, os pixels que possam ser a variação de um caractere para o fundo branco são marcados. Agora a região entre o primeiro e último pixel encontrado é marcada como a possível região do caractere. As classes e algoritmos utilizados para a delimitação da região dos caracteres se encontram no Apêndice G.

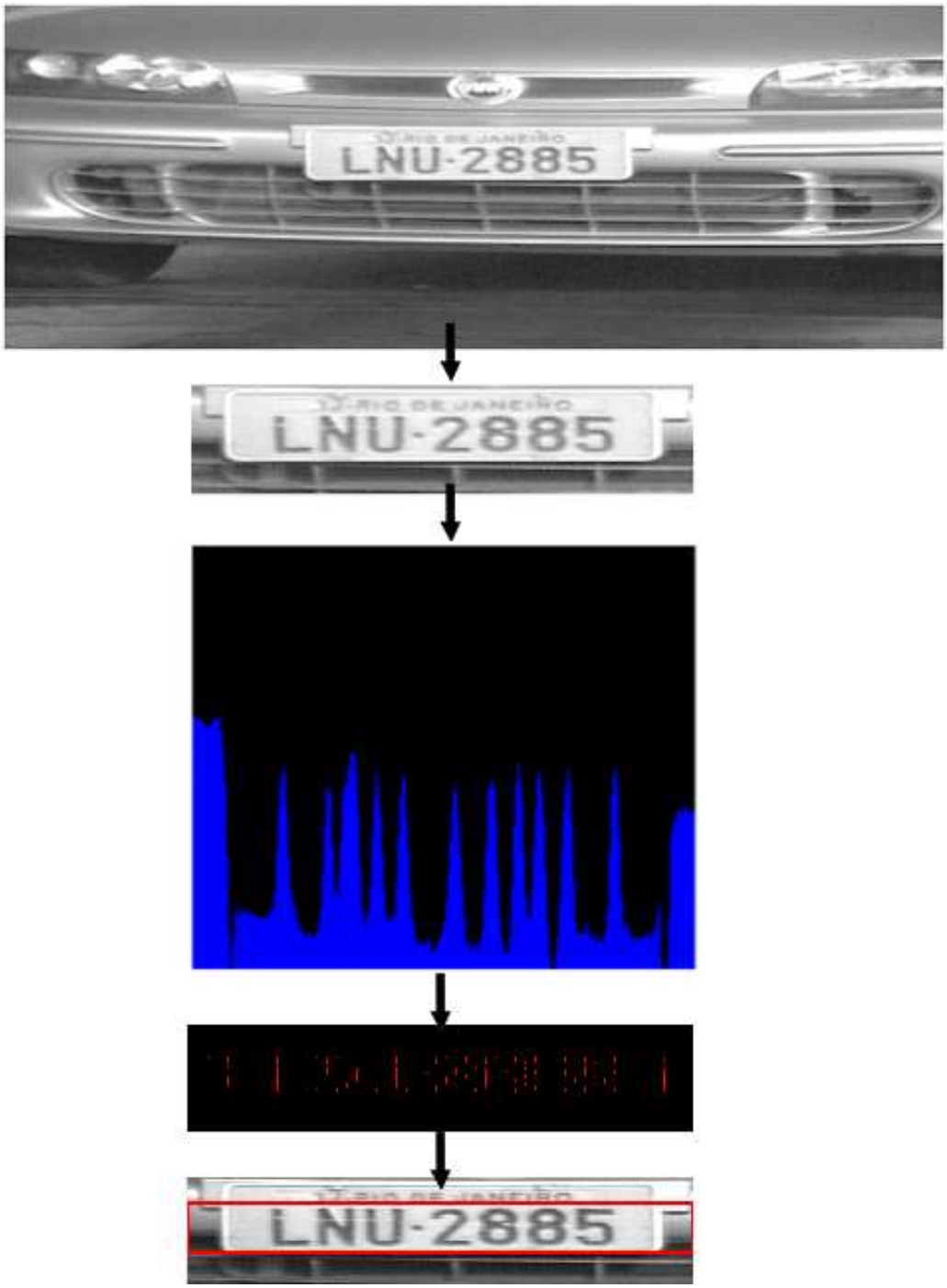


Figura 54 - Seqüência de passos para delimitar a região dos caracteres. Localizar a placa, analisar a assinatura, marcar pixels de transição, delimitar região dos pixels encontrados

4.5.2. Segmentação dos caracteres

Para a segmentação dos caracteres foram testados dois métodos, buscando variações na vertical na região delimitada dos caracteres, e analisando o gráfico da somatória da região delimitada dos caracteres. Os algoritmos desses métodos se encontram no Apêndice G.

O método de análise do gráfico varre a região delimitada com uma máscara do tamanho da metade de um caractere apróximadamente. Esse método não utiliza nenhum pré-processamento na imagem. No algoritmo foi utilizada uma máscara de 5 pixels de largura e o tamanho da região delimitada em altura. Ao percorrer a região delimitada é feita uma somatória de todos os pixels dentro da máscara. Os valores vão sendo salvos, e depois são convertidos para uma escala entre 0 e 255. Sendo *T* o valor da somatória da região da máscara, *Max* o maior valor que se deseja ter no gráfico que vai assumir o valor de 255, P_{mim} e P_{max} os valores da menor e maior somatória, e *TNovo* o novo valor que será utilizado para montar o gráfico, a conversão para uma escala entre 0 e 255 pode ser expressa pela fórmula :

$$TNovo = \frac{Max * (T - P_{max})}{(P_{mim} - P_{max})}$$

O gráfico formado pela conversão de escala é mostrado na Figura 55, pode se observar que cada pico no gráfico representa um caracter. O próximo passo é tentar dividir esses picos em caracteres.

(a)

(b)

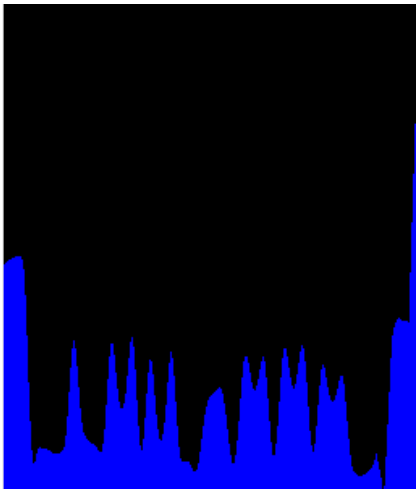


Figura 55 - Região dos caracteres delimitada (a). Gráfico da somatória da região delimitada (b)

Observando o gráfico e a placa, fica visível que para cada caractere existe um pico, exatamente na mesma região em que se encontra o caractere. Cada pico do gráfico representa a mesma posição em X da região delimitada dos caracteres, e também mostra a quantidade de valores escuros naquela região. Para separar os caracteres utilizando os dados do gráfico foi feito um algoritmo parecido com o que busca as variações de intensidade. No algoritmo os elementos do gráfico são salvos em uma estrutura, cada elemento irá possuir uma posição em X e um valor entre 0 e 255, que foi calculado pela fórmula de escala. Os elementos são varridos com certo intervalo entre os elementos, o intervalo utilizado foi 5, em busca de picos que possam ser caracteres . Ao varrer os elementos serão consideradas possíveis regiões de caracteres somente os elementos que possuírem uma diferença mínima e máxima entre seus valores. Após encontrar todas as diferenças ou picos dentro do limite, são procuradas as que estão entre uma distância mínima e máxima para um possível caractere. Os valores utilizados foram diferença mínima 18, diferença máxima de 38, distância mínima 5 e distância máxima de 60. Porém os resultados não foram satisfatórios como é mostrado na Figura 56.

Figura 56 - Tentativa de localizar os caracteres através do método da análise do gráfico da somatória

Como esse método não funcionou muito bem foi tentado outro método, mais simples e com melhores resultados. Varrer a região delimitada dos caracteres na vertical, procurando onde existe maior variação.

Antes de utilizar essa varredura é feita uma binarização da placa utilizando a API ImageJ. A imagem é varrida na posição horizontal procurando em um intervalo de pixels, se existe alguma variação maior que a variação mínima de um caractere. Como está sendo utilizada a binarização , os valores dos pixels serão 0 ou 1 , e como algumas placas podem ficar com os

caracteres muito finos devido a binarização, foi escolhido variação mínima de 1 e intervalo entre os pixels de 1 também. Cada vez que são encontradas variações consecutivamente à medida que a imagem vai sendo varrida, a região dessas variações é armazenada em uma lista temporária. Quando é encontrada uma região sem variação, as variações armazenadas anteriormente podem ser consideradas um caractere. Essas regiões são unidas em uma imagem em uma lista de caracteres. A Figura 57 mostra a seqüência de passos realizados para segmentar os caracteres, a delimitação da região dos caracteres, a binarização, a localização das regiões com maior transição na vertical e por fim essas regiões são salvas em imagens para o reconhecimento.

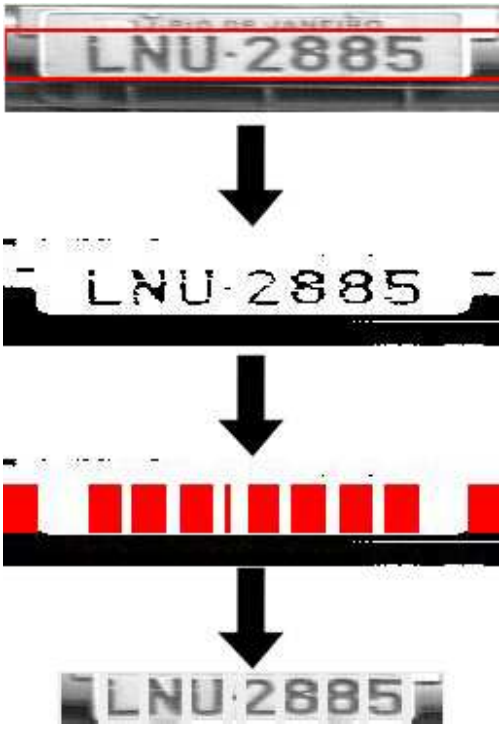


Figura 57- Seqüência de passos para localização dos caracteres utilizando o método que procura variações na vertical

Esse método teve maior taxa de acerto que o método anterior, apesar de salvar algumas partes da imagem que não são caracteres como pode ser visto na Figura 58. Esse erro pode ser tratado no reconhecimento dos caracteres, descartando as imagens que não forem caracteres.

4.6. Reconhecimentos dos Caracteres

Essa seção mostra quais foram os métodos usados para o reconhecimento de caracteres, caracteres obtidos através da segmentação, o qual gera uma lista de imagens no qual cada imagem passa por um processo de reconhecimento. O reconhecimento dos caracteres é dividido em duas partes, o treinamento e o reconhecimento dos caracteres. Os códigos para reconhecimento e treinamento estão no Apêndice H. Qual o conjunto de treinamento e de testes será explicado no capítulo de resultados , pois foram feitos vários testes com 12% e 20% da base de dados.

O módulo de reconhecimento de caracteres foi desenvolvido de forma que ele possa aprender conforme o usuário utilize o sistema, o mesmo é integrado ao sistema de localização de placas e caracteres para que o usuário possa através de um mesmo software localizar a placa, localizar os caracteres e reconhecê-los. A Figura 58 mostra a janela de treinamento e reconhecimento.

Figura 58- Janela que mostra os resultados do reconhecimento também serve para treinar as redes neurais

4.6.1. Treinamento

O treinamento consiste em selecionar uma amostra (imagem de um caractere obtido pela fase de segmentação) e transformá-la em uma estrutura de dados no qual possa ser tratado pela rede neural.

Para o reconhecimento dos caracteres foi utilizada uma Rede Neural de uma única camada, o que serve muito bem para classificar sistemas distintos. Como em nosso problema

temos vários caracteres, foi proposta a criação de várias redes neurais, sendo que cada rede neural é responsável pelo reconhecimento de um determinado caractere.

O treinamento de uma rede neural se dá pela imagem reconhecida pela mesma, caso a resposta de uma rede neural não seja verdadeira, esta rede precisa ser recalibrada, ou seja, precisar passar por mais um processo de treinamento.

Para realizar o treinamento de uma rede neural para reconhecer determinada imagem como um caracter, esta imagem passa por três etapas, essas são:

- Converter segmentação do caractere para cinza
- Converter segmentação do caractere para uma imagem de escala conhecida
- Converter segmentação do caractere para uma imagem esqueletizada

4.6.1.1. Converter segmentação do caractere para cinza

A conversão para uma imagem de escala cinza foi implementada considerando a possibilidade de se trabalhar com apenas uma camada de cor. Imagens coloridas contem três camadas de cores, Verde, Vermelho e Azul. A classe BufferedImage denota cada camada como banda. Abaixo um exemplo de como selecionar um pixel em uma posição [x, y].

```
bufferedImage.getRaster().getSample(x, y, 0)
```

Como pode ser visto o método getSample retorna o pixel em x e y de uma determinada banda.

4.6.1.2. Conversão da imagem segmentada do caractere para uma imagem de escala uniforme

É necessário que a imagem seja de um tamanho reconhecido e padronizado pela rede neural, sabemos que uma imagem é nada mais que uma matriz, onde cada posição dessa matriz contém a informação de um pixel, como a intensidade desse pixel. Cada pixel de uma imagem tem um peso associado a ele em uma rede neural. O tamanho da matriz é de 37 pixels de altura e 20 pixels de largura.

Em um perceptron é realizada a somatória da multiplicação dos pesos. Ao converter a imagem para entradas no perceptron, cada pixel irá representar uma entrada. Como um perceptron está preparado para um determinado número de entradas se uma imagem possuir um tamanho diferente do determinado apresentará uma falha, já que o perceptron está preparado para aquela quantidade de entradas.

4.6.1.3. Esqueletizar caractere

Como o sistema de rede neural de uma única camada serve unicamente para realizar a classificação de sistemas distintos, quanto mais amostras distintas para a rede neural, melhor é.

A esqueletização consiste em uma técnica de erosão que deixa apenas o esqueleto do caracter, sendo os pixels da imagem convertidos entre somente dois valores, “255” e “0”, onde 255 o pixel ativo e “0” é o pixel desativado.

No caso da Figura 59, a parte branca da imagem é a parte ativa, ou seja, o valor do pixel nessa área é 255, e a parte preta da imagem é a parte que não nos interessa.

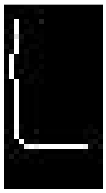


Figura 59 - Imagem esqueletizada um caractere L

4.6.1.4. Conversão da imagem para vetor

Como pode ser visto na implementação do perceptron, para o mesmo realizar o reconhecimento de uma determinada imagem, este precisa realizar a somatória da multiplicação dos pesos com a amostra.

Então, para que tal processo seja realizado, é gerado um vetor da seguinte forma, as linhas da matriz de 37 de altura por 20 de largura são unidas em uma única linha, onde o fim de uma linha é unido ao começo da linha seguinte, esse vetor será enviado para a rede neural.

4.6.1.4. Processo de treinamento do perceptron

Para o perceptron chegar à capacidade de “responder” se esse vetor enviado para ele é de algo que ele reconhecível são necessárias várias iterações e testes até que o erro seja o mínimo possível. Como já foi dito anteriormente esse processo é muito parecido ao treinamento de uma criança, quanto mais amostras para o reconhecimento, melhor é o desempenho do perceptron.

O algoritmo de treinamento funciona da seguinte maneira: o percetron possui vários pesos, onde cada peso representa um atributo da amostra, em uma imagem de 20 x 37, temos um vetor de 740 posições, onde cada posição contém um valor para o atributo, no perceptron possui um vetor de mesmo tamanho, então o resultado do processo de reconhecimento é a iteração desse vetor, somando o resultado da multiplicação entre o vetor peso na determinada posição pela mesma posição só que no vetor da amostra. Após essa somatória o resultado será separado em 0 e 1, onde valores maiores que 0 são representados como 1, e valores menores que 0 são representados por 0.

Para podermos solucionar este problema de reconhecimento de caracteres, foi desenvolvido um gerenciador de redes neurais.

Quando é solicitado o treinamento de uma imagem para poder ser reconhecida, o sistema percorre todas as redes neurais instanciadas e treina cada uma, quando a rede neural for a rede neural responsável pelo o reconhecimento da amostra em questão, essa amostra é definida de uma amostra verdadeira, ou seja, o valor desejado dessa amostra é 1, e quando a rede neural não for responsável pelo o reconhecimento da amostra em questão, é definido que é uma amostra falsa, onde o valor desejado é 0, ao total são feitas 36 iterações, onde cada iteração é o treinamento de uma rede neural, o treinamento da rede neural é feita até que a amostra seja reconhecida pela rede neural em questão, ou até que exeda o limite de 1000 iterações.

Em um treinamento de uma nova placa, por exemplo, “ABC-1234”, todas as redes são treinadas sete vezes, uma vez para reconhecer o caracter “A”, uma vez para reconhecer o “B” e assim por diante, onde cada redá rede neural é treinada até o reconhecimento do caracter, ou exceder o limite de 1000 iterações.

Abaixo segue o trecho do código que mostra como é feita essa distribuição dos treinamentos das redes:

```
//Itera todas as redes neurais, de A a Z, e de 0 a 9
            for (final RedeNeural neural :
BtnCaracterReconhecido.this.gerenciadorRedeNeural.getRedes()) {
                //Caso a rede neural seja a responsavel pelo reconhecimento do caracer, é treinada
como uma amostra verdadeira
                if (neural.getCaracterReconhecido() == resposta) {
                   int i = 0;
                   //Treina a rede até que reconheça ou que exeda 1000 iterações
                   while ((!neural.getPerceptron().treinarAmostra(amostraVerdadeira)) && (i++ <
1000)) {
                      ;
                   }
                } else {
                   int i = 0;
                  //Caso a rede neural não seja a responsavel pelo reconhecimento do caracer, é
treinada como uma amostra verdadeira
                   while ((!neural.getPerceptron().treinarAmostra(amostraFalsa)) && (i++ < 1000))
{
                      ;
                   }

                }
            }
```

Após o treinamento de todas as redes neurais são salvos no XML os novos pesos de cada perceptron, fazendo com que não se perca esses esforços.

4.6.2. Reconhecimento

O reconhecimento dos caracteres consiste em perguntar para todas as redes neurais instanciadas se alguma delas reconhece a amostra.

Como o sistema foi desenvolvido de maneira que cada rede neural reconheça um caractere, mais de uma rede pode reconhecer uma determinada amostra, um exemplo disso é o caractere “O” e “0”, onde até mesmo para o ser humano, é difícil de ser diferenciado.

Para saber se uma rede neural reconhece uma determinada amostra, essa amostra em forma de vetor, como já foi dito antes, é confrontada com cada peso de cada perceptron, caso retorna verdadeiro significa que essa rede neural reconhece a amostra em questão, caso a resposta seja falsa, isso significa que essa rede neural não reconhece essa amostra, abaixo segue algoritmo em java que realiza o cálculo:

```
public int getNet(final double valor) {
      return valor > 0 ? 1 : 0;
   }


public double getSaidaCalculada(final int[] valores) {
```

```
      return this.pesoBias +
this.configuracaoPerceptron.getFormatedDouble(this.somatorioPesosValores(valor
es));
   }


private double somatorioPesosValores(final int valores[]) {
      double resultado = 0;
      for (int i = 0; i < valores.length; i++) {
         resultado += this.pesos[i] * valores[i];
      }
      return resultado;
   }
```

Como pode ser visto, existe uma função getNet, o qual retorna 1 caso tenha reconhecido e 0 caso contrário.

O sistema foi desenvolvido de forma que aprenda conforme o uso do mesmo, de forma dinâmica, onde a cada imagem é testada nas redes neurais e também podem ser treinadas com novas imagens.

Quando o sistema encontra os caracteres pela primeira vez e as redes neurais ainda não estão treinadas, não é possível recohecer os caracteres, pois nenhuma rede está calibrada para isso. O processo de treinamento é um processo de treinamento monitorado, onde, o usuário treina a rede indicando o resultado correto e testa ao mesmo tempo.

Como é visto na Figura 60, não foi possível reconhecer nenhum caracter, pois foi a primeira vez que o sistema é utilizado.

Figura 60 - Reconhecimento de caracteres sendo executado pela primeira vez

Ao realizar os treinamentos das amostras uma vez e alguns caracteres foram reconhecidos, porém não todos, pois essas redes neurais precisam ser mais treinadas.A Figura 61 ilustra essa situação.

Figura 61 - Mesma placa aberta logo após as redes serem treinadas

Para a bateria de testes foram usadas 49 imagens onde os caracteres puderam ser segmentagos automaticamente, como pode ser visto a figura abaixo, apartir da terceira imagem as redes já começam a apresentar alguns resultados.

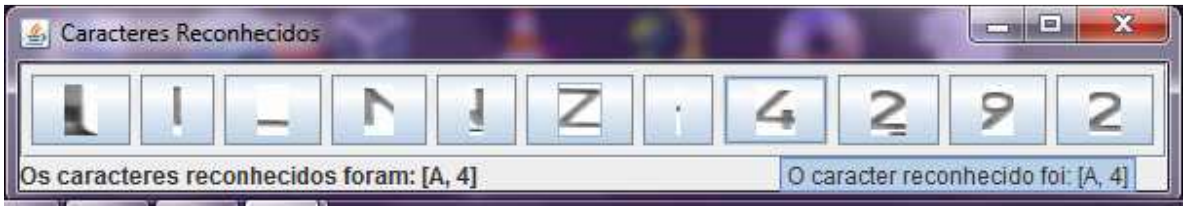


Figura 62 - O caractere 4 foi identificado por duas redes neurais, a rede neural A (errada) e a rede neural 4

Capítulo 5. Resultados e Conclusões

Esse capítulo irá mostrar quais foram os resultados obtidos pelos métodos utilizados no desenvolvimento do sistema. Serão mostrados os pontos fortes e fracos das técnicas utilizadas

Os resultados também serão comparados com resultados de outros sistemas para varificar se o sistema apresentou resultados satisfatórios ou não.

Também serão discutidos possíveis trabalhos que podem ser realizados em cima desses, quais foram as principais dificuldade e a conclusão final.

5.1. Resultados da Localização da Placa

Nessa seção serão mostrados todos os resultados obtidos nos métodos utilizados para localização da placa, tanto na localização do ponto dentro da placa quanto na localização da placa em si. É importante ressaltar que os métodos forneceram ou apresentaram bons resultados tanto para placas brancas quanto para placas vermelhas.

5.1.1. Resultados da Localização de um Ponto Dentro da Placa

A Tabela 10 mostra quais foram os resultados para os diferentes tipos de pré-processamentos para que seja realizada a localização de um ponto dentro da placa, utilizando a base de imagens citada no capítulo de desenvolvimento. Serão consideradas somente as 75 imagens com placas em bom estado.

Tabela 10 - Resultados para diferentes tipos de pré-processamentos na localização um ponto dentro da placa

Pré-processamento	Acertos	Erros	Taxa de acerto
Sobel	53	22	71 %
Esqueleto	40	35	53%
Sobel somente matriz vertical	66	9	88%
Sobel somente matriz vertical e esqueleto	71	4	95%

Como pode ser visto na Tabela 10, a taxa de acerto do algoritmo que busca um ponto dentro da placa é de 95% se utilizado como pré-processamento o filtro Sobel somente com a máscara vertical e a esqueletização. O método funcionou para todas as 12 placas vermelhas, as placas que tiveram problemas estavam em tonalidades muito claras ou escuras.

5.1.2. Resultados da Localização da Placa

Os resultados dos diferentes tipos de pré-processamento e métodos utilizados na fase de localização da placa são mostrados na Tabela 11. Também serão consideradas somente as 75 imagens em boas condições de análise. Os métodos a seguir utilizaram o melhor método para localizar um ponto dentro da placa que é o que utiliza o filtro Sobel somente com a matriz vertical e esqueleto para poder continuar com a localização da placa.

Tabela 11- Resultados para diferentes tipos de pré-processamentos e métodos para localizar a placa

Pré-processamento	Tipo de máscara	Acertos	Erros	Taxa de acerto
Sobel	Máscara oca com bordas de 15 pixels	48	27	64%
Sobel	Máscara oca com bordas de 4 pixels	43	32	57%
Sobel e esqueleto	Máscara oca com bordas de 4 pixels	52	23	69%
Sobel somente matriz vertical	Máscara oca com bordas de 15 pixels	69	6	92%
Sobel somente matriz vertical e esqueleto	Máscara oca com bordas de 15 pixels	70	5	93%
Sobel somente matriz vertical	Máscara inteira com valores iguais a 1	71	4	95%

Como pode ser visto na Tabela 11, o método com melhor desempenho foi utilizar o filtro sobel somente com a matriz vertical com uma máscara com todos os valores iguais a 1. Mas os resultados comparados com os métodos que utilizam máscaras ocas com bordas de 15 pixels são muito próximos e portanto o resultado poderia ser diferente utilizando outras bases.

Comparando o melhor método para localização da placa com a tese de mestrado de CARVALHO (2006) que utiliza morfologia matemática para a localização da placa e que utiliza a mesma base em um dos testes, a taxa de acerto foi de 78%. Isso mostra que os resultados desse trabalho foram superiores em 3 dos dos métodos utilizados. Porém o trabalho de CARVALHO (2006) localiza várias placas ao mesmo tempo na imagem, enquanto os métodos que foram desenvolvidos nesse trabalho localizam apenas uma placa por foto. No melhor método 11 das placas vermelhas foram localizadas, 1 foi parcialmente localizada, e os outros 3 erros ocorreram devido ao erro do algoritmo anterior que localizou um ponto que não estava dentro da placa.

5.2. Resultados da Segmentação dos Caracteres

Essa seção mostra os resultados dos métodos utilizados para a segmentação dos caracteres.

5.2.1. Resultados da Delimitação da Região dos Caracteres

Na delimitação da região dos caracteres foram utilizadas as 100 imagens, testando o método para ver se o mesmo consegue identicar que não existem catacteres na imagem. O Gráfico 1 mostra quais foram os resultados da delimitação da região dos caracteres.

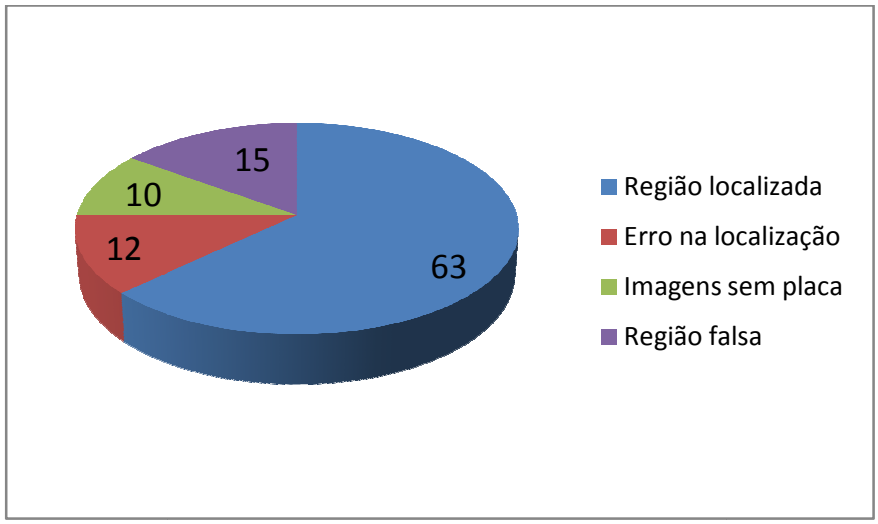


Gráfico 1 - Resultados da delimitação da região dos caracteres

Como pode ser visto no Gráfico 1 das 25 imagens sem placa o sistema identificou que 10 imagens não possuem caracteres , ou seja não possuem placas, 15 imagens(sem placas ou com placas sem condições de serem analisadas) foram falçamente identificadas com caracteres, 12 imagens com placa não tiveram seus caracteres delimitados corretamente e 63 imagens tiveram a região dos caracteres delimitada corretamente. Pode ser constatado então que a delimitação dos caracteres obteve um acerto de 73%, sendo 63% de delimitaçãoes corretas e 10% de imagens que foram constatadas sem placas. Das 12 placas que não tiveram seus caracteres delimitados corretamente, 4 são devido a não localização da placa.

5.2.2. Resultados da Segmentação dos Caracteres

Esse método foi utilizado com base no método de delimitação da região dos caracteres. A segmentação dos caracteres foi o método o que obteve o pior desempenho. O erro se deu na verdede devido ao método de binarização utilizado que não destacou os caracteres com muita nitidez, e também porque o método não funcionou nas placas de cor vermelha. O Gráfico 2 mostra os resultados da segmentação dos caracteres .

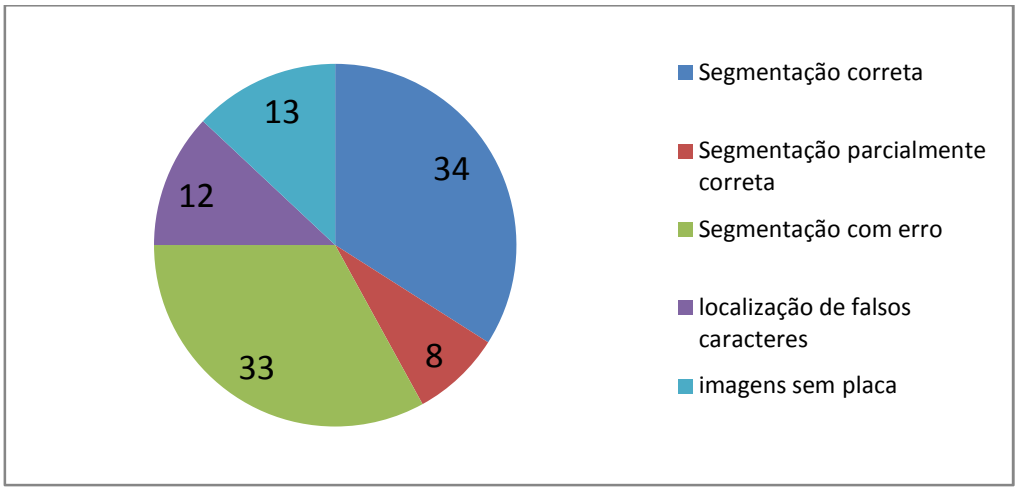


Gráfico 2 - Resultados da segmentação dos caracteres

Ao analisar o Gráfico 2 pode se perceber que das 100 imagens 34 tiveram os caracteres das placas dos carros segmentados , 8 tiveram os caracteres parcialmente segmentados onde foi considerado até 3 caracteres segmentados com erro, 33 placas com caracteres não segmentados corretamente ou não encontrados, 12 imagens sem placa onde foram encontrados caracteres , e 13 imagens sem placa onde não foram encontrados caracteres. A taxa de acerto pode considera como sendo de 47% somando as 34 segmentações corretas com as 13 identificações de ausência de placa na imagem. Mas se não for levado em consideração o detalhe de que o algoritmo não consegue segmentar caracteres de placas vermelhas, a taxa de acerto aumenta para 53%, como é mostrado no Gráfico 3.

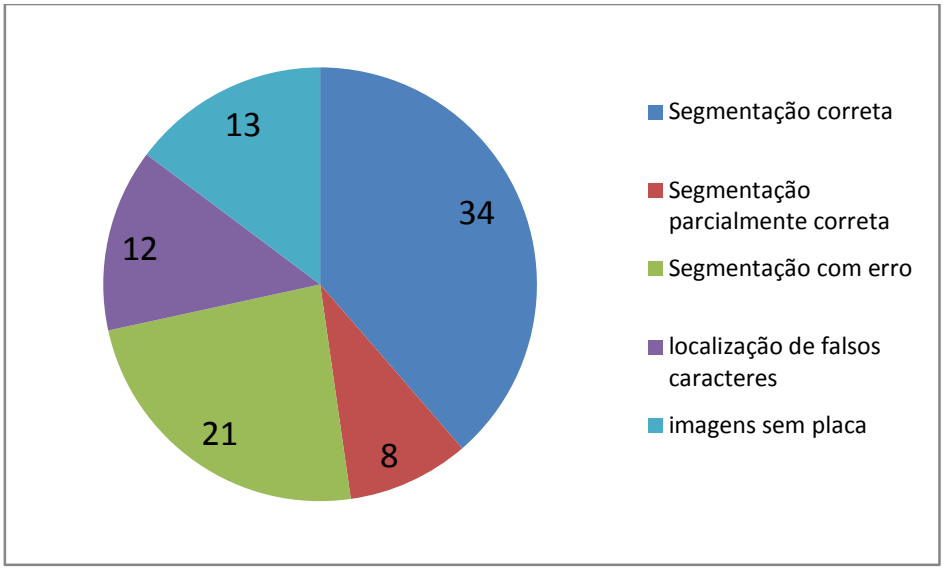


Gráfico 3 - Resultados da segmentação dos caracteres, sem considerar as placas vermelhas

Se ainda forem ignorados os erros do algoritmo anterior que são 12 regiões com erro na localização dos careceteres dos 53% a taxa de acerto sobe para 62%.

Unindo o tempo de processamento da localização da placa e sementação dos caracteres, o tempo de resposta do algoritmo é aproximadamente 4 segundos com a exibição de todos os dados na tela como está sendo mostrado no Apêndice G, e também aproximadamente 4 segundos somente com o processamento da imagem sem exibir nada na tela.

5.3. Resultados da identificação dos Caracteres

Essa seção mostra os resultados dos métodos utilizados para o reconhecimento dos caracteres.

O teste foi feito com 49 imagens, que obtiveram melhor resultado na segmentação dos caracteres. A Tabela 12 mostra os resultados com um treinamento de 12% dessa base, ou seja, 6 imagens.

Tabela 12 – Tabela de 12% de treinamento

Placas	12% TREINADO		
	TREINADO	RECONHECIDOS	RECONHECIDOS PARCIAL
LOH 8516	L, O, H,8,5,1,6		
LCD 7645		NENHUM	NENHUM
LNZ 4292		NENHUM	NENHUM
JLW 9464		6	NENHUM
KRD 7366		NENHUM	NENHUM
LNJ 8753		NENHUM	NENHUM
LNP 1417	L,N,P,1,4,1,7	L	NENHUM
LID 5816		L	NENHUM
DUR 7778		NENHUM	NENHUM
LNF 5873		L	8
LNR 3839		NENHUM	NENHUM
LIN 2148		NENHUM	NENHUM
KMV 8068	K,M,V,8,0,6,8	NENHUM	NENHUM
LNZ 7675		6	NENHUM
LNO 3968		NENHUM	8
LNP 7221		L	NENHUM
LND 3416		NENHUM	NENHUM
LNZ 3777		NENHUM	NENHUM
LNG 6631	L,N,G,6,6,3,1		
KNI 0508		NENHUM	NENHUM
KMU 0087		NENHUM	NENHUM
LOE 2022		NENHUM	NENHUM
LCS 8217		NENHUM	1
LBD 2319		NENHUM	NENHUM
CSE 9780	C,S,E,9,7,8,0		
LBU 5699		L	NENHUM
LAY 2596		6	NENHUM
LBL 9606		L,L	NENHUM
LKA 8614		6	1
LNI 2993		NENHUM	NENHUM
LOE 0836	L,O,E,0,8,3,6	L	
AAY 5127		1	NENHUM
LOA 2563		NENHUM	NENHUM
KND 8865		NENHUM	NENHUM
LNR 0491		NENHUM	NENHUM
KNP 9837		NENHUM	NENHUM
LAV 5233		NENHUM	NENHUM
KMG 6921		NENHUM	NENHUM
LBO 1230		NENHUM	NENHUM
KOD 7347		NENHUM	NENHUM
LNU 2885		L	NENHUM
LCN 7724		NENHUM	NENHUM
ABY 9198		NENHUM	NENHUM
GZA 2477		NENHUM	NENHUM
LAR 7207		NENHUM	NENHUM
LKI 6549		L	NENHUM
CTB 1569		NENHUM	NENHUM
LOF 3405		NENHUM	NENHUM
LIK 3988		NENHUM	NENHUM

O caractere “L” que esteve em 34 amostras foi reconhecido em 10 amostras, ou seja, uma taxa de 29,41% de acerto. Esse caractere foi treinado com 4 amostras, sendo que no segundo treinamento, o mesmo já foi reconhecido.

O caractere “6” que esteve em 23 amostras foi reconhecido em 4 amostras, ou seja, uma taxa de acerto de 17,39%, sendo treinado com 5 amostras.

O caractere “8” que esteve em 19 amostras foi reconhecido parcialmente em 2 amostras, ou seja, uma taxa de 10,52%, sendo treinado em 5 amostras. Uma amostra reconhecida parcialmente significa que mais de uma rede neural reconheceu esse caractere.

O caractere “1” que esteve em 13 amostras e foi reconhecido parcialmente em 2 amostras e reonhecida em uma amostra, ou seja, uma taxa de 23,07% de acerto, ele foi treinado com 4 amostras.

É importante dizer que esses testes foram feitos em outras amostras, sem serem usadas as amostras que foram treinadas.

Após o treinamento em 12% das amostras, as redes neurais foram zeradas, ou seja, todas as redes neurais foram geradas com pesos aleatórios novamente, e então, foi feito o teste com 20% das amostras.

Tabela 13 - Tabela de 20% de treinamento

Placa	20% TREINADO		
	TREINADO	RECONHECIDOS	RECONHECIDOS PARCIAL
LOH 8516		NENHUM	NENHUM
LCD 7645	L,C,D,7,6,4,5		
LNZ 4292		NENHUM	NENHUM
JLW 9464		NENHUM	NENHUM
KRD 7366		NENHUM	NENHUM
LNJ 8753		NENHUM	NENHUM
LNP 1417		NENHUM	NENHUM
LID 5816		NENHUM	NENHUM
DUR 7778		NENHUM	NENHUM
LNF 5873		L	NENHUM
LNR 3839		NENHUM	NENHUM
LIN 2148		NENHUM	4
KMV 8068		NENHUM	NENHUM
LNZ 7675		NENHUM	NENHUM
LNO 3968		L	NENHUM
LNP 7221	L,N,P,7,2,2,1		
LND 3416		NENHUM	NENHUM
LNZ 3777		N	NENHUM
LNG 6631		NENHUM	NENHUM
KNI 0508	K,N,I,0,5,0,8		
KMU 0087		NENHUM	NENHUM
LOE 2022		NENHUM	NENHUM
LCS 8217		NENHUM	NENHUM
LBD 2319		9	NENHUM
CSE 9780	C,S,E,9,7,8		
LBU 5699		NENHUM	NENHUM
LAY 2596		9	NENHUM
LBL 9606		NENHUM	6
LKA 8614	L,A,K,8,6,1,4		
LNI 2993		L	NENHUM
LOE 0836		NENHUM	NENHUM
AAY 5127		NENHUM	NENHUM
LOA 2563		NENHUM	NENHUM
KND 8865	K,N,D,8,8,6,5		
LNR 0491		9	NENHUM
KNP 9837		NENHUM	NENHUM
LAV 5233		NENHUM	NENHUM
KMG 6921		NENHUM	9
LBO 1230	L,B,O,1,2,3,0		
KOD 7347		NENHUM	NENHUM
LNU 2885		NENHUM	NENHUM
LCN 7724	L,C,N,7,7,2,4		2
ABY 9198		NENHUM	NENHUM
GZA 2477	G,Z,A,2,4,7,7		
LAR 7207		NENHUM	NENHUM
LKI 6549	L,K,I,6,5,4,9		
CTB 1569		NENHUM	C
LOF 3405		NENHUM	NENHUM
LIK 3988		9	NENHUM

O caractere “L” que esteve em 30 amostras e foi reconhecido em 3 amostras, obteve uma taxa de acerto de 10,00%, sendo que foi treinado com 4 amostras. No segundo treinamento, o mesmo já foi reconhecido.

O caractere “N” que esteve em 15 amostras e foi reconhecido em 1 amostra, obteve uma taxa de 6,66%, este foi treinado com 4 amostras.

O caractere “9” que esteve em 18 amostras e foi reconhecido em 5 amostras obteve uma taxa de 26,31% de acerto, ele foi treinado com 2 amostras.

O caractere “6” que esteve em 19 amostras e foi reconhecidos parcialmente em 1 amostra, obteve uma taxa de 5,263% e foi treinado em 2 amostras.

O caractere “2” que esteve em 16 amostras e foi reconhecidos parcialmente em 1 amostra, obteve uma taxa de 6,25% e foi treinado em 2 amostras.

O caractere “C” que esteve em 2 amostras e foi reconhecidos parcialmente em 1 amostra, obteve uma taxa de 6,25% e foi treinado em 2 amostras.

Foi verificado que mesmo a rede neural sendo treinada para o reconhecimento de um caractere, ou até que exeda o limite de treinamentos, o que acontece em nosso sitema é que uma rede neural é treinada para realizar o reconhecimento de um determinado caractere, e o não reconhecimento de outros caracteres, por exemplo: o sistema possui 36 redes neurais, sendo que cada rede é responsável pelo reconhencimento de um determinado caracter. Para o treinamento de um determinado caractere é mostrado o caractere para a rede e dizendo que aquele caractere é o caractere que ele deve reconhecer, e a mesma amostra é mostrada as outras redes neurais, só que mostradas como não sendo o caracter que tem que ser reconhecido, então quando uma amostra com o caracter “A” é entrada na rede para ser treinada, a rede neural responsável pelo reconhecimento do caractere “A” é treinada até que se obtenha uma resposta verdadeira, querendo dizer que a rede neural em questão está com os pesos certos, enquanto isso as outras redes neurais também são treinadas, só que para não reconhecer o caracter em questão como é o caso da rede neural responsável pelo reconhecimento do caractere “B”, enquanto a rede neural “A” é treinada para o reconhecimento do caractere “A”, a rede neural “B” é treinada para o não reconhecimento do caractere “A”, o problema é que quando é entrada uma figura com o caractere “B”, a rede neural “A” é reajustada, de forma que ele não reconheça o caractere “B”, porém quando a amostra “A” é mostrada ao sistema, como a rede neural “A” pode acabar dizendo que não é um caractere “A”, então o algoritmo para o treinamento foi alterado, de que forma que

realize o treinamento até que todas as redes neurais sejam treinadas para a amostra em questão, como também para as outras amostras. O abaixo é mostrado o algoritmo.

```
while (true) {

                //Itera todas as amostras
                for (final AmostraListaItem item : BtnCaracterReconhecido.amostraListaItems) {
                   treinado = false;
                   //Realiza o treinamento até o reconhecimento no máximo 100 iterações
                   while ((!treinado) && (qtdTreinamentos++ < 100)) {
                      treinado = true;
                      //Itera todas as redes neurais
                      for (final RedeNeural neural :
BtnCaracterReconhecido.this.gerenciadorRedeNeural.getRedes()) {
                         if (neural.getCaracterReconhecido() == item.caracter) {
                            treinado =
((neural.getPerceptron().treinarAmostra(item.amostraVerdadeira)) && treinado);
                         } else {
                            treinado = ((neural.getPerceptron().treinarAmostra(item.amostraFalsa))
&& treinado);
                         }
                      }
                   }
                }

                if (treinado) {
                   break;
                }

                    }
```

Com o algortimo novo, foram realizados novos testes, abaixo segue tabela de testes realizados em 12% das amostras.

Tabela 14 - Tabela de 12% de treinamento com amostras antigas

Placa	10% TREINADO TODAS		
	TREINADO	RECONHECIDOS	RECONHECIDOS PARCIAL
LOH 8516	L,O,H,8,5,1,6		
LCD 7645		NENHUM	6
LNZ 4292		NENHUM	9
JLW 9464		NENHUM	NENHUM
KRD 7366		NENHUM	NENHUM
LNJ 8753		NENHUM	NENHUM
LNP 1417	L,N,P,1,4,1,7	L	
LID 5816		L	NENHUM
DUR 7778		8	NENHUM
LNF 5873		L	8
LNR 3839		L,8	NENHUM
LIN 2148		L,1	8
KMV 8068	K,M,V,8,0,6,8		
LNZ 7675		NENHUM	NENHUM
LNO 3968			8
LNP 7221		L,1	NENHUM
LND 3416		NENHUM	NENHUM
LNZ 3777		NENHUM	L,7
LNG 6631	L,N,G,6,6,3,1		
KNI 0508		8	NENHUM
KMU 0087		NENHUM	NENHUM
LOE 2022		NENHUM	NENHUM
LCS 8217		NENHUM	NENHUM
LBD 2319		L	NENHUM
CSE 9780	C,S,E,9,7,8,0	8	
LBU 5699		L	NENHUM
LAY 2596		NENHUM	6
LBL 9606		L,L	NENHUM
LKA 8614		L,8	NENHUM
LNI 2993		NENHUM	NENHUM
LOE 0836	L,O,E,0,8,3,6	L	
AAY 5127		NENHUM	NENHUM
LOA 2563		NENHUM	NENHUM
KND 8865		NENHUM	NENHUM
LNR 0491		1	NENHUM
KNP 9837		NENHUM	NENHUM
LAV 5233		NENHUM	NENHUM
KMG 6921		NENHUM	NENHUM
LBO 1230		NENHUM	L,1
KOD 7347		NENHUM	NENHUM
LNU 2885		L,8	NENHUM
LCN 7724		NENHUM	NENHUM
ABY 9198		NENHUM	NENHUM
GZA 2477		NENHUM	NENHUM
LAR 7207		NENHUM	L
LKI 6549		L	NENHUM
CTB 1569		NENHUM	NENHUM
LOF 3405		NENHUM	NENHUM
LIK 3988		NENHUM	3

O caractere “L” que esteve em 32 amostras e foi reconhecido em 15 amostras, obteve uma taxa de 46,87% e foi treinado em 4 amostras.

O caractere “8” que esteve em 19 amostras e foi reconhecido em 8 amostras, obteve uma taxa de 42,10% e foi treinado em 5 amostras.

Como pode ser visto, a rede neural “L” foi de 29,41% de taxa de acerto para 46,87% de taxa de acerto com esse novo algortimo, abaixo segue tabela com 20% de treinamento:

Tabela 15 - Tabela de 20% de treinamento com amostras antigas

Placa	20% TREINADO TODAS		
	TREINADO	RECONHECIDOS	RECONHECIDOS PARCIAL
LOH 8516	L,O,H,8,5,1,6		
LCD 7645		6	NENHUM
LNZ 4292		NENHUM	NENHUM
JLW 9464		NENHUM	NENHUM
KRD 7366		NENHUM	NENHUM
LNJ 8753		NENHUM	NENHUM
LNP 1417	L,N,P,1,4,1,7	L	NENHUM
LID 5816		8	5
DUR 7778		7,8	NENHUM
LNF 5873	L,N,F,5,8,7,3	L,8,7	NENHUM
LNR 3839		L,8	NENHUM
LIN 2148		L,8	NENHUM
KMV 8068	K,M,V,8,0,6,8		
LNZ 7675		6,5	NENHUM
LNO 3968			9
LNP 7221		L,1	NENHUM
LND 3416		NENHUM	NENHUM
LNZ 3777		L	NENHUM
LNG 6631	L,N,G,6,6,3,1		
KNI 0508		8	NENHUM
KMU 0087		NENHUM	NENHUM
LOE 2022		NENHUM	NENHUM
LCS 8217		1	NENHUM
LBD 2319		NENHUM	NENHUM
CSE 9780	C,S,E,9,7,8,0	8	
LBU 5699		L	5,9
LAY 2596		NENHUM	6
LBL 9606	L,B,L,9,6,0,6	L,L,6,6	NENHUM
LKA 8614		L,8	NENHUM
LNI 2993		NENHUM	NENHUM
LOE 0836	L,O,E,0,8,3,6	L	
AAY 5127		NENHUM	NENHUM
LOA 2563		NENHUM	NENHUM
KND 8865	K,N,D,8,8,6,5	NENHUM	NENHUM
LNR 0491		1	NENHUM
KNP 9837		NENHUM	NENHUM
LAV 5233		NENHUM	NENHUM
KMG 6921		NENHUM	NENHUM
LBO 1230		NENHUM	L,1
KOD 7347		NENHUM	NENHUM
LNU 2885		L,8	NENHUM
LCN 7724		NENHUM	NENHUM
ABY 9198		NENHUM	NENHUM
GZA 2477		NENHUM	NENHUM
LAR 7207		NENHUM	L
LKI 6549	L,K,I,6,5,4,9	L	NENHUM
CTB 1569		NENHUM	NENHUM
LOF 3405		NENHUM	NENHUM
LIK 3988		L	NENHUM

O caractere “L” que esteve em 28 amostras e foi reconhecido em 10 amostras, obteve uma taxa de 35,71% e foi treinado em 8 amostras.

Isso indica que quantidade de amostras a serem treinadas foi almentada, porém o resultado não foi o esperado, pois não depende somente da rede neural, e também das amostras utilizadas nas mesmas, como foi visto nesse caso, as amostras utilizadas nesse processo são de pior qualidade do que as utilizadas no processo de treinamento com 12%.

5.4. Conclusão

Desenvolver um Sistema de reconhecimento automático de placas de veículos não é nada trivial, envolve muito conhecimento em várias técnicas diferentes. Como foi pesquisado, o simples fato de segmentar a placa de um veículo é algo muito dificultoso, e também é um passo de muita importância para no sistema.

A metodologia proposta nesse estudo foi criar um sistema que possa localizar e identificar placas de veículos automaticamente, o que se pode dizer que foi alcançado em partes, pois ainda é preciso aprimorar a segmentação dos caracteres, os resultados também poderiam ter sido melhores no reconhecimento dos caracteres, pois a pequena base de 100 imagens não foi o suficiente para um treinamento adequado das redes neurais.

As principais dificuldades foram encontrar métodos que resolvessem o problema de forma simples e que funcionasse para placas vermelhas e brancas e que diferenciassem símbolos e fárois, o que ocorreu na maioria dos métodos. Os resultados como um todo foi satisfatório, mas pode ainda ser aprimorado. Outra grande dificuldade foi a segmentação dos caracteres, muito mais trabalhoso que a identificação da placa.

Como trabalhos futuros poderiam ser aprimorados os algoritmos e técnicas, principalmente a segmentação dos caracteres, e encontrar uma maneira de fazer com que a máscara que busca a placa na imagem seja variável automaticamente de acordo com o tamanho da placa e depois portar o sistema a plataforma móvel android que utiliza a linguagem Java.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBUQUERQUE, M. P. de**. Projeto de Reconhecimento de Placas de Veículos Brasileiros**. CENTRO BRASILEIRO DE PESQUISAS FÍSICAS, 2006. Disponível em: <http://www.cbpf.br/cat/pdsi/lpr/lpr.html> Acesso em: 22/05/2011.

BAILER,Werner. **Writing ImageJ Plugins – A Tutorial**. Upper Austria University of Applied Sciences, Austria, 2006, Disponível em : <http://www.imagingbook.com/fileadmin/goodies/ijtutorial/tutorial171.pdf> Acesso em: 31/10/2011.

CARVALHO, Jonh. Dissertação de Mestrado. **Uma Abordagem de Segmentação de Placas de Automóveis Baseada em Morfologia Matemática.** UFF, RJ, 2006. Disponível em: < http://www.ic.uff.br/PosGraduacao/Dissertacoes/296.pdf> Acesso em: 22/05/2011.

CONCI, Aura; MONTEIRO, Leonardo H. **RECONHECIMENTO DE PLACAS DE VEÍCULOS POR IMAGEM.** UFF, RJ, 2004. Disponível em: <http://www.ic.uff.br/~aconci/CONENPLACAS.pdf> Acesso em: 22/05/2011.

CONSELHO NACIONAL DE TRÂNSITO. **RESOLUÇÃO Nº 241, DE 22 DE JUNHO DE 2007** – CONTRAN Disponível em: < http://www.denatran.gov.br/download/Resolucoes/RESOLUCAO_231.pdf > Acesso em: 22/05/2011.

GONZALEZ, R. C.; WOODS, R. E. **Processamento Digital de Imagens.** 3. ed **.** São Paulo: Pearson, 2010.

GONZALEZ, R. C.; WOODS, R. E. **Processamento de Imagens Digitais.** SP: Edgard Blucher, 2000.

GUINDO Bruno G; THOMÉ, Antonio C.; RODRIGUES, Roberto J. **RECONHECIMENTO AUTOMÁTICO DE PLACAS DE VEÍCULOS AUTOMOTORES ATRAVÉS DE REDES NEURAIS ARTIFICIAIS.** UFRJ, RJ, 2002. Disponível em: < http://www.labic.nce.ufrj.br/downloads/2cbcomp_2002.pdf > Acesso em: 22/05/2011.

HAYKIN, Simon. **Neural Networks - A comprehensive Foundation.** 2. ed. Hamilton, Ontario, Canada: Pearson, 1999.

LANDINI, Gabriel. **Auto Threshold and Auto Local Threshold.** University of Birmingham, Inglaterra, 2011. Disponível em: < http://www.dentistry.bham.ac.uk/landinig/software/autothreshold/autothreshold.html> Acesso em: 30/10/2011.

MARQUES FILHO, O; VIEIRA NETO, H. **Processamento Digital de Imagens.** RJ: Brasport, 1999.

MCCARTHY, John**. WHAT IS ARTIFICIAL INTELLIGENCE?.** STANFORD, California, Estados Unidos, 2007. Disponível em: <http://www-formal.stanford.edu/jmc/whatisai/node1.html> Acesso em: 30/10/2011.

MINISTÉRIO DAS CIDADES. DEPARTAMENTO NACIONAL DE TRÂNSITO – DENATRAN. COORDENADORIA GERAL DE INFORMATIZAÇÂO E ESTATÍSTICA – CGIE. **RENAVAM REGISTRO NACIONAL DE VEÍCULOS AUTOMOTORES. MANUAL DE PROCEDIMENTOS.** 1990.
Disponível em: <http://www.detran.ce.gov.br/consultas/arquivos/manrenavam.pdf > Acesso em: 22/05/2011.

MIRANDA, Jose I. **Processamento de Imagens Digitais Prática Usando Java.** SP: Embrapa, 2006.

MUNIZ, Bruno D. Trabalho de Conclusão de Curso. **Utilização de Redes Neurais para Identificação de Caracteres de Placas de Veículos Automotores.** Instituto de Ensino Superior COC, SP, 2007.

PEREZ, Luís. **Qual a lógica das letras nas placas dos carros?** Super Interessante, n.200, Maio, 2004. Disponível em: <http://super.abril.com.br/superarquivo/2004/conteudo_124527.shtml> Acesso em: 22/05/2011.

PRATT, W. K.**Digital Image Processing**, Wiley Interscience, 1991. (2nd ed.).

RUSSEL, Stuart; NORVIG, Peter. **Artificial Intelligence – A Modern Approch**, Prentice – Hall, New Jersey, 1995.

SOUZA, Caio, *et al*. **RECONHECIMENTO DE PLACAS DE AUTOMÓVEIS ATRAVÉS DE CÂMERAS IP.** Universidade IMES, SP, 2006. Disponível em: <http://www.aedb.br/seget/artigos06/916_Copia%20de%20Placas.pdf> Acesso em: 22/05/2011.

SOFTVAIRES. Reconhecimento de Placas. **Aplicações.**
Disponível em: < http://www.softvaires.com.br/pgs_conteudo/lpr_aplicacoes.htm > Acesso em: 22/05/2011.

TECNIMA. Produtos. **Outras Aplicações.**
Disponível em: < http://www.tecnima.com.br/site/produtos.asp?item=6 > Acesso em: 22/05/2011.

TONSIG, Sérgio L. **Redes Neurais Artificiais Multicamadas e o Algoritmo Backpropagation,** 2000 Disponivel em: <http://funk.on.br/esantos/doutorado/INTELIG%CANCIA%20ARTIFICIAL/T%C9CNICAS/REDES%20NEURAIS/Redes%20Neurais%20Artificiais%20Multicamadas/Backp.PDF> Acesso em: 30/10/2011.

UNIVERSIDADE ESTADUAL DE CAMPINAS.
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO.
DEPARTAMENTO DE ENGENHARIA DA COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL. **Sistemas de Reconhecimento da Placa de Licenciamento Veicular.** 2006. Disponível em: < http://www.dca.fee.unicamp.br/~gaiotto/projects/srplv.html> Acesso em: 22/05/2011.

APÊNDICE A

Abaixo são mostrados os algoritmos utilizados para a realização do Filtro Sobel.

Algoritmo para filtragem Sobel vertical e horizontal é mostrado abaixo:

```
public BufferedImage PassaAltaInteiro() {
      // Matrizes de convolução Sobel.
      final int[][] SobelHorizontal = { { 1, 2, 1 }, { 0, 0, 0 }, { -1, -2, -1
} };
      final int[][] SobelVertical = { { -1, 0, 1 }, { -2, 0, 2 }, { -1, 0, 1 }
};
      long pixelAtual;
      /* Imagens para a convolução sendo "im" a imagem obtida a partir da
      imagem original da classe , "imNova" a imagem que recebe o filtro
      horizontal e imNova2 que recebe o filtro vertical.
       */
      final BufferedImage im = new BufferedImage(this.getImage().getWidth(),
      this.getImage().getHeight(),BufferedImage.TYPE_BYTE_GRAY);
      final BufferedImage imNova = new
      BufferedImage(this.getImage().getWidth(), this.getImage().getHeight(),
      BufferedImage.TYPE_BYTE_GRAY);
      final BufferedImage imNova2 = new
      BufferedImage(this.getImage().getWidth(),
      this.getImage().getHeight(),BufferedImage.TYPE_BYTE_GRAY);
      // im recebe a imagem original
      final Graphics g = im.getGraphics();
      g.drawImage(this.getImage(), 0, 0, null);
      g.dispose();
      /*
       São criados os objetos "Raster" a partir das imagens criadas sendo o
       objeto "Raster" um objeto que só permite leitura e "WritableRaster" um
       objeto que permite leitura e escrita.
      */
      final Raster wRaster = im.getRaster();
      final WritableRaster raster = imNova.getRaster();
      final WritableRaster raster2 = imNova2.getRaster();

      // Primeira convolução com a máscara Sobel horizontal.
      for (int i = 1; i < im.getWidth() - 1; i++) {
            for (int j = 1; j < im.getHeight() - 1; j++) {
                  pixelAtual = wRaster.getSample(i - 1, j - 1, 0) *
                  SobelHorizontal[0][0] + wRaster.getSample(i, j - 1, 0)*
                  SobelHorizontal[0][1] + wRaster.getSample(i + 1, j - 1, 0) *
                  SobelHorizontal[0][2]+ wRaster.getSample(i - 1, j, 0) *
                  SobelHorizontal[1][0] + wRaster.getSample(i, j, 0)*
                  SobelHorizontal[1][1] +
                  wRaster.getSample(i + 1, j, 0) * SobelHorizontal[1][2]+
                  wRaster.getSample(i - 1, j + 1, 0) * SobelHorizontal[2][0] +
                  wRaster.getSample(i, j + 1, 0)* SobelHorizontal[2][1] +
                  wRaster.getSample(i + 1, j + 1, 0) * SobelHorizontal[2][2];
```

```
                    // Condição para inpedir que o pixel seja maior que 255 ou
                    menor que 0
                    if (pixelAtual > 255) {
                          pixelAtual = 255;
                    } else
                          if (pixelAtual < 0) {
                                pixelAtual = 0;
                          }
                    raster.setSample(i, j, 0, pixelAtual);
              }
        }
        // Segunda convolução com a máscara Sobel vertical.
        for (int i = 1; i < im.getWidth() - 1; i++) {
              for (int j = 1; j < im.getHeight() - 1; j++) {
                    pixelAtual = wRaster.getSample(i - 1, j - 1, 0) *
                    SobelVertical[0][0] + wRaster.getSample(i, j - 1, 0)*
                    SobelVertical[0][1] +
                    wRaster.getSample(i + 1, j - 1, 0) * SobelVertical[0][2]+
                    wRaster.getSample(i - 1, j, 0) * SobelVertical[1][0] +
                    wRaster.getSample(i, j, 0)* SobelVertical[1][1] +
                    wRaster.getSample(i + 1, j, 0) * SobelVertical[1][2]+
                    wRaster.getSample(i - 1, j + 1, 0) * SobelVertical[2][0] +
                    wRaster.getSample(i, j + 1, 0)* SobelVertical[2][1] +
                     wRaster.getSample(i + 1, j + 1, 0) * SobelVertical[2][2];
                    // Condição para impedir que o pixel seja maior que 255 ou
                    menor que 0.
                    if (pixelAtual > 255) {
                          pixelAtual = 255;
                    } else
                          if (pixelAtual < 0) {
                                pixelAtual = 0;
                          }
                    raster2.setSample(i, j, 0, pixelAtual);
              }
        }
        // Junção dos resultados obtidos nas duas imagens.
        for (int i = 1; i < im.getWidth() - 1; i++) {
              for (int j = 1; j < im.getHeight() - 1; j++) {
                    pixelAtual = raster.getSample(i, j, 0) +
raster2.getSample(i, j, 0);
                    if (pixelAtual > 255) {
                          pixelAtual = 255;
                    } else
                          if (pixelAtual < 0) {
                                pixelAtual = 0;
                          }
                    raster2.setSample(i, j, 0, pixelAtual);
                    }
        }
        return imNova2;

}
```

Algoritmo para filtragem Sobel vertical :

```
public BufferedImage PassaAltaVertical() {
      // Matriz de convolução Sobel.
      final int[][] SobelVertical = { { -1, 0, 1 }, { -2, 0, 2 }, { -1, 0, 1 }
};
      long pixelAtual;
      /* Imagens para a convolução sendo "im" a imagem obtida a partir da
      imagem original da classe e imNova2 que recebe o filtro vertical.
       */

      final BufferedImage im = new BufferedImage(this.getImage().getWidth(),
      this.getImage().getHeight(),
      BufferedImage.TYPE_BYTE_GRAY);
      final BufferedImage imNova2 = new
      BufferedImage(this.getImage().getWidth(), this.getImage().getHeight(),
      BufferedImage.TYPE_BYTE_GRAY);

      // im recebe a imagem original.
      final Graphics g = im.getGraphics();
      g.drawImage(this.getImage(), 0, 0, null);
      g.dispose();
      /*
       São criados os objetos "Raster" a partir das imagens criadas sendo o
       objeto "Raster" um objeto que só permite leitura e "WritableRaster" um
       objeto que permite leitura e escrita.
      */

      final Raster wRaster = im.getRaster();
      final WritableRaster raster = imNova2.getRaster();
      // Convolução com a máscara Sobel vertical.
      for (int i = 1; i < im.getWidth() - 1; i++) {
            for (int j = 1; j < im.getHeight() - 1; j++) {
                  pixelAtual = wRaster.getSample(i - 1, j - 1, 0) *
                  SobelVertical[0][0] + wRaster.getSample(i, j - 1, 0)*
                  SobelVertical[0][1] +
                  wRaster.getSample(i + 1, j - 1, 0) * SobelVertical[0][2]+
                  wRaster.getSample(i - 1, j, 0) * SobelVertical[1][0] +
                  wRaster.getSample(i, j, 0)* SobelVertical[1][1] +
                  wRaster.getSample(i + 1, j, 0) * SobelVertical[1][2]+
                  wRaster.getSample(i - 1, j + 1, 0) * SobelVertical[2][0] +
                  wRaster.getSample(i, j + 1, 0)* SobelVertical[2][1] +
                  wRaster.getSample(i + 1, j + 1, 0) * SobelVertical[2][2];
                  // Condição para impedir que o pixel seja maior que 255 ou
                  menor que 0.
                  if (pixelAtual > 255) {
                        pixelAtual = 255;
                  } else
                        if (pixelAtual < 0) {
                              pixelAtual = 0;
                  }
                  raster.setSample(i, j, 0, pixelAtual);
            }
      }
      return imNova2;
}
```

APÊNDICE B

Esse apêndice mostra os algoritmos para binarização da imagem.

Utilizando as bibliotecas do Java:

```
public BufferedImage Binariza() {

            // Esse método apenas cria uma imagem do tipo "BINARY" e insere a
            imagem original dentro dela.
            final BufferedImage im = new
            BufferedImage(this.getImage().getWidth(),
            this.getImage().getHeight(),BufferedImage.TYPE_BYTE_BINARY);
            final Graphics g = im.getGraphics();
            g.drawImage(this.getImage(), 0, 0, null);
            g.dispose();
            return im;

}
```

Utilizando a API do ImageJ:

```
public BufferedImage autoThreshold() {
            // Na binarização utilizando o ImageJ método o autoThreshold()
            realiza o trabalho.
            final BufferedImage imThreshold = new BufferedImage(im.getWidth(),
            im.getHeight(), BufferedImage. TYPE_BYTE_GRAY);
            final Graphics g = imThreshold.getGraphics();
            g.drawImage(this.image, 0, 0, null);
            g.dispose();
            final ByteProcessor byteProc = new ByteProcessor(imThreshold);
            byteProc.autoThreshold();
            this.processed = byteProc.getBufferedImage();
            return byteProc.getBufferedImage();
}
```

Utilizando um algoritmo com limiar fixo:

```
public BufferedImage Threshold(final BufferedImage imagem) {
            //Nesse método todos os pixels que tiverem o valor maior que ou
            igual a 125 passam a ser brancos e os que forem menores passam a
            ser pretos.
            final BufferedImage im = new BufferedImage(imagem.getWidth(),
            imagem.getHeight(), BufferedImage.TYPE_BYTE_GRAY);
            for (int i = 0; i < imagem.getWidth(); i++) {
                  for (int j = 0; j < imagem.getHeight(); j++) {
                        if (imagem.getRaster().getSample(i, j, 0) >= 125) {
                              im.getRaster().setSample(i, j, 0, 255);
                        } else {
```

```
                        im.getRaster().setSample(i, j, 0, 0);
                    }
                }
            }
            return im;
        }
```

APÊNDICE C

Esse apêndice mostra o algoritmo para esqueletização da imagem.

```
public BufferedImage Esqueleto() {
	// É criada uma imagem com a filtragem Sobel vertical.
	BufferedImage im = this.PassaAltaVertical();
	// É criado o tipo ByteProcessor da API ImageJ que é responsável
	pelo tratamento da imagem.
	final ByteProcessor processador = new ByteProcessor(im);
	// É realizada a binarização da imagem.
	processador.autoThreshold();
	// A operação de esqueletização só reconhece linhas pretas no
	fundo branco então é realizada uma inversão de cores ou negativo.
	processador.invertLut();
	// Operação de esqueletização.
	processador.skeletonize();
	// Após a esqueletização é realizada outra inversão de cores para
	que as cores voltem a ser linhas brancas no fundo preto.
	processador.invertLut();
	// Após a esqueletização o objeto processador cria um objeto
	BufferedImage.
	return processador.getBufferedImage());

}
```

APÊNDICE D

Nesse apêndice serão mostrados os algoritmos e classes utilizados para realizar a equalização.

```
/*
	Classe responsável pelo armazenamento do Histograma da imagem.
*/
public class Histograma {
	/*
	Vetor que representa o histograma sendo os seus índices são os tons de
	cinza e os valores armazenados nos índices a quantidade de cada tom de
	cinza.
	*/
	private int[] base;
	//imagem do histograma.
	private BufferedImage bufferedImage;

	//Contrutores da classe Histograma.
	private Histograma() {
		super();
	}

	public Histograma(final BufferedImage bufferedImage) {
		this.bufferedImage = bufferedImage;
		this.iniciarHistograma();
	}

	public int[] getBase() {
		return this.base;
	}

	// Método que retorna o histograma equalizado
	public Histograma getEqualizado() {
		final Histograma resultado = new Histograma();
		resultado.base = new int[this.base.length];
		resultado.base[0] = this.base[0];
		int somatoria = 0;
		for (int i = 0; i < resultado.base.length; i++) {
			somatoria = somatoria + this.base[i];
			resultado.base[i] = somatoria;
		}
		final int I = this.bufferedImage.getWidth() *
		this.bufferedImage.getHeight() / 255;
		for (int i = 0; i < resultado.base.length; i++) {
			int novoValor = resultado.base[i] / I;
			novoValor = Math.round(novoValor) - 1;
			resultado.base[i] = (novoValor > 0) ? novoValor : 0;
		}
		return resultado;
	}

	//Método que inicializa o Histograma.
	private void iniciarHistograma() {
		this.base = new int[256];
		for (int x = 0; x < this.bufferedImage.getWidth(); x++) {
```

```
                for (int y = 0; y < this.bufferedImage.getHeight(); y++) {
                      this.base[this.bufferedImage.getRaster().getSample(x,
y, 0)]++;
                }
          }
      }

}
```

```
/*
Esse Método é chamado pela classe de filtragem que irá criar o
histograma, chamar o método para equalizá-lo e depois a partir do
histograma equalizado criar a imagem equalizada.
*/

public BufferedImage Equaliza() {
      final BufferedImage equalizado = new
      BufferedImage(this.image.getWidth(),
      this.image.getHeight(),BufferedImage.TYPE_BYTE_GRAY);
      final Histograma histograma = new Histograma(this.image);
      final Histograma histEqualizado = histograma.getEqualizado();
      for (int x = 0; x < this.image.getWidth(); x++) {
            for (int y = 0; y < this.image.getHeight(); y++) {
                  final int novoValor =
                  histEqualizado.getBase()[this.image.getRaster().getSam
                  ple(x, y, 0)];
                  equalizado.getRaster().setSample(x, y, 0, novoValor);
            }
      }
      return equalizado;
}
```

APÊNDICE E

Esse Apêndice tem como objetivo mostrar os métodos e classes utilizados para realizar a delimitação da região da placa, para isso também são utilizados os códigos dos Apêndices A, B e C anteriores, sendo do Apêndice B somente a binarização utilizando ImageJ.

A classe Ponto é utilizada por várias classes do sistema, ela representa um ponto 2d no espaço , possuindo os atributos X e Y que são as coordenadas do ponto no espaço.

```
/*
Classe Ponto que representa um Ponto 2d no espaço
*/

public class Ponto {
	private int x;
	private int y;

	public Ponto() {
		this.x = 0;
		this.y = 0;
	}

	public Ponto(final int x, final int y) {
		this.x = x;
		this.y = y;
	}

	public int getX() {
		return this.x;
	}

	public int getY() {
		return this.y;
	}

	public void setX(final int x) {
		this.x = x;
	}

	public void setY(final int y) {
		this.y = y;
	}

}
```

A classe abstrata AbstractScannerImagem é herdada por algumas classes do projeto é utilizada para percorrer a imagem com uma máscara. Essa classe também possibilita que a máscara percorra apenas uma parte da imagem. Imagine uma imagem e um retângulo ou uma

janela na imagem que delimita a região que será percorrida pela máscara, é esse o papel do objeto pontoCentral que instância a classe Ponto, que será o centro desse retângulo. A partir do centro do retângulo, a altura do retângulo é calculada variando para cima e para baixo o valor da máscara dividido pelo atributo divisorJanela. O mesmo é feito com a largura.

```
/*
Classe abstrata utilizada para percorrer a imagem com uma máscara.
*/

public abstract class AbstractScannerImagem {
	/*
	As variáveis alturaJanela e larguraJanela são as dimensões da máscara
	que irá percorrer a imagem.
	*/
	protected final int alturaMascara;
	private float divisorJanela;//variável que é utilizada para calcular a
altura.
	private final BufferedImage image; //imagem para a localização da placa.
		private Ponto pontoCentral = null; //Ponto central utilizado para
	delimitar a região percorrida pela máscara.

	/*
	Construtor simples para percorrer a imagem toda.
	*/

	public AbstractScannerImagem(final BufferedImage image, final int
largura, final int altura) {
		this.image = image;
		this.larguraMascara = largura;
		this.alturaMascara = altura;
	}


	/*
	Construtor que delimita a região que será percorrida pela máscara, onde
	não tem necessidade de passar o divisor, que assume o valor 1 como valor
	default.
	*/

	public AbstractScannerImagem(final BufferedImage image, final int
largura, final int altura, final Ponto ponto) {
		this.image = image;
		this.larguraMascara = largura;
		this.alturaMascara = altura;
		this.pontoCentral = ponto;
		this.divisorJanela = 1;
	}



	/*
```

```
	Construtor que delimita a região que será percorrida pela máscara, onde
	é passado o divisor
	*/

	public AbstractScannerImagem(final BufferedImage image, final int
largura, final int altura, final Ponto ponto,
			final float divisor) {
		this.image = image;
		this.larguraMascara = largura;
		this.alturaMascara = altura;
		this.pontoCentral = ponto;
		if (divisor == 0) {
			this.divisorJanela = 1;
		} else {
			this.divisorJanela = divisor;
		}

	}


	public int getAltura() {
		return this.alturaMáscara;
	}

	public BufferedImage getImage() {
		return this.image;
	}

	public int getLargura() {
		return this.larguraMáscara;
	}




	/*
	Método responsável por percorrer a imagem utilizando a máscara.
	*/

	public void scanear() {
		// Variáveis utilizadas para percorrer a imagem
		int larguraPercorreImagem;
		int alturaPercorridaImagem;
		int posXini;
		int posYini;
		int posXfim;
		int posYfim;

		//Ponto inicial da Janela
		final Ponto pontoInicial = new Ponto();



		// O ponto inicial da janela sua largura e altura são calculados.
		Se o ponto central não for passado então o espaço que será
```

```
            percorrido é a altura e a largura da imagem subtraída da altura e
            largura da máscara.
            if (this.pontoCentral != null) {
                  posXini = (int) (this.pontoCentral.getX() -
            (this.larguraMascara / this.divisorJanela));
                  posYini = (int) (this.pontoCentral.getY() -
            (this.alturaMascara / this.divisorJanela));
                  if (posXini < 0) {
                        posXini = 0;
                  }
                  if (posYini < 0) {
                        posYini = 0;
                  }
                  pontoInicial.setX(posXini);
                  pontoInicial.setY(posYini);
                  posXfim = (int) (this.pontoCentral.getX() +
                  (this.larguraMascara / this.divisorJanela));
                  posYfim = (int) (this.pontoCentral.getY() +
                  (this.alturaMascara / this.divisorJanela));
                  if (posXfim > this.image.getWidth()) {
                        posXfim = this.image.getWidth();
                  }
                  if (posYfim > this.image.getHeight()) {
                        posXfim = this.image.getHeight();
                  }
                  larguraPercorreImagem = posXfim;
                  alturaPercorridaImagem = posYfim;
            } else {
                  larguraPercorreImagem = this.image.getWidth() -
this.getLargura();
                  alturaPercorridaImagem = this.image.getHeight() -
this.getAltura();

            }
            for (int x = pontoInicial.getX(); x < larguraPercorreImagem; x++)
{
                  for (int y = pontoInicial.getY(); y <
alturaPercorridaImagem; y++) {
                        this.scanear(x, y);
                  }
            }
      }

      /*
      Esse método abstrato foi criado para que as classes que implementarem
      essa classe abstrata  possam fazer os cálculos da convolução de
      diferentes maneiras, como soma ou média por exemplo.
      */

      protected abstract void scanear(int x, int y);

}
```

A classe ScannerImagemMedia herda a classe AbstractScannerImagem implementando seu método scanear, nele é feita a média de todos os pixels dentro máscara conforme o método

for sendo chamado pela superclasse. A média é impressa na imagem de saída imageOut que é um objeto da classe BufferedImage. Após ser feita a média de toda a imagem o pixel com nível de intensidade mais alto da imagem imageOut é encontrado através do método PontoMaiorIntensidade, e sua posição é retorna através de um objeto da classe Ponto.

```
/*
Classe responsável por fazer a média da imagem e retornar o ponto de maior
intensidade.
*/

public class ScannerImagemMedia extends AbstractScannerImagem {
	// Objeto que irá guardar a imagem com a média.
	private final BufferedImage imageOut;

	/*
	Construtor da classe média que recebe como parâmetros imagem em que será
	achado o ponto com maior intensidade, o tamanho da máscara e a imagem de
	saída e repassa alguns para a superclasse.
	*/
	public ScannerImagemMedia(final BufferedImage image, final int largura,
	final int altura,
			final BufferedImage imageOut) {
		super(image, largura, altura);
		this.imageOut = imageOut;
		// TODO Auto-generated constructor stub
	}




	/*
	Método que percorre a máscara a partir de um ponto e retorna a média da
máscara.
	*/

	private int getMedia(final int x, final int y) {
		int media = 0;
		for (int xAux = x; xAux < this.getLargura() + x; xAux++) {
			for (int yAux = y; yAux < this.getAltura() + y; yAux++) {
				media += this.getImage().getRaster().getSample(xAux,
yAux, 0);
			}
		}
		media = media / (this.getLargura() * this.getAltura());
		return media;
	}

	/*
	Método que percorre a imagem de saída para achar o ponto de maior
intensidade.
	*/
```

```
	public Ponto PontoMaiorIntensidade() {
		int maior = 0;
		int xMaior = 0;
		int yMaior = 0;
		for (int x = 0; x < this.imageOut.getWidth(); x++) {
			for (int y = 0; y < this.imageOut.getHeight(); y++) {
				if (maior < this.imageOut.getRaster().getSample(x, y,
0)) {
					maior = this.imageOut.getRaster().getSample(x,
y, 0);
					xMaior = x;
					yMaior = y;
				}
			}
		}
		return new Ponto(xMaior, yMaior);

	}

	/*
	Método abstrato sobrescrito para pegar a média de um ponto utilizando a
	máscara da superclasse e depois imprimir esse valor na posição do centro
	da máscara em uma imagem de saída.
	*/
	@Override
	protected void scanear(final int x, final int y) {
		final int media = this.getMedia(x, y);
		this.imageOut.getRaster().setSample(x + (this.getLargura() / 2), y
+ (this.getAltura() / 2), 0, media);
	}
}
```

APÊNDICE F

Nesse Apêndice será mostrado o código feito para localizar a região da placa, para isso também são utilizados os códigos do Apêndice E.

A classe ScannerImagemMaiorSoma herda a classe AbstractScannerImagem e tem o objetivo de encontrar a região da placa na imagem. Para isso ela precisa receber o ponto de maior intensidade calculado pela classe ScannerImagemMedia em seu construtor e passá-lo para o construtor de sua superclasse, junto com o tamanho da máscara , a imagem em que será buscada a placa e o divisor .

Essa classe possui duas variáveis estáticas que indicam qual o tipo de máscara para ser feito o cálculo, uma mácara oca ou uma máscara inteira. Uma máscara oca terá bordas com pixels de valor 1 e interior de valor 0 . A máscara inteira terá todo o seu interior com valor 1. Ao instanciar uma classe ScannerImagemMaiorSoma deve ser passado o tipo de máscara no construtor que será salva no atributo tipo Janela. A máscara é representada pela matriz janela . Os atributos alturaRetangulo, larguraRetangulo, correcaoErroX e correcaoErroY são utilizados somente quando é desenhado um retângulo na imagem original dizendo onde placa se localiza, sendo que os atributos alturaRetangulo e larguraRetangulo são setados como o valor da máscara se não for passado nenhum parâmetro no contrutor. Já os atributos correcaoErroX e correcaoErroY redimensionam o retângulo que será desenhado na imagem para que o mesmo não perca possíveis partes da placa que ficaram fora da localização final da máscara. O atributo tamBorda é o tamanho da borda que será utilizado na máscara oca. O pontoSomaMax guarda o ponto inicial do retângulo da janela na imagem em que a placa foi localizada. O atributo somaMax guarda o valor da maior soma encontrada.

A classe funciona buscando o pixel com maior somatória através do método scanear da superclasse que percorre toda a imagem chamando o método scanear da subclasse que identifica qual o tipo de máscara é utilizada para fazer a somatória, e faz a somatória de cada ponto salvando sempre o valor e a localização do pixel com maior somatória. Ao terminar de percorrer a imagem o ponto com a maior somatória está salvo.

```
/*
Classe responsável por localizar a região da placa.
*/

public class ScannerImagemMaiorSoma extends AbstractScannerImagem {
```

```
	public final static int JANELA_INTEIRA = 1; // variável estática que
	representa uma máscara inteira.
	public final static int JANELA_OCA = 2; // variável estática que
	representa uma máscara oca.
	private final int alturaRetangulo; // altura do retângulo da região da
	placa que será desenhado.
	private final int correcaoErroX; // variável que redimensiona a
	coordenada X da região da placa localizada no momento em que ela for
	desenhada
	private final int correcaoErroY; // variável que redimensiona a
	coordenada Y da região da placa localizada no momento em que ela for
	desenhada.
	private int janela[][];//variável que representa a máscara
	private final int larguraRetangulo; // largura do retângulo da região da
	placa que será desenhado.

	private final Ponto pontoSomaMax = new Ponto();// ponto inicial do
retângulo onde está     a região da placa.
	private int somaMax = 0; //valor da maior somatória.
	private int tamBorda = 1; //tamanho da borda da máscara oca.
	private final int tipoJanela; //tipo de janela utilizada .

	/*
	Construtor simples.
	*/

	public ScannerImagemMaiorSoma(final BufferedImage image, final int
	largura, final int altura, final Ponto ponto,final float divisor){

		super(image, largura, altura, ponto, divisor);
		this.tipoJanela = ScannerImagemMaiorSoma.JANELA_INTEIRA;
		this.larguraRetangulo = this.larguraMascara;
		this.alturaRetangulo = this.alturaMascara;
		if (this.tipoJanela == ScannerImagemMaiorSoma.JANELA_OCA) {
			this.InicializaJanelaOca();
		}
		this.correcaoErroX = 0;
		this.correcaoErroY = 0;

	}

	/*
	Construtor que seta todos os atributos com exceção do tamanho da borda
da máscara oca.
	*/

	public ScannerImagemMaiorSoma(final BufferedImage image, final int
	largura, final int altura, final Ponto ponto,final float divisor, final
	int larguraRet, final int alturaRet, final int tipoJan, final int
	correcErroX,
			final int correcErroY) {
		super(image, largura, altura, ponto, divisor);
		this.tipoJanela = tipoJan;
		this.larguraRetangulo = larguraRet;
		this.alturaRetangulo = alturaRet;
		this.correcaoErroX = correcErroX;
		this.correcaoErroY = correcErroY;
```

```
            if (this.tipoJanela == ScannerImagemMaiorSoma.JANELA_OCA) {
                  this.InicializaJanelaOca();// se a máscara for do tipo
                  máscara oca , então a máscara é inicializada
            }

      }

      /*
      Método que realiza a somatória dos pixels dentro da máscara
      */

      public ScannerImagemMaiorSoma(final BufferedImage image, final int
      largura, final int altura, final Ponto ponto, final float divisor, final
      int larguraRet, final int alturaRet, final int tipoJan, final int
      correcErroX,
               final int correcErroY, final int borda) {
            super(image, largura, altura, ponto, divisor);
            this.tipoJanela = tipoJan;
            this.larguraRetangulo = larguraRet;
            this.alturaRetangulo = alturaRet;
            this.correcaoErroX = correcErroX;
            this.correcaoErroY = correcErroY;
            this.tamBorda = borda;
            if (this.tipoJanela == ScannerImagemMaiorSoma.JANELA_OCA) {
                  this.InicializaJanelaOca();// se a máscara for do tipo
                  máscara oca , então a máscara é inicializada.
            }
      }

      /*
      Método que desenha um retângulo na região encontrada da placa.
      */

      public BufferedImage DesenhaRetanguloMaiorSoma(final BufferedImage
image) {
            final BufferedImage imageOut = new
BufferedImage(this.getImage().getWidth(),
      this.getImage().getHeight(),BufferedImage.TYPE_INT_RGB);
            final Graphics g = imageOut.getGraphics();
            g.drawImage(image, 0, 0, null);
            final Graphics2D g2 = (Graphics2D) g;
            g2.setStroke(new BasicStroke(2.0f));
            g.setColor(Color.red);
            final int xIni;
            final int yIni;
            final int xFim;
            final int yFim;
            xIni = this.correcaoErroX + (this.pontoSomaMax.getX());
            yIni = this.correcaoErroY + (this.pontoSomaMax.getY());
            xFim = this.correcaoErroX + (this.pontoSomaMax.getX() +
this.larguraRetangulo);
            yFim = this.correcaoErroY + (this.pontoSomaMax.getY() +
this.alturaRetangulo);
            g.drawLine(xIni, yIni, xFim, yIni);
            g.drawLine(xIni, yFim, xFim, yFim);
            g.drawLine(xIni, yIni, xIni, yFim);
            g.drawLine(xFim, yIni, xFim, yFim);
```

```
			g.dispose();
			return imageOut;
		}

		/*
		Método que realiza a somatória dos valores da máscara a partir de um
		ponto na placa utilizando uma máscara com todos os valores 1.
		*/

		private void EncontraMaiorSoma(final int x, final int y) {
			int Soma = 0;
			for (int xAux = x; (xAux < this.getLargura() + x) && (xAux <
			this.getImage().getWidth()); xAux++) {
				for (int yAux = y; (yAux < this.getAltura() + y) && (yAux <
				this.getImage().getHeight()); yAux++) {
					Soma += this.getImage().getRaster().getSample(xAux,
yAux, 0);
				}
			}
			if (Soma > this.somaMax) {
				this.somaMax = Soma;
				this.pontoSomaMax.setX(x);
				this.pontoSomaMax.setY(y);
			}

		}




		/*
		Método que realiza a somatória dos valores da máscara a partir de um
		ponto na placa utilizando uma máscara oco com os valores das bordas 1 e
		no seu interior 0.
		*/


		private void EncontraMaiorSomaJanelaOca(final int x, final int y) {
			int Soma = 0;
			int xJanela = 0;
			int yJanela = 0;
			for (int xAux = x; (xAux < this.getLargura() + x) && (xAux <
					this.getImage().getWidth()); xAux++) {
				yJanela = 0;
				for (int yAux = y; (yAux < this.getAltura() + y) && (yAux <
				this.getImage().getHeight()); yAux++) {
					Soma += this.getImage().getRaster().getSample(xAux,
					yAux, 0) * this.janela[xJanela][yJanela];
					yJanela++;
				}
				xJanela++;
			}
			if (Soma > this.somaMax) {
				this.somaMax = Soma;
				this.pontoSomaMax.setX(x);
```

```
                this.pontoSomaMax.setY(y);
        }
}

public Ponto getPontoMaiorSoma() {
        return this.pontoSomaMax;
}

public Ponto getPontoSomaMax() {
        return this.pontoSomaMax;
}

public int getTamBorda() {
        return this.tamBorda;
}

/*
Método que inicializa a janela oca.
*/

public void InicializaJanelaOca() {
        this.janela = new int[this.getLargura()][this.getAltura()];
        for (int x = 0; x < this.getLargura(); x++) {
                for (int y = 0; y < this.getAltura(); y++) {
                        if ((y >= this.getAltura() - this.getTamBorda())
                        || (y < this.getAltura() - (this.getAltura() -
                        this.getTamBorda()))
                        || (x >= this.getLargura() - this.getTamBorda())
                        || (x < this.getLargura() - (this.getLargura() -
                        this.getTamBorda()))) {
                                this.janela[x][y] = 1;
                        } else {
                                this.janela[x][y] = 0;
                        }
                }
        }
}

/*
Método abstrato sobrescrito para pegar a maior somatória de um ponto
utilizando.
*/

@Override
protected void scanear(final int x, final int y) {
        if (this.tipoJanela == ScannerImagemMaiorSoma.JANELA_OCA) {
                this.EncontraMaiorSomaJanelaOca(x, y);
        } else {
                this.EncontraMaiorSoma(x, y);
        }

}

public void setTamBorda(final int tamBorda) {
        this.tamBorda = tamBorda;
}
```

}

A comunicação entre as classes ScannerImagemMedia e ScannerImagemMaiorSoma é mostrada no método LocalizaPlaca, que instancia a classe ScannerImagemMedia recupera o ponto dentro da placa e depois passa esse ponto para a classe ScannerImagemMaiorSoma localizar a placa.

```
/*
Método que localiza a placa utilizando as classes ScannerImagemMedia e
ScannerImagemMaiorSoma.
*/

public BufferedImage LocalizaPlaca() {
            // Tamanho da máscara.
            final int largura = 240;
            final int altura = 40;
            // Esqueletização do filtro Sobel com matriz vertical.
            final BufferedImage imEsqueleto = this.Esqueleto();
            // que armazena o resultado da média.
            final BufferedImage im = new
            BufferedImage(this.getImage().getWidth(),
            this.getImage().getHeight(),BufferedImage.TYPE_BYTE_GRAY);
            // Classe que realiza a média da imagem
            final ScannerImagemMedia imagemMedia = new
            ScannerImagemMedia(imEsqueleto, largura, altura, im);
            // Chamada do método que irá realizar a média da imagem.
            imagemMedia.scanear();
            // Após realizar a média, o ponto de maior intensidade da média de
            toda a imagem é recuperado.
            final Ponto p = imagemMedia.PontoMaiorIntensidade();
            //Sobel  vertical.
            final BufferedImage imPassa = this.PassaAltaVertical();
            //Ponto de maior intensidade é passado para a placa ser localizada
            final ScannerImagemMaiorSoma maiorSoma = new
            ScannerImagemMaiorSoma(imPassa, largura, altura, p, 2, 255, 55,
            ScannerImagemMaiorSoma.JANELA_INTEIRA, -5, -5);
            //Placa é localizada pelo método scanear.
            maiorSoma.scanear();
            //Retângulo com tamanho da placa é desenhado na imagem original
            return maiorSoma.DesenhaRetanguloMaiorSoma(this.getImage());

      }
```

APÊNDICE G

Esse apêndice mostra as classes e métodos utilizados na segmentação dos caracteres. A primeira classe que mostrada é a classe Retangulo, ela é utilizada na delimitação da região dos caracteres. Ela possui duas instancias da classe ponto, o pontoFinal e pontoInicial, que representam o ponto inicial e o ponto final de um retângulo.

```
/*
Classe Retangulo, utilizada na delimitação da região dos caracteres.
*/

public class Retangulo {
	/*
	atributos
	*/

	private Ponto pontoFinal;
	private Ponto pontoInicial;

	/*
	Construtores
	*/

	public Retangulo() {
		this.pontoInicial = new Ponto();
		this.pontoFinal = new Ponto();
	}
	public Retangulo(final Ponto pontoInicial, final Ponto pontoFinal) {
		this.pontoInicial = pontoInicial;
		this.pontoFinal = pontoFinal;
	}

	/*
	 Getters e Setters
	*/

	public Ponto getPontoFinal() {
		return this.pontoFinal;
	}

	public Ponto getPontoInicial() {
		return this.pontoInicial;
	}

	public void setPontoFinal(final Ponto pontoFinal) {
		this.pontoFinal = pontoFinal;
	}

	public void setPontoInicial(final Ponto pontoInicial) {
		this.pontoInicial = pontoInicial;
	}
```

```
}
```

A classe Pixel é utilizada para representação de um pixel em tons de cinza na imagem, contendo uma instância da classe ponto, e um atributo que representa o tom de cinza do pixel.

```
/*
Classe Pixel, representa um pixel em tons de cinza.
*/

public class Pixel {
	/*
	atributos
	*/

	private Ponto ponto;
	private int tomCinza;

	/*
	Construtores
	*
	public Pixel() {

	}

	public Pixel(final Ponto p, final int tom) {
		this.setPonto(p);
		this.setTomCinza(tom);
	}

	/*
	 Getters e Setters
	*/

	public Ponto getPonto() {
		return this.ponto;
	}

	public int getTomCinza() {
		return this.tomCinza;
	}

	public void setPonto(final Ponto ponto) {
		this.ponto = ponto;
	}

	public void setTomCinza(final int tonCinza) {
		this.tomCinza = tonCinza;
```

```
	}
}
```

A classe Placa recebe uma imagem, o ponto em que a placa do carro foi encontrada, a largura e altura da placa, e a partir da imagem original criam uma nova imagem contendo a placa veículo. Essa classe também possui atributos para deslocar o ponto em que a placa foi encontrada, que são as variáveis erroX e erroY. Esses atributos tem o objetivo de ajusta a região encontrada da placa, pois muitas placas encontradas ficavam fora do centro da região encontrada.

```
public class Placa {
	/*
	Atributos da classe placa
	*/
	private final int altura;
	private final int erroX;
	private final int erroY;
	private BufferedImage imagem;
	private final int largura;
	private final Ponto pontoInicial;

	/*
	Construtores
	*/

	public Placa(final BufferedImage image) {
		this.imagem = image;
		this.pontoInicial = new Ponto();
		this.altura = 0;
		this.largura = 0;
		this.erroX = 0;
		this.erroY = 0;
		this.imagem = image;
	}

	public Placa(final BufferedImage image, final Ponto p, final int
	largura, final int altura) {
		this.pontoInicial = p;
		this.altura = altura;
		this.largura = largura;
		this.erroX = 0;
		this.erroY = 0;
		int xIni;
		int yIni;
		int xFim;
		int yFim;
		xIni = this.erroX + (this.pontoInicial.getX());
		if (xIni < 0) {
			xIni = 0;
```

```
            }
            yIni = this.erroY + (this.pontoInicial.getY());
            if (yIni < 0) {
                  yIni = 0;
            }
            xFim = this.erroX + (this.pontoInicial.getX() + this.largura);
            if (xIni > this.imagem.getWidth()) {
                  xFim = this.imagem.getWidth();
            }
            yFim = this.erroY + (this.pontoInicial.getY() + this.altura);
            if (yFim > this.imagem.getHeight()) {
                  yFim = this.imagem.getHeight();
            }

            this.imagem = new BufferedImage(largura, altura, image.getType());
            final Graphics2D area = (Graphics2D)
this.imagem.getGraphics().create();
            area.drawImage(image, 0, 0, this.imagem.getHeight(),
            this.imagem.getWidth(), xIni, yIni, xFim, yFim, null);
            area.dispose();
      }

      public Placa(final BufferedImage image, final Ponto p, final int
      largura, final int altura, final int erroX,final int erroY) {
            this.imagem = image;
            this.pontoInicial = p;
            this.altura = altura;
            this.largura = largura;
            this.erroX = erroX;
            this.erroY = erroY;
            int xIni;
            int yIni;
            int xFim;
            int yFim;
            xIni = this.erroX + (this.pontoInicial.getX());
            if (xIni < 0) {
                  xIni = 0;
            }
            yIni = this.erroY + (this.pontoInicial.getY());
            if (yIni < 0) {
                  yIni = 0;
            }
            xFim = this.erroX + (this.pontoInicial.getX() + this.largura);
            if (xIni > this.imagem.getWidth()) {
                  xFim = this.imagem.getWidth();
            }
            yFim = this.erroY + (this.pontoInicial.getY() + this.altura);
            if (yFim > this.imagem.getHeight()) {
                  yFim = this.imagem.getHeight();
            }
            this.imagem = new BufferedImage(xFim - xIni, yFim - yIni,
image.getType());
            final Graphics2D area = (Graphics2D)
this.imagem.getGraphics().create();
            area.drawImage(image, 0, 0, xFim - xIni, yFim - yIni, xIni, yIni,
            xFim, yFim, null);
            area.dispose();
```

```
	}

	public BufferedImage getImagem() {
		return this.imagem;
	}

}
```

A classe ScannerImagemAssinatura é responsável pela segmentação dos caracteres. Ela não herda a classe AbstractScannerImagem como as outras classes que manipulam a imagem. As listas de dados da classe servem para diferentes funcionalidades. A lista listaTransicoes deve ser a primeira a ser inicializada, essa lista guarda os pixels de transição entre escalas de níveis de cinza, que são as transições entre os caracteres. Essa lista é inicializada através do método inicialistaTransicoes. Esse método executa o algoritmo para delimitar a região da placa onde estão localizados os caracteres. O método deve receber como parâmetro o intervalo em que os pixels serão percorridos, a diferença de intensidade entre os pixels desse intervalo para que eles possam ser inseridos na lista, a distância máxima entre os caracteres, a distância mínima entre os caracteres, a quantidade de pixels máxima por linha e a quantidade de pixels mínima por linha. Durante a execução desse método o objeto RetCaracteresPlaca é inicializado com um objeto da classe Retangulo que contém a localização da região dos caracteres. A lista listaPicosCaracteresMedia guarda a somatória da região dos caracteres da placa. Essa lista é inicializada pelo método inicializaListaPicosCaracteresMedia que recebe como parâmetro o tamanho em X da máscara que será percorrida. Esse método insere na lista os valores da somatória da região para que o método localizaCaracteresAnaliseGrafico execute o algoritmo de análise do gráfico da somatória da região delimitada dos caracteres. Esse método recebe como parâmetros o intervalo que será percorrido entre os valores da lista, a distância mínima e máxima entre cada elemento da lista, e a diferença de intensidade máxima e mínima entre os elementos da lista. A listaPicosCaracteresMedia representa o gráfico. Após ser executado esse método insere os pontos onde os caracteres possam estar na lista listaPontosCaracteres. Porém a lista listaPontosCaracteres também pode ser iniciada pelo método localizaCaracteresVariacaoVertical que inicia a lista caracteres também. Esse método executa o algoritmo que irá varrer a região delimitada dos caracteres na vertical, procurando onde existe maior variação de níveis de cinza. O método recebe como parâmetro a variação, e o intervalo em que os pixels serão percorridos.

```
/*
```

```
Classe responsável pela segmentação dos caracteres.
*/


public class ScannerImagemAssinatura {

	/*
	Declaração das listas que serão utilizadas no processo de segmentação de
	caracteres, algumas listas são utilizadas por métodos distintos para a
	segmentação .
	*/

	private final LinkedList<BufferedImage> caracteres = new
LinkedList<BufferedImage>();
	private final BufferedImage imagem; //imagem que será processada.
	private final LinkedList<Pixel> listaPicosCaracteresMedia = new
LinkedList<Pixel>();
	private final LinkedList<Pixel> listaPixelMeio = new
LinkedList<Pixel>();
	private final LinkedList<Ponto> listaPontosCaracteres = new
LinkedList<Ponto>();
	private final LinkedList<Pixel> listaTransicoes = new
LinkedList<Pixel>();
	private final Retangulo RetCaracteresPlaca = new Retangulo();

	/*
	Construtores, para caso seja passada uma imagem ou um objeto do tipo
placa.
	*/
	public ScannerImagemAssinatura(final BufferedImage im) {
		this.imagem = im;
	}

	public ScannerImagemAssinatura(final Placa placa) {
		this.imagem = placa.getImagem();
	}









	/*
	Método que desenha na placa a delimitação realizada na região dos
	caracteres utilizando o objeto RetCaracteresPlaca que já foi
	inicializado após a execução do método inicialistaTransicoes, esse
	método sofre sobrecarga,deixando a opção de utilizar a imagem da classe
	ou de outra placa.
	*/

	public BufferedImage DesenhaRetanguloCaracteres() {
		final BufferedImage im = new
		BufferedImage(this.imagem.getWidth(),this.imagem.getHeight(),
		BufferedImage.TYPE_INT_RGB);
		final Graphics g = im.getGraphics();
```

```
		g.drawImage(this.imagem, 0, 0, null);
		final Graphics2D g2 = (Graphics2D) g;
		g2.setStroke(new BasicStroke(2.0f));
		g.setColor(Color.red);
		final int xIni = this.RetCaracteresPlaca.getPontoInicial().getX();
		final int yIni = this.RetCaracteresPlaca.getPontoInicial().getY();
		final int xFim = this.RetCaracteresPlaca.getPontoFinal().getX();
		final int yFim = this.RetCaracteresPlaca.getPontoFinal().getY();
		g.drawLine(xIni, yIni, xFim, yIni);
		g.drawLine(xIni, yFim, xFim, yFim);
		g.drawLine(xIni, yIni, xIni, yFim);
		g.drawLine(xFim, yFim, xFim, yIni);
		g.dispose();
		return im;
	}

	public BufferedImage DesenhaRetanguloCaracteres(final Placa placa) {
		final BufferedImage im = new
		BufferedImage(placa.getImagem().getWidth(),
		placa.getImagem().getHeight(), BufferedImage.TYPE_INT_RGB);
		final Graphics g = im.getGraphics();
		g.drawImage(placa.getImagem(), 0, 0, null);
		final Graphics2D g2 = (Graphics2D) g;
		g2.setStroke(new BasicStroke(2.0f));
		g.setColor(Color.red);
		final int xIni = this.RetCaracteresPlaca.getPontoInicial().getX();
		final int yIni = this.RetCaracteresPlaca.getPontoInicial().getY();
		final int xFim = this.RetCaracteresPlaca.getPontoFinal().getX();
		final int yFim = this.RetCaracteresPlaca.getPontoFinal().getY();
		g.drawLine(xIni, yIni, xFim, yIni);
		g.drawLine(xIni, yFim, xFim, yFim);
		g.dispose();
		return im;
	}
	/*
	Método que lê a lista listaPontosCaracteres e desenha faixas nos pontos
	encontrados.
	*/
	public BufferedImage Fatia() {
		final int yIni = this.RetCaracteresPlaca.getPontoInicial().getY();
		final int yFim = this.RetCaracteresPlaca.getPontoFinal().getY();
		final BufferedImage im = new BufferedImage(this.imagem.getWidth(),
		this.imagem.getHeight(),BufferedImage.TYPE_INT_RGB);
		final Graphics g = im.getGraphics();
		g.drawImage(this.imagem, 0, 0, null);
		g.setColor(Color.red);
		for (final Ponto p : this.listaPontosCaracteres) {
			g.drawLine(p.getX(), yIni, p.getX(), yFim);
		}
		g.dispose();
		return im;
	}
	/*
	Método que insere na listaPixelMeio, os pixels que estão no meio da
	imagem.
	*/
	public LinkedList<Pixel> GetListaPixelsMeio() {
```

```
            for (int i = 0; i < this.imagem.getWidth(); i++) {
                  final Ponto pontoAtual = new Ponto(i,
                  this.imagem.getHeight() / 2);
                  final int tomAtual = this.imagem.getRaster().getSample(i,
                  this.imagem.getHeight() / 2, 0);
                  final Pixel pixelAtual = new Pixel(pontoAtual, tomAtual);
                  this.listaPixelMeio.add(pixelAtual);
            }

            return this.listaPixelMeio;
      }

      /*
      Método delimita a região dos caracteres da placa, salvando os pixels de
      transições em na lista listaTransicoes.
      */

      public LinkedList<Pixel> inicialistaTransicoes(final int
      distanciaPixels, final int diferencaIntensidade,final int
      distanciaCaracMax, final int distanciaCaracMin, final int
      quantidadePixMax, final int quantidadePixMin) {
            int tomAnterior = 0;
            int tomAtual = 0;
            int quantidadePixels = 0;
            boolean PrimeiroPontoIncializado = false;
            final LinkedList<Pixel> listaTemp = new LinkedList<Pixel>();
            tomAnterior = this.imagem.getRaster().getSample(0, 0, 0);
            Ponto pontoAnterior = new Ponto();
            for (int j = 0; j < this.imagem.getHeight(); j++) {
                  for (int i = 0; i < this.imagem.getWidth(); i = i +
distanciaPixels) {
                        tomAtual = this.imagem.getRaster().getSample(i, j, 0);
                        if (Math.abs(tomAnterior - tomAtual) >=
diferencaIntensidade) {
                              final Ponto pontoAtual = new Ponto(i, j);
                              final Pixel pixelAtual = new Pixel(pontoAtual,
tomAtual);
                              if (PrimeiroPontoIncializado == false) {
                                    PrimeiroPontoIncializado = true;
                                    listaTemp.add(pixelAtual);
                                    quantidadePixels++;
                              } else {
                                    if (pontoAnterior.getY() !=
pontoAtual.getY()) {
                                          if ((quantidadePixels >=
quantidadePixMin) &&
      (quantidadePixels <= quantidadePixMax)) {

                              this.listaTransicoes.addAll(listaTemp);
                                          }
                                          listaTemp.clear();
                                          quantidadePixels = 0;
                                    } else {
                                          if ((pontoAnterior.getX() !=
                                          pontoAtual.getX())   &&
                                          (pontoAtual.getX() -
```

```
                                                pontoAnterior.getX() >=
                                                distanciaCaracMin) &&
                                                (pontoAtual.getX() -
                                                pontoAnterior.getX() <=
                                                distanciaCaracMax)) {
                                                      listaTemp.add(pixelAtual);
                                                      quantidadePixels++;
                                                }
                                          }

                                    }
                                    pontoAnterior = new Ponto(i, j);
                              }
                              tomAnterior = this.imagem.getRaster().getSample(i, j,
0);
                        }
                  }
                  return this.listaTransicoes;
            }

            /*
            Após ser delimitada a região dos caracteres com os pixels inseridos na
            lista listaTransicoes, um retângulo com as dimensões da placa é criado e
            salvo no objeto RetCaracteresPlaca. Esse método só delimita os
            caracteres na horizontal
            */

            public Retangulo inicializaRetanguloCaracteres(final int erroYini, final
int erroYfim)     {
                  if (!this.listaTransicoes.isEmpty()) {
                        final int xInicial = 0;
                        final int xFinal = this.imagem.getWidth();
                        int yInicial;
                        int yFinal;
                        if (this.listaTransicoes.getFirst().getPonto().getY() -
                        erroYini > 0) {
                              yInicial =
                              this.listaTransicoes.getFirst().getPonto().getY() -
                              erroYini;
                        } else {
                              yInicial = 0;
                        }
                        if (this.listaTransicoes.getLast().getPonto().getY() +
                        erroYfim < this.imagem.getHeight()) {
                              yFinal =
                              this.listaTransicoes.getLast().getPonto().getY()+
                              erroYfim;
                        } else {
                              yFinal = this.imagem.getHeight();
                        }
                        final Ponto pontoInicial = new Ponto(xInicial, yInicial);
                        final Ponto pontoFinal = new Ponto(xFinal, yFinal);
                        this.RetCaracteresPlaca.setPontoInicial(pontoInicial);
                        this.RetCaracteresPlaca.setPontoFinal(pontoFinal);
                  }
                  return this.RetCaracteresPlaca;
            }
```

```
/*
Realiza a mesma função do método inicializaRetanguloCaracteres, e
delimita a região dos caracteres também na vertical
*/
public Retangulo inicializaRetanguloCaracteresTotal(final int erroYini,
final int erroYfim) {
	if (!this.listaTransicoes.isEmpty()) {
		int xInicial = this.imagem.getWidth();
		int xFinal = 0;
		int yInicial;
		int yFinal;
		if (this.listaTransicoes.getFirst().getPonto().getY() -
		erroYini > 0) {
		yInicial = this.listaTransicoes.getFirst().getPonto().getY()
		- erroYini;
		} else {
			yInicial = 0;
		}
		if (this.listaTransicoes.getLast().getPonto().getY() +
		erroYfim < this.imagem.getHeight()) {
			yFinal =
			this.listaTransicoes.getLast().getPonto().getY() +
			erroYfim;
		} else {
			yFinal = this.imagem.getHeight();
		}
		for (final Pixel pixelAtual : this.listaTransicoes) {
			if (xInicial > pixelAtual.getPonto().getX()) {
				xInicial = pixelAtual.getPonto().getX();
			}
			if (xFinal < pixelAtual.getPonto().getX()) {
				xFinal = pixelAtual.getPonto().getX();
			}
		}
		final Ponto pontoInicial = new Ponto(xInicial, yInicial);
		final Ponto pontoFinal = new Ponto(xFinal, yFinal);
		this.RetCaracteresPlaca.setPontoInicial(pontoInicial);
		this.RetCaracteresPlaca.setPontoFinal(pontoFinal);
	}
	return this.RetCaracteresPlaca;
}

/*
Método que analisa a lista listaPicosCaracteresMedia inicalizada pelo
método InicializaListaPicosCaracteresMedia em busca de caracteres ,
inserindo os ponto encontrados na lista listaPontosCaracteres
*/

public LinkedList<Ponto> LocalizaCaracteresAnaliseGrafico(final int
variacaoPixels, final int distanciaMin,  final int distanciaMax, final
int difIntensidadeMin, final int diIntensidadeMax) {
	Pixel pixelAnterior = this.listaPicosCaracteresMedia.getFirst();
	boolean primeiraiteracao = true;
		for (int x = 0; x < this.listaPicosCaracteresMedia.size(); x
		= x + variacaoPixels) {
```

```
                  final Pixel pixelAtual =
this.listaPicosCaracteresMedia.get(x);
                  if (primeiraiteracao == true) {
                        if ((Math.abs(pixelAtual.getTomCinza() -
                        pixelAnterior.getTomCinza()) >= difIntensidadeMin)
                        && (Math.abs(pixelAtual.getTomCinza() -
                        pixelAnterior.getTomCinza()) <= diIntensidadeMax)) {

      this.listaPontosCaracteres.add(pixelAtual.getPonto());
                              primeiraiteracao = false;
                        }
                  } else {
                        if ((Math.abs(pixelAtual.getTomCinza() -
                        pixelAnterior.getTomCinza()) >= difIntensidadeMin)
                        && (Math.abs(pixelAtual.getTomCinza() -
                        pixelAnterior.getTomCinza()) <= diIntensidadeMax)) {

      this.listaPontosCaracteres.add(pixelAtual.getPonto());
                        }
                  }
                  pixelAnterior = pixelAtual;
            }
            Ponto pontoAnterior;
            if (!this.listaPontosCaracteres.isEmpty()) {
                  pontoAnterior = this.listaPontosCaracteres.getFirst();
                  final LinkedList<Ponto> removidos = new LinkedList<Ponto>();
                  for (int i = 1; i < this.listaPontosCaracteres.size(); i++)
{
                        final Ponto p = this.listaPontosCaracteres.get(i);
                        if (((p.getX() - pontoAnterior.getX() >=
distanciaMin))
                                    && ((p.getX() - pontoAnterior.getX() <=
distanciaMax))) {
                              pontoAnterior = p;
                        } else {
                              removidos.add(p);
                        }
                  }
                  this.listaPontosCaracteres.removeAll(removidos);
            }
            return this.listaPontosCaracteres;

      }
      /*
      Método que analisa segmenta os caracteres pela varredura vertical na
      imagem, inserido os pontos em X dos caracteres encontrados na lista
      listaPontosCaracteres e os caracteres na lista caracteres.
      */

      public LinkedList<BufferedImage>
      LocalizaCaracteresVariacaoVertical(final Placa pla, final int
      variacaoMin,final int variacao) {
            int tomAnterior;
            int tomAtual;
            int totalVariacao = 0;
            for (int i = 0; i <
this.RetCaracteresPlaca.getPontoFinal().getX(); i++) {
```

```
                    tomAnterior = pla.getImagem().getRaster().getSample(i,
                    this.RetCaracteresPlaca.getPontoInicial().getY(), 0);
                    for (int j =
                    this.RetCaracteresPlaca.getPontoInicial().getY() + 1; j <
                    this.RetCaracteresPlaca.getPontoFinal() .getY(); j++) {
                          tomAtual = pla.getImagem().getRaster().getSample(i, j,
0);
                          if (Math.abs(tomAnterior - tomAtual) > variacaoMin) {
                                totalVariacao++;
                          }
                    }
                    if (totalVariacao >= variacao) {
                          this.listaPontosCaracteres.add(new Ponto(i,

      this.RetCaracteresPlaca.getPontoFinal().getY()));
                    }
                    totalVariacao = 0;
              }
              if (!this.listaPontosCaracteres.isEmpty()) {
                    int xAnterior =
this.listaPontosCaracteres.getFirst().getX();
                    int xAtual;
                    final LinkedList<Ponto> listaTemp = new LinkedList<Ponto>();
                    for (int i = 1; i < this.listaPontosCaracteres.size(); i++)
{
                          final Ponto p = this.listaPontosCaracteres.get(i);
                          xAtual = p.getX();
                          if ((xAtual == (xAnterior + 1)) && (!(i ==
                          (this.listaPontosCaracteres.size() - 1)))) {
                                listaTemp.add(p);
                          } else if (!listaTemp.isEmpty()){
                                final int largura = (listaTemp.getLast().getX()
                                - listaTemp.getFirst().getX()) + 1;
                                final int altura =
                                this.RetCaracteresPlaca.getPontoFinal().getY()
                                 -
this.RetCaracteresPlaca.getPontoInicial().getY();
                                final BufferedImage im = new
                                BufferedImage(largura, altura,
                                BufferedImage.TYPE_BYTE_GRAY);
                                int x = 0;
                                for (final Ponto pTemp : listaTemp) {
                                      int y = 0;
                                      for (int j =
                                      this.RetCaracteresPlaca.getPontoInicial().
                                      getY(); j <
                                      this.RetCaracteresPlaca.getPontoFinal().ge
                                      tY(); j++) {
                                            final int tomTemp =
                                            this.imagem.getRaster().getSample(pTe
                                            mp.getX(), j, 0);
                                            im.getRaster().setSample(x, y, 0,
tomTemp);

                                            y++;
                                      }
                                      x++;
                                }
```

```
                        this.caracteres.add(im);
                        listaTemp.clear();
                    }
                    xAnterior = p.getX();
                }
            }
            return this.caracteres;
        }

        /*
        Método que realize somatória da região delimitada da placa, através de
        uma máscara com a mesma altura da região encontrada e com a largura do
        parâmetro larguraCaractere. Esse método possui uma sobrecarga , para que
        possa ser utilizada a imagem da classe ou outra passada como parâmetro.
        */
        public LinkedList<Pixel> InicializaListaPicosCaracteresMedia(final int
        larguraCaractere) {
            int soma;
            int maiorPico = 0;
            final int Max = 255;
            int menorPico = Max;
            for (int i = 0; i < this.imagem.getWidth(); i++) {
                soma = 0;
                for (int x = i; (x < i + larguraCaractere) && (x <
                this.imagem.getWidth()); x++) {
                    for (int y =
                    this.RetCaracteresPlaca.getPontoInicial().getY(); y <
                    this.RetCaracteresPlaca.getPontoFinal().getY(); y++) {
                        soma = soma +
this.imagem.getRaster().getSample(x, y, 0);
                    }
                }
                if (soma > maiorPico) {
                    maiorPico = soma;
                }
                if (soma < menorPico) {
                    menorPico = soma;
                }
                final Ponto pontoAtual = new Ponto(i,
                this.RetCaracteresPlaca.getPontoFinal().getY());
                final int tomAtual = soma;
                final Pixel pixelAtual = new Pixel(pontoAtual, tomAtual);
                this.listaPicosCaracteresMedia.add(pixelAtual);
            }
            for (int i = 0; i < this.listaPicosCaracteresMedia.size(); i++) {
                final Pixel pixelMedia =
this.listaPicosCaracteresMedia.get(i);
                final int novoTom;
                if ((maiorPico - menorPico) != 0) {
                    novoTom = (Max * (pixelMedia.getTomCinza() -
                    menorPico)) / (maiorPico - menorPico);
                } else {
                    novoTom = 0;
                }
                this.listaPicosCaracteresMedia.get(i).setTomCinza(novoTom);
            }
```

```
            return this.listaPicosCaracteresMedia;
      }



      public LinkedList<Pixel> InicializaListaPicosCaracteresMedia(final int
      larguraCaractere, final Placa placa) {
            int soma;
            int maiorPico = 0;
            final int Max = 255;
            int menorPico = Max;
            for (int i = 0; i < placa.getImagem().getWidth(); i++) {
                  soma = 0;
                        for (int x = i; (x < i + larguraCaractere) && (x <
                        placa.getImagem().getWidth()); x++) {
                              for (int y =
                              this.RetCaracteresPlaca.getPontoInicial().getY()
                              ; y <
                              this.RetCaracteresPlaca.getPontoFinal().getY();
                              y++) {
                                    soma = soma +
                                    placa.getImagem().getRaster().getSample(x,
                                    y, 0);
                                    if (soma > maiorPico) {
                                          maiorPico = soma;
                                    }
                                    if (soma < menorPico) {
                                          menorPico = soma;
                                    }
                              }
                        }
                        final Ponto pontoAtual = new Ponto(i,
                        this.RetCaracteresPlaca.getPontoFinal().getY());
                        final int tomAtual = soma;
                        final Pixel pixelAtual = new Pixel(pontoAtual, tomAtual);
                        this.listaPicosCaracteresMedia.add(pixelAtual);
            }
            for (int i = 0; i < this.listaPicosCaracteresMedia.size(); i++) {
                  final Pixel pixelMedia =
this.listaPicosCaracteresMedia.get(i);
                  int novoTom;
                  if ((maiorPico - menorPico) != 0) {
                        novoTom = (Max * (pixelMedia.getTomCinza() -
                        menorPico)) / (maiorPico - menorPico);
                  } else {
                        novoTom = 0;
                  }
                  this.listaPicosCaracteresMedia.get(i).setTomCinza(novoTom);
            }
            return this.listaPicosCaracteresMedia;
      }

}
```

A classe Grafico realiza a montagem de gráficos e plotações recebendo como parâmetro listas de pontos ou pixels que foram processados pela classe ScanerImagemAssinatura e exibe esses gráficos e plotações na tela.

```
/*
	Classe Gráfico responsável por mostrar as listas processadas na tela em
	forma de gráficos ou plotações.
*/
public class Grafico extends JFrame {
	/*
	Atributos
	*/

	private final int altura;
	public BufferedImage image;
	private final int largura;
	private final LinkedList<Pixel> listaPixels = new LinkedList<Pixel>();

	/*
	Construtores
	*/

	public Grafico(final int largura, final int altura, final String nome) {
		super(nome);
		this.largura = largura;
		this.altura = altura;
	}

	public Grafico(final LinkedList<Pixel> listaPixels, final int largura,
	final int altura, final String nome) {
		super(nome);
		this.largura = largura;
		this.altura = altura;
		this.listaPixels.addAll(listaPixels);
	}

	/*
	Inicia a classe para plotação de valores. Método com sobrecarga.
	*/

	public void iniciaPlot() {
		final BufferedImage grafico = new BufferedImage(this.largura,
		this.altura, BufferedImage.TYPE_INT_RGB);
		for (final Pixel pixelNew : this.listaPixels) {
			grafico.getRaster().setSample(pixelNew.getPonto().getX(),
			pixelNew.getPonto().getY(), 0, 255);
		}

		final JLabel label = new JLabel(new ImageIcon(grafico));
		this.setLayout(new FlowLayout());
		this.add(label);
		this.image = grafico;
	}

	public void iniciaPlot(final LinkedList<Ponto> listaPontos) {
```

```
            final BufferedImage grafico = new BufferedImage(this.largura,
            this.altura, BufferedImage.TYPE_INT_RGB);
            for (final Ponto pixelNew : listaPontos) {
                  grafico.getRaster().setSample(pixelNew.getX(),
pixelNew.getY(), 0, 255);
            }
            final JLabel label = new JLabel(new ImageIcon(grafico));
            this.setLayout(new FlowLayout());
            this.add(label);
            this.image = grafico;
      }

      /*
      Inicia a classe para exibir um gráfico.
      */

      public void iniciaSequencia() {
            final BufferedImage grafico = new BufferedImage(this.largura,
            this.altura, BufferedImage.TYPE_INT_RGB);
            final Graphics g = grafico.getGraphics();
            g.setColor(Color.BLUE);
            int i = 0;
            for (final Pixel pixelNew : this.listaPixels) {
                  g.drawLine(i, grafico.getHeight(), i, pixelNew.getTomCinza()
                  + this.altura - 255);
                  i++;
            }
            g.dispose();
            final JLabel label = new JLabel(new ImageIcon(grafico));
            this.setLayout(new FlowLayout());
            this.add(label);
            this.image = grafico;
      }

}
```

O Método caracteresFrame imprime na tela os caracteres encontrados

```
private void caracteresFrame(final LinkedList<BufferedImage> listaCaracteres)
{
            final JFrame caracteresFrame = new JFrame("Caracteres");
            caracteresFrame.setLayout(new FlowLayout());
            for (final BufferedImage im : listaCaracteres) {
                  final JLabel label = new JLabel(new ImageIcon(im));
                  caracteresFrame.add(label);
            }
            caracteresFrame.setVisible(true);
            caracteresFrame.setBackground(Color.BLUE);
            caracteresFrame.setSize(200, 100);

      }
```

O Método Localiza realiza a localização das placas e dos caracteres e os imprime na tela, fazendo todas as classes do projeto se comunicar.

```
public BufferedImage Localiza() {
            final int largura = 240;
            final int altura = 40;
            final BufferedImage imEsqueleto = this.Esqueleto();
            final BufferedImage im = new
            BufferedImage(this.getImage().getWidth(),
            this.getImage().getHeight(),
            BufferedImage.TYPE_BYTE_GRAY);
            final ScannerImagemMedia imagemMedia = new
            ScannerImagemMedia(imEsqueleto, largura, altura, im);
            imagemMedia.scanear();
            final Ponto p = imagemMedia.PontoMaiorIntensidade();
            final BufferedImage imPassa = this.PassaAltaVertical();
            final ScannerImagemMaiorSoma maiorSoma = new
            ScannerImagemMaiorSoma(imPassa, largura, altura, p, 2, 255, 55,
             ScannerImagemMaiorSoma.JANELA_INTEIRA, -7, -5);
            maiorSoma.scanear();
            final Placa placa = new Placa(this.image,
            maiorSoma.getPontoSomaMax(), 255, 55, -7, -5);
            final ScannerImagemAssinatura assinatura = new
            ScannerImagemAssinatura(placa.getImagem());
            final Grafico graficoMeio = new
            Grafico(assinatura.GetListaPixelsMeio(), 255, 300, "Gráfico do
            Meio");
            graficoMeio.iniciaSequencia();
            graficoMeio.setVisible(true);
            graficoMeio.setSize(350, 350);

      graficoMeio.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
            final Grafico grafico = new
            Grafico(assinatura.inicialistaTransicoes(5, 60, 38, 10, 28, 7),
            255, 55,
            "Plotação das Transições");
            grafico.iniciaPlot();
            grafico.setVisible(true);
            grafico.setSize(300, 100);

      grafico.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
            assinatura.inicializaRetanguloCaracteres(3, 5);
            final Grafico graficoMedia = new
            Grafico(assinatura.InicializaListaPicosCaracteresMedia(5), 255,
            300,
            "Gráfico da Média dos Caracteres");
            graficoMedia.iniciaSequencia();
            graficoMedia.setVisible(true);
            graficoMedia.setSize(350, 350);
            this.processed = assinatura.DesenhaRetanguloCaracteres();
            final Placa placaPassa = new
Placa(this.autoThreshold(placa.getImagem()));
            final LinkedList<BufferedImage> listaImagens =
            assinatura.LocalizaCaracteresVariacaoVertical(placaPassa, 1, 1);
            this.caracteresFrame(listaImagens);
```

```
            final JFrame telaPlaca = new JFrame("Placa");
            final JLabel label = new JLabel(new
ImageIcon(placaPassa.getImagem()));
            telaPlaca.setLayout(new FlowLayout());
            telaPlaca.add(label);
            telaPlaca.setVisible(true);
            telaPlaca.setSize(placaPassa.getImagem().getWidth() + 50,
            placaPassa.getImagem().getHeight() + 50);
            final JFrame telaPlacaFatiada = new JFrame("Placa Fatiada");
            final JLabel labelfatiado = new JLabel(new
ImageIcon(assinatura.Fatia()));
            telaPlacaFatiada.setLayout(new FlowLayout());
            telaPlacaFatiada.add(labelfatiado);
            telaPlacaFatiada.setVisible(true);
            telaPlacaFatiada.setSize(placaPassa.getImagem().getWidth() + 50,
            placaPassa.getImagem().getHeight() + 50);
            return assinatura.DesenhaRetanguloCaracteres();


      }
```

APÊNDICE H

Esse apêndice mostra as classes e métodos utilizados no reconhecimento de caracteres. O reconhecimento de caracteres é composto de nove classes, onde cada classe é responsável por uma parte no processo. A primeira classe mostrada é a classe Amostra.

A classe amostra possui os seguintes métodos:

```
/*
    * Construtor
    * @atributos: Serve apenas para saber quantos atributos essa amostra
possui
    * @valores: Vetor contendo o valor de cada atributo
    * @valorDesejado: Serve para informar para a rede neural se essa amostra
deve ser reconhecida ou não pela mesma
    */
        public Amostra(final Atributos atributos, final int[] valores, final
int valorDesejado);

/*
    * Retorna o vetor com os valores dos atributos
    */
        public int[] getValores();
```

A classe Atributos serve apenas para armazenar quantos atributos existem em uma amostra.

```
/*
    * Construtor
    * @qtdAtributos: Serve para setar a quantidade de atributos de uma amostra
    */
        public Atributos(final int qtdAtributos);
```

A interface IConfiguracaoPerceptron serve para realizar uma configuração padrão dos perceptrons do sistema.

```
public interface IConfiguracaoPerceptron {

   public Atributos getAtributos();

   public double getFormatedDouble(double valor);

   public double[] getPesos();

   public double getTaxaAprendizado();

      }
```

A classe perceptron é uma das classes principais do reconhecimento dos caracteres, pois é a representação computacional de um neurônio.

A classe perceptron possui os seguintes campos:

```
private final IConfiguracaoPerceptron configuracaoPerceptron;
private double pesoBias = 0;
private final double[] pesos;
```

Sendo que o primeiro Server para armazenar uma referencia para configuração do perceptron, o segundo Server para armazenar o peso do bias, já que este é tratado diferente dos outros pesos.

```
/*
    * Construtor
    * @configuracaoPerceptron armazena uma referencia para a configuração do
perceptron
    */
        public Perceptron(final IConfiguracaoPerceptron
configuracaoPerceptron);

/*
    * Função de ativação, onde caso o valor seja maior que zero, retorna 1 e
    * caso contrario retorna 0
    */
   public int getNet(final double valor) {
      return valor > 0 ? 1 : 0;
         }


/*
    * getSaidaCalculada: é o metodo que retorna o que foi reconhecido por este
    * perceptron de acordo com os atributos passados por parametro
    */
   public double getSaidaCalculada(final int[] valores) {
      return this.pesoBias +
this.configuracaoPerceptron.getFormatedDouble(this.somatorioPesosValores(valor
es));
         }

/*
      O método abaixo é o responsável pelo o reajuste dos pesos de um
perceptron, fazendo com que o mesmo “aprenda”
    * recalcularPesoBias: método responsável por realizar o ajuste dos pesos
de
    * acordo com o erro e os valores
    */
```

```
   private void recalcularPesos(final int[] valores, final int saidaCalculada,
final int saidaDesejada) {
      for (int i = 0; i < valores.length; i++) {
         this.pesos[i] += this.configuracaoPerceptron.getTaxaAprendizado() *
(saidaDesejada - saidaCalculada) * valores[i];
         this.pesos[i] =
this.configuracaoPerceptron.getFormatedDouble(this.pesos[i]);
      }
      this.recalcularPesoBias(saidaCalculada, saidaDesejada);

         }

/*
    * retorna a soma dos valores contra os pesos do perceptron
    */
   private double somatorioPesosValores(final int valores[]) {
      double resultado = 0;
      for (int i = 0; i < valores.length; i++) {
         resultado += this.pesos[i] * valores[i];
      }
      return resultado;
         }

/*
    * Metodo principal o qual atravez de uma amostra é decido se precisa fazer
    * um ajuste nos pesos do perceptron
    */
   public boolean treinarAmostra(final Amostra amostra) {
      final int[] valores = amostra.getValores();
      boolean resultado = true;
      final int saidaCalculada = this.getNet(this.getSaidaCalculada(valores));
      final int saidaDesejada = amostra.getValorDesejado();

      if (saidaCalculada != saidaDesejada) {
         resultado = false;
         this.recalcularPesos(valores, saidaCalculada, saidaDesejada);
      }
      return resultado;
         }
```

A classe RedeNeural serve para poder gerenciar o perceptron e armazenar mais algumas informações como em nosso exemplo o caractere reconhecido, assim eu consigo saber qual o caractere que essa rede neural é treinada para reconhecer.

```
/*
    * Construtor
    *
    * @configuracaoPerceptron: são os parametros iniciais
    *
    * @caracterReconhecido: o caractere que esta rede neural reconhece
    */
   public RedeNeural(final IConfiguracaoPerceptron configuracaoPerceptron,
final char caracterReconhecido);
```

```
   /*
    * Retorna o caractere que a rede reconhece
    */
   public char getCaracterReconhecido() {
      return this.caracterReconhecido;
   }

   public Perceptron getPerceptron() {
      return this.perceptron;
   }

   public boolean isCaracterReconhecido(final Amostra amostra) {
      return
(this.perceptron.getNet(this.perceptron.getSaidaCalculada(amostra.getValores()
)) == 1);
   }

   public boolean isCaracterReconhecido(final int[] valores) {
      return
(this.perceptron.getNet(this.perceptron.getSaidaCalculada(valores)) == 1);
         }
```

A classe GerenciadorRedeNeural é a classe responsável pelo gerenciamento das redes neurais, assim como descobrir qual rede neural é responsável por reconhecer um determinado caractere e manter a comunicação entre as redes.

```
public class GerenciadorRedeNeural implements IConfiguracaoPerceptron {

   private final Atributos atributos;
   private DocumentBuilder db;
   private DocumentBuilderFactory dbf;
   private final DecimalFormat decimalFormat;
   private Document doc;
   final File gerenciadorRedeNeuralXml;
   private final LinkedList<RedeNeural> redes;

   /*
    * Construtor: responsável pela inicialização das redes neurais e arquivo
XML
    */
   public GerenciadorRedeNeural() throws ParserConfigurationException,
SAXException, IOException {
      this.redes = new LinkedList<RedeNeural>();
      this.atributos = new Atributos(Configuracao.getLarguraIdeal() *
Configuracao.getAlturaIdeal());
      this.decimalFormat = new DecimalFormat("0.0000");
      this.gerenciadorRedeNeuralXml = new File("GerenciadorRedeNeural.xml");
      this.construirArquivoXml();
   }

   /*
```

```
     * Retorna a rede neural de acordo com o caractere passado
     */
    private RedeNeural acharRedeNeural(final char caracter) {
        for (final RedeNeural r : this.redes) {
            if (r.getCaracterReconhecido() == caracter) {
                return r;
            }
        }
        return null;

    }

    /*
     * Instancia uma rede neiral pelo nó passado do XML
     */
    private void adicionarRedeNeural(final Element redeNeural) {
        final RedeNeural neural = new RedeNeural(new IConfiguracaoPerceptron() {

            @Override
            public Atributos getAtributos() {
                return GerenciadorRedeNeural.this.getAtributos();
            }

            @Override
            public double getFormatedDouble(final double valor) {
                return GerenciadorRedeNeural.this.getFormatedDouble(valor);
            }

            @Override
            public double[] getPesos() {
                final double[] resultado = new
double[GerenciadorRedeNeural.this.getAtributos().getQtdAtributos()];
                Node pesos = null;
                for (int i = 0; i < redeNeural.getChildNodes().getLength(); i++) {
                    if
(redeNeural.getChildNodes().item(i).getNodeName().equalsIgnoreCase("pesos")) {
                        pesos = redeNeural.getChildNodes().item(i);
                        break;
                    }
                }

                int r = 0;
                for (int i = 0; i < pesos.getChildNodes().getLength(); i++) {
                    try {
                        resultado[r] =
Double.parseDouble(pesos.getChildNodes().item(i).getTextContent());
                        r++;
                    }
                    catch (final Exception e) {
                        continue;
                    }
                }
                return resultado;
            }

            @Override
            public double getTaxaAprendizado() {
```

```
            return GerenciadorRedeNeural.this.getTaxaAprendizado();
         }
      }, redeNeural.getAttribute("caracterReconhecido").charAt(0));
      this.redes.add(neural);
   }

   /*
    * Adiciona uma rede neural na estrutura XML definida
    */
   private void adicionarRedeNeural(final Element redesNeurais, final
RedeNeural neural) {
      final Element redeNeural = this.doc.createElement("redeNeural");
      redesNeurais.appendChild(redeNeural);
      redeNeural.setAttribute("caracterReconhecido",
String.valueOf(neural.getCaracterReconhecido()));
      final Element pesos = this.doc.createElement("pesos");
      redeNeural.appendChild(pesos);

      for (int i = 0; i < neural.getPerceptron().getPesos().length; i++) {
         final Element peso = this.doc.createElement("p" + i);

peso.setTextContent(String.valueOf(neural.getPerceptron().getPesos()[i]));
         pesos.appendChild(peso);
      }

      this.redes.add(neural);
   }

   /*
    * Atualiza os dados de uma rede neural no arquivo XML
    */
   public void atualizarRedeNeural(final RedeNeural neural) throws
TransformerException, IOException {
      final Element redesNeurais = this.doc.getDocumentElement();
      for (int i = 0; i < redesNeurais.getChildNodes().getLength(); i++) {
         final Node nodeRede = redesNeurais.getChildNodes().item(i);
         if (nodeRede.getAttributes() == null) {
            continue;
         }
         if
(nodeRede.getAttributes().getNamedItem("caracterReconhecido").getTextContent()
.toUpperCase().charAt(0) == neural
                  .getCaracterReconhecido()) {
            for (int p = 0; p < nodeRede.getChildNodes().getLength(); p++) {
               final Node nodeP = nodeRede.getChildNodes().item(p);
               if (nodeP.getNodeName().equalsIgnoreCase("pesos")) {
                  for (int pi = 0; pi < nodeP.getChildNodes().getLength();
pi++) {
                     try {
                        final int index =
Integer.parseInt(nodeP.getChildNodes().item(pi).getNodeName().replace("p",
""));

nodeP.getChildNodes().item(pi).setTextContent(String.valueOf(neural.getPercept
ron().getPesos()[index]));
                     }
                     catch (final Exception exception) {
```

```
                        continue;
                    }

                }
            }
        }
        this.salvar();
        return;
      }
    }

  }

  /*
   * Instancia um arquivo xml e cria as redes neurais de acordo com o mesmo
   */
  private void construirArquivoXml() throws ParserConfigurationException,
SAXException, IOException {
    this.dbf = DocumentBuilderFactory.newInstance();
    this.db = this.dbf.newDocumentBuilder();

    if (this.gerenciadorRedeNeuralXml.exists()) {
      try {
        this.doc = this.db.parse(this.gerenciadorRedeNeuralXml);
      }
      catch (final Exception e) {
        this.doc = this.db.newDocument();
      }
    } else {
      this.doc = this.db.newDocument();
    }

    this.construirPerceptrons();
  }

  /*
   * construirPerceptrons: constrói todas as redes neurais de acordo com o
XLM,
   * caso não tenha, crie e adicione no arquivo XML
   */
  private void construirPerceptrons() {
    Element redesNeurais = this.doc.getDocumentElement();
    if (redesNeurais == null) {
      redesNeurais = this.doc.createElement("redesNeurais");
      this.doc.appendChild(redesNeurais);
    }

    final NodeList redeNeuralList =
redesNeurais.getElementsByTagName("redeNeural");

    /*
     * Instancia as redes neurais de acordo com o arquivo XML
     */
    for (int i = 0; i < redeNeuralList.getLength(); i++) {
      final Element redeNeural = (Element) redeNeuralList.item(i);
      this.adicionarRedeNeural(redeNeural);
    }
```

```
        /*
         * Constroi as redes de 0 a 9
         */
        for (char c = 'A'; c <= 'Z'; c++) {
            if (this.acharRedeNeural(c) == null) {
                this.adicionarRedeNeural(redesNeurais, new RedeNeural(this, c));
            }
        }

        /*
         * Constroi as redes de 0 a 9
         */
        for (char c = '0'; c <= '9'; c++) {
            if (this.acharRedeNeural(c) == null) {
                this.adicionarRedeNeural(redesNeurais, new RedeNeural(this, c));
            }
        }

    }

    @Override
    public Atributos getAtributos() {
        return this.atributos;
    }

    /*
     * Formata o valor de acordo com a mascara definida
     */
    @Override
    public double getFormatedDouble(final double valor) {
        return Double.parseDouble(this.decimalFormat.format(valor).replace(",",
".","."));
    }

    private double getPesoAleatorio() {
        double resultado = Math.random();
        while ((resultado < this.getPesoMinimo()) || (resultado >
this.getPesoMaximo())) {
            resultado = Math.random();
        }
        return this.getFormatedDouble(resultado);
    }

    protected double getPesoMaximo() {
        return 0.9;
    }

    protected double getPesoMinimo() {
        return 0.1;
    }

    /*
     * Gera um vetor de pesos alietarios
     */
    @Override
    public double[] getPesos() {
```

```
      final double[] resultado = new
double[this.getAtributos().getQtdAtributos()];
      for (int i = 0; i < resultado.length; i++) {
         resultado[i] = this.getPesoAleatorio();
      }

      return resultado;

   }

   /*
    * Retorna a rede neural de acordo com o caractere passado como argumento.
    */
   public RedeNeural getRedeNeural(final char caracter) {
      for (final RedeNeural r : this.redes) {
         if (r.getCaracterReconhecido() == caracter) {
            return r;
         }
      }
      final RedeNeural novaRede = new RedeNeural(this, caracter);
      this.redes.add(novaRede);
      return novaRede;
   }

   /*
    * Retorna as redes neurais instanciadas
    */
   public LinkedList<RedeNeural> getRedes() {
      return this.redes;
   }

   /*
    * getTaxaAprendizado: retorna a taxa de aprendizado padrao para cada novo
    * perceptron
    */
   @Override
   public double getTaxaAprendizado() {
      return 0.1;
   }

   /*
    * reconhecerCaracter: retorna uma lista das redes neurais que reconheceram
a
    * imagem passada como argumento
    */
   public LinkedList<Character> reconhecerCaracter(final BufferedImage
bufferedImage) {
      final LinkedList<Character> resultado = new LinkedList<Character>();
      final Amostra amostra = BufferedImageUtils.toAmostra(bufferedImage, 1);

      for (final RedeNeural r : this.redes) {
         if (r.isCaracterReconhecido(amostra)) {
            resultado.add(r.getCaracterReconhecido());
         }
      }
      return resultado;
```

```
   }

   /*
    * Salva todas as informação das redes neurais no arquivo XML
    */
   public File salvar() throws TransformerException, IOException {
      final TransformerFactory transformerFactory =
TransformerFactory.newInstance();
      final Transformer transformer = transformerFactory.newTransformer();
      transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
      transformer.setOutputProperty(OutputKeys.INDENT, "yes");
      final FileOutputStream fileOutputStream = new
FileOutputStream(this.gerenciadorRedeNeuralXml);
      final StreamResult result = new StreamResult(fileOutputStream);
      final DOMSource domSource = new DOMSource(this.doc);
      transformer.transform(domSource, result);
      fileOutputStream.close();
      return this.gerenciadorRedeNeuralXml;

   }
}
```

A classe BtnCaracterReconhecido serve para fazer a integração do usuário com a rede neural, com esta classe é possível realizar o treinamento das redes, um treinamento supervisionado, onde o usuário fala para a rede o que é a imagem em questão. Quando o usuário pedir para a rede treinar uma imagem, é mostrada para todas as redes e cada uma é treinada de acordo com a imagem.

```
public BtnCaracterReconhecido(final GerenciadorRedeNeural
gerenciadorRedeNeural, final LinkedList<Character> caracterReconhecido,
            final BufferedImage bufferedImage) {
      this.caracterReconhecido = caracterReconhecido;
      this.gerenciadorRedeNeural = gerenciadorRedeNeural;
      this.bufferedImage = bufferedImage;
   }

   @Override
   public void actionPerformed(final ActionEvent e) {
      try {

         if (JOptionPane.showConfirmDialog(null, "O caractere reconhecido foi
: " + Arrays.toString(this.caracterReconhecido.toArray())
                  + " está correto?", "Reconhecimento de Caracteres",
JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE) ==
JOptionPane.NO_OPTION) {
            final char resposta = JOptionPane.showInputDialog("Entre com o
caractere correto").toUpperCase().charAt(0);

            final Amostra amostraVerdadeira =
BufferedImageUtils.toAmostra(this.bufferedImage, 1);
            final Amostra amostraFalsa =
BufferedImageUtils.toAmostra(this.bufferedImage, 0);
            this.gerenciadorRedeNeural.getRedeNeural(resposta);
```

```
            for (int t = 0; t < 5; t++) {
               new Thread(new Runnable() {

                  @Override
                  public void run() {
                     for (final RedeNeural neural :
BtnCaracterReconhecido.this.gerenciadorRedeNeural.getRedes()) {
                        if (neural.getCaracterReconhecido() == resposta) {
                           new Thread(new Runnable() {

                              @Override
                              public void run() {
                                 int i = 0;
                                 while
((!neural.getPerceptron().treinarAmostra(amostraVerdadeira)) && (i++ < 1000))
{
                                    ;
                                 }
                              }
                           }).start();

                        } else {
                           new Thread(new Runnable() {

                              @Override
                              public void run() {
                                 int i = 0;
                                 while
((!neural.getPerceptron().treinarAmostra(amostraFalsa)) && (i++ < 1000)) {
                                    ;
                                 }
                              }
                           }).start();

                        }
                     }

                  }
               }).start();
            }

            for (final RedeNeural neural :
this.gerenciadorRedeNeural.getRedes()) {
               this.gerenciadorRedeNeural.atualizarRedeNeural(neural);
            }

            this.caracterReconhecido.clear();
            this.caracterReconhecido.add(resposta);

         }
      }
      catch (final Exception exception) {
         JOptionPane.showMessageDialog(null, exception.getMessage(),
"Atenção", JOptionPane.WARNING_MESSAGE);
      }

         }
```