UNIVERSIDADE FEDERAL DE ITAJUBÁ

PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Sistema de Visão Computacional Embarcado para Identificação de Placas de Trânsito

Diógenes da Silva Leonel

Itajubá, Novembro de 2016

UNIVERSIDADE FEDERAL DE ITAJUBÁ
PROGRAMA DE PÓS GRADUAÇÃO EM
CIÊNCIA E TECNOLOGIA DA COMPUTAÇÃO

Diógenes da Silva Leonel

Sistema de Visão Computacional Embarcado para Identificação de Placas de Trânsito

Dissertação submetida ao Programa de Pós-Graduação em Ciência e Tecnologia da Computação como parte dos requisitos para obtenção do Título de Mestre em Ciência e Tecnologia da Computação

Área de Concentração: Matemática da Computação

Orientador: Prof. Dr. Carlos Henrique Valério de Moraes

Novembro de 2016
Itajubá - MG

Agradecimentos

A Deus, por me proporcionar saúde e inteligência para trilhar este caminho que me propuz.

A meus pais, Denilson e Jane, por me ensinarem o valor e importância dos estudos, além do apoio nos momentos mais difíceis. Também por não deixarem desistir dos meus sonhos e o total apoio à minhas decisões.

A minha irmã, Larissa, pelo incentivo de continuar batalhando.

A minha namorada, Patricia, pela companhia, encorajamento e carinho durante todo o processo do mestrado.

Ao meu orientador, Carlos Henrique, pela paciência e por ter acreditado no meu potencial, mesmo quando eu já não acreditava mais. Também por dispor de seu tempo para esclarecer dúvidas e conversar sobre os mais diversos tipos de assuntos.

Ao Rodrigo Maximiano, pela oportunidade de trabalhar com sistemas embarcados e por sanar minhas dúvidas a respeito desse tema.

A CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo suporte financeiro durante todo o período de realização deste mestrado.

A nossa maior glória não reside no fato de nunca cairmos, mas sim em levantarmo-nos sempre depois de cada queda.
Oliver Goldsmith

Resumo

Um sistema de visão para identificação de sinais de trânsito coleta informações importantes para que o motorista tome decisões no momento correto, tornando a direção segura e, consequentemente, reduzindo o número de acidentes nas vias. Para que isso ocorra é necessária sua popularização, fato que somente irá se concretizar com a redução do custo de aquisição. Este trabalho lida com a redução do custo de tais sistemas através da utilização de um sistema embarcado acessível e de soluções abertas. É utilizado como hardware embarcado a Beaglebone Black, uma SBC de baixo custo, compacta e eficiente energeticamente. Como metodologia de visão computacional é utilizado a junção do *framework* Viola & Jones com um processo de segmentação para a fase de detecção. As regiões provenientes dessa fase são classificadas através de múltiplas SVM lineares utilizando vetores provenientes da combinação das técnicas HOG e LDA. Ambas fases são testadas em bases de dados, GTSDB e GTSRB, publicas desafiadoras, além de testes em vídeo para a fase de detecção. Destes testes foram extraídos métricas de tempo e desempenho, que demonstram a aplicabilidade do sistema em soluções comerciais de baixo custo.

Abstract

A vision system for traffic signs identification collects information for the driver to make decisions at the right time, making driving safer and reducing the number of road accidents. For this to occur it is necessary its popularization, which will only come true by reducing the cost of acquisition. This work deals with the reduction of the cost of such systems through the use of an accessible embedded system and open solutions. The Beaglebone Black, a low-cost, compact and energy-efficient SBC is used as embedded hardware. As a computer vision methodology for detection phase, the framework Viola & Jones is used in conjuction with a segmentation process. The regions from this phase are classified through multiple linear SVM using vectors from the combination of HOG and LDA. Both phases are tested in challenging public databases, GTSDB and GTSRB, in addition to video tests for the detection phase. From these tests were extracted time and performance metrics, which shows the applicability in low-cost commercial solutions.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário p. 13

1 Introdução p. 15

1.1 Objetivos . . . . . p. 17

1.2 Revisão Bibliográfica . . . . . p. 18

1.2.1 Soluções de Hardware . . . . . p. 18

1.2.2 Técnicas de Visão Computacional . . . . . p. 19

1.3 Organização do Trabalho . . . . . p. 23

2 Fundamentação Teórica p. 24

2.1 Imagem Digital . . . . . p. 24

2.2 Equalização de Histogramas . . . . . p. 32

2.3 Filtro Mediana . . . . . p. 37

2.4 Componentes Conectadas . . . . . p. 43

2.5 Detecção . . . . . p. 48

2.5.1 Framework Viola e Jones . . . . . p. 48

2.5.2 *Multi-Block Local Binary Pattern* . . . . . p. 60

2.6 Reconhecimento . . . . . p. 64

2.6.1 Histograma de Gradientes Orientados . . . . . p. 64

2.6.2 Analise de Discriminante Linear . . . . . p. 68
2.6.3 Máquina de Vetores de Suporte Linear . . . . . p. 73
2.7 Sistemas Embarcados . . . . . p. 82
2.8 Análise de Desempenho . . . . . p. 88

3 Desenvolvimento p. 94
3.1 Segmentação . . . . . p. 97
3.2 Detecção . . . . . p. 102
3.3 Reconhecimento . . . . . p. 104
3.4 Integração com Sistema Embarcado . . . . . p. 107

4 Resultados p. 115
4.1 Bases de Dados . . . . . p. 115
4.2 Resultados Detecção . . . . . p. 119
4.3 Resultados Reconhecimento . . . . . p. 124

5 Conclusão p. 128
5.1 Trabalhos Futuros . . . . . p. 129

Apêndice A – Ferramentas de Desenvolvimento p. 130
A.1 OpenCV . . . . . p. 130
A.2 QT . . . . . p. 132

Anexo A – Valores da Matriz Confusa p. 134

Anexo B – Exemplos de Detecção p. 136

Anexo C – Exemplos de Reconhecimento p. 140
C.1 Acertos . . . . . p. 140
C.2 Erros . . . . . p. 142

Anexo D – Análise em vídeo da Detecção p. 144

D.1 Proibitório . . . . . p. 144

D.2 Mandatório . . . . . p. 145

D.3 Perigo . . . . . p. 146

Referências p. 147

Lista de Figuras

1 Sistema completo para Identificação de Placas de Trânsito. . . . . p. 16

2 Etapas de um sistema de reconhecimento de placa de trânsito. . . . . p. 20

3 Representação matricial de uma imagem. . . . . p. 27

4 Canais para um imagem no espaço de cores RGB. FONTE: (PHOTOS..., 2016) . . . . . p. 28

5 Espaço de cores RGB. . . . . p. 29

6 Espaço de cores HSV. . . . . p. 31

7 Imagens de contrastes diferentes e respectivos histogramas. FONTE: (HOUBEN et al., 2013) . . . . . p. 33

8 Exemplo de aplicação de histograma em uma imagem de baixo contraste. FONTE: (STALLKAMP et al., 2012) . . . . . p. 34

9 Gráficos de uma equalização de contraste apresentado na Tabela 2. . . . p. 36

10 Possíveis conectividades de um pixel. . . . . p. 38

11 Exemplo do processo de convolução para uma imagem fictícia. . . . . p. 39

12 Comparativo da remoção de ruídos, para uma vizinhança 3×3 entre os filtros mediana, média, moda. FONTE:(HUTCHISON, 2016) . . . . . p. 41

13 Cálculo do filtro de mediana para uma vizinhança 3×3 com conectividade 8. . . . . p. 42

14 Filtro mediana aplicado com três tamanhos diferentes de vizinhança (*kernel*). FONTE:(HUTCHISON, 2016) . . . . . p. 42

15 Componentes conectada (a) de uma imagem binária (b). . . . . p. 44

16 Ilustração da existência de um cainho em uma imagem binária. . . . . p. 44

17 Possíveis conectividades de um pixel. . . . . p. 45

18 Regiões R1 e R2 só estão conectadas se considerado uma adjacência-8. . . p. 45

19 Passos de um algoritmo de componentes conectadas. FONTE: Shah (1997). . . . . p. 46

20 Módulos do *framework* Viola & Jones . . . . . p. 49

21 Haar-like básicos para uma janela de detecção. . . . . p. 50

22 Exemplo de cálculo e uso de uma imagem integral. Valores em negrito soma-se enquanto que, itálico subtrai . . . . . p. 51

23 Funcionamento do algoritmo *Adaboost*. Baseado em (SZELISKI, 2010). . . p. 54

24 Algoritmo *Adaboost*. Adaptado de: (VIOLA; JONES, 2004). . . . . p. 55

25 Algoritmo de criação da cascata de classificadores . Adaptado de: (VIOLA; JONES, 2004). . . . . p. 57

26 Cascata de classificadores proposto por (VIOLA; JONES, 2001). Subjanelas negativas (N) são rejeitadas, enquanto que positivas (V) passam para o próximo estagio. . . . . p. 58

27 Características *haar-like* extendidas (LIENHART; MAYDT, 2002), onde: (a) são de borda,(b) de linha,(c) de ponto. . . . . p. 59

28 Exemplo de cálculo do descritor LBP. Baseado em (OJALA; PIETIKAINEN; HARWOOD, 1994) . . . . . p. 61

29 Calculo do MB-LBP para uma vizinhança 3×3. . . . . p. 62

30 Passos da extração de um descritor HOG. . . . . p. 65

31 Ilustração das etapas de extração do HOG: (a) imagem de entrada; (b) cálculo do orientação e magnitude do gradiente; (c) agrupamento de *pixels* em células; (d) cálculo do histograma de cada célula; (e) agrupamento de células em blocos sobrepostos e normalização; (f) obtenção do descritor através da concatenação dos histogramas da etapa (e). . . . . p. 65

32 Cálculo do gradiente. . . . . p. 66

33 Votação ponderada. . . . . p. 67

34 Disposição de blocos e células em uma imagem. . . . . p. 68

35 Diferença entre o LDA e o PCA. . . . . p. 70

36 Etapas do cálculo da LDA. . . . . p. 70

37 Comparativo entre um problema linear e não linear para um conjunto de dados bidimensionais. . . . . p. 75

38 Cálculo da distância entre os hiperplanos H_1 e H_2. . . . . p. 76

39 Validação cruzada *k-fold*, com k igual a 5. . . . . p. 80

40 Arquitetura básica do Linux embarcado. FONTE: (PRADO, 2016) . . . . p. 84

41 BeagleBone Black. . . . . p. 86

42 Imagem contendo objetos e resultado de um sistema de detecção, para um exemplo fictício. . . . . p. 91

43 Virtualizador VirtualBox. . . . . p. 95

44 Ambiente de desenvolvimento do *Qt Creator*. . . . . p. 96

45 Fluxograma do processo de segmentação. . . . . p. 98

46 Resultados da detecção. (a) Imagem Original; (b) Segmentação por cor; (c) Regiões de Interesse; (d) Localização da placa detectada. . . . . p. 102

47 Fluxograma do processo de segmentação. . . . . p. 103

48 Fluxograma do processo de segmentação. . . . . p. 104

49 HOG (Histogram of Oriented Gradients) da placa detectada. . . . . p. 105

50 Cálculo da dimensão do vetor HOG, com os parâmetros utilizados. . . . . p. 106

51 Diferença compilação nativa e cruzada. . . . . p. 108

52 Ferramentas presentes na *toolchain arm-linux-gnueabihf*. . . . . p. 110

53 Comunicações feitas pelo *Kernel*. . . . . p. 110

54 Esquema usual para o trabalho com sistemas embarcados. . . . . p. 111

55 Placas de trânsito presentes na base de dados GTSRB e seus respectivos grupos. FONTE: (STALLKAMP et al., 2012) . . . . p. 116

56 Exemplos de placas de trânsito, presentes na GTSRB, nas mais diversas condições. FONTE: (STALLKAMP et al., 2012) . . . . p. 117

57 Grupos de placas, agrupadas por forma e cor, presentes na GTSDB. FONTE: (HOUBEN et al., 2013) . . . . p. 118

58 Exemplo de cenas naturais presentes no conjunto de teste da GTSDB. FONTE: (HOUBEN et al., 2013) . . . . . p. 119

59 Gráfico comparativo do desempenho da detecção. (a) sem o processo de segmentação; (b) com processo de segmentação. . . . . p. 121

60 Tempo médio (ms) de execução da detecção no Desktop e na BeagleBone Black (Resolução de 640×480 *pixels*). . . . . p. 124

61 Matriz de confusão da fase de reconhecimento. . . . . p. 125

62 Matriz de confusão da fase de reconhecimento. . . . . p. 127

63 Tempo médio (ms) de execução do reconhecimento na BeagleBone Black. p. 127

72 Proibitório - Vídeo 1 . . . . . p. 144

73 Proibitório - Vídeo 2 . . . . . p. 144

74 Proibitório - Vídeo 3 . . . . . p. 144

75 Proibitório - Vídeo 4 . . . . . p. 144

76 Mandatório - Vídeo 1 . . . . . p. 145

77 Mandatório - Vídeo 2 . . . . . p. 145

78 Mandatório - Vídeo 3 . . . . . p. 145

79 Mandatório - Vídeo 4 . . . . . p. 145

80 Perigo - Vídeo 1 . . . . . p. 146

81 Perigo - Vídeo 2 . . . . . p. 146

82 Perigo - Vídeo 3 . . . . . p. 146

83 Perigo - Vídeo 4 . . . . . p. 146

Lista de Tabelas

1 Comparativo Câmera Fotográfica e olho humano. . . . . p. 25

2 Mapeamento (*lookup table*) de uma intensidade r_k para s_k . . . . . p. 36

3 Várias placas de desenvolvimento e seus respectivos preços. FONTE: (SRIRAM; ILLURI, 2014) . . . . . p. 86

4 Resumo das funcionalidades da BeagleBone Black. FONTE:(HE; HUANG; WOLTMAN, 2014) . . . . . p. 87

5 Tabela de contingência . . . . . p. 89

6 Matriz Confusa para um exemplo fictício de reconhecimento de dígitos. . p. 90

7 Valores de *Threshold* para as cores vermelho e azul . . . . . p. 100

8 Comparativo de taxa de detecção para 10% de precisão. Fonte: (HOUBEN et al., 2013) . . . . . p. 122

9 Resultados do teste em video da fase de detecção realizado para as 3 classes de placas de trânsito . . . . . p. 122

10 Comparativo da metodologia de reconhecimento proposta. Fonte: (STALLKAMP et al., 2012) . . . . . p. 126

Glossário

Adaboost	*Adaptative Boosting*
ABI	*Application Binary Interface*
ADAS	*Advanced Driver Assitance System*
ANN	*Artificial Neural Networks*
ASIC	*Application-specific Integrated Circuit*
BBB	*BeagleBone Black*
BSP	*Board Support Package*
CDF	*Cumulative Distribution Functio*
CLAHE	*Contrast Limited Adaptive Histogram Equalization*
CMOS	*Complementary Metal Oxide Semiconductor*
CNN	*Convolutional Neural Networs*
DT	*Distance Transform*
DTB	*Distance to Borders*
EABI	*Embedded-Application Binary Interface*
FN	*False Negative*
FP	*False Positive*
FPGA	*Field Programmable Gate Arrays*
GCC	*GNU Compiler Collection*
GPIO	*General Purpose Input/Output*
GPL	*General Public License*
GPU	*Graphic Processing Units*
GUI	*Graphical User Interface*
HOG	*Histogram of Oriented Gradients*
HSI	*Hue, Saturation, Intensity*
HSV	*Hue, Saturation, Value*
IDE	*Integrated Development Environment*
LBP	*Local Binary Pattern*
LDA	*Linear Discriminant Analysis*
MB-LBP	*Multiblock Normalization Local Binary Pattern*
NFS	*Network File System*

OpenCV	*Open Source Computer Vision*
PCA	*Principal Component Analysis*
PDF	*Probability Density Function*
RGB	*Red, Green, Blue*
SDK	*Sofware Development Kit*
SBC	*Single Board Computers*
SFC	*Split-Flow Cascade*
SIFT	*Scale-invariant feature transform*
SVM	*Support Vector Machine*
TM-LBP	*Tilted MN-LBP*
TN	*True Negative*
TP	*True Positive*
TSD	*Traffic Sign Detection*
TSDR	*Traffic Sign Detection and Recognition*

1 Introdução

O número de automóveis tem aumentado substancialmente a cada ano e uma das consequências deste fato é o aumento constante de acidentes de trânsito. Além disso, o excesso de informações nas rodovias e cidades, tais como celular, trafego intenso, poluição visual e sonora, consomem a atenção dos motoristas de modo a reduzi-la na condução do veículo e nos perigos existentes nas vias (KLAUER et al., 2014; STRAYER; DREWS; CROUCH, 2006).

Com o intuito de ajudar os motoristas a reagirem as mais diversas condições da rodovia e melhorar sua segurança, surgiram as soluções conhecidas como *Advanced Driver Assitance System* (ADAS). Esses sistemas auxiliam o motorista no processo de condução do veículo e são projetados de forma a aumentar a segurança do carro e consequentemente a segurança nas vias. As tecnologias ADAS podem ser separados com relação as suas funcionalidades.

As funcionalidades de segurança têm como objetivo evitar colisões e acidentes, fornecendo maneiras de alertar o motorista para potenciais problemas ou evitar colisões através da implementação de proteções e tomada de controle do veículo. Outro tipo de funcionalidade são as adaptativas que podem ser responsáveis por ligar as luzes automaticamente, prover controle de velocidade, frenagem automática, alertar o motorista de outros carros ou acidentes, entre outras. Dentre essas tecnologias ADAS pode-se destacar as seguintes: estacionamento automático, sistema de comunicação veicular, detecção de ponto cego, sistema anti-colisão, detecção de pedestres entre outros (SHAOUT; COLELLA; AWAD, 2011).

Um dos componentes chave do ADAS é a detecção e reconhecimento automático de placas de trânsito ou *Traffic Sign Detection and Recognition* (TSDR), que permite o automóvel reconhecer as placas de trânsito em ambientes do mundo real. Esses sistemas são considerados um problema desafiador dentro da área de visão computacional com alta relevância pratica no mundo real, cujo tópico tem sido objeto de pesquisa por várias

décadas. A junção das etapas de detecção e reconhecimento é conhecida como identificação e é um problema de classificação de múltiplas categorias, possuindo uma frequência desbalanceada de classes

As placas de trânsito têm um papel fundamental na direção e segurança dos motoristas. São elas responsáveis pela navegação e que ditam as regras de circulação, determinam as leis de trânsito e indicam perigo ou a necessidade de uma maior atenção. Assim, uma detecção e reconhecimento bem-sucedidos permite alertar, com antecedência, o motorista de perigos, orienta-lo na navegação ou mesmo indicar a necessidade de maior atenção em determinado trecho da rodovia.

A Figura 1 ilustra um sistema completo para identificação de sinalização de trânsito. Esse é montado no para-brisa do automóvel, sendo composto por uma câmera ligada a um hardware embarcado.

A câmera é responsável pela a aquisição das imagens, que são processadas pelo embacado, que possui um software de visão computacional para identificação de placas de trânsito. Assim que identificada nessa imagem uma placa, o sistema pode atuar através de alertas, que podem ser sinais sonoros ou visuais, ou mesmo interferir na direção, reduzindo a velocidade ou parando o carro. No entanto, neste trabalho não serão abordados o processo de aquisição e atuação presentes em um sistema completo.

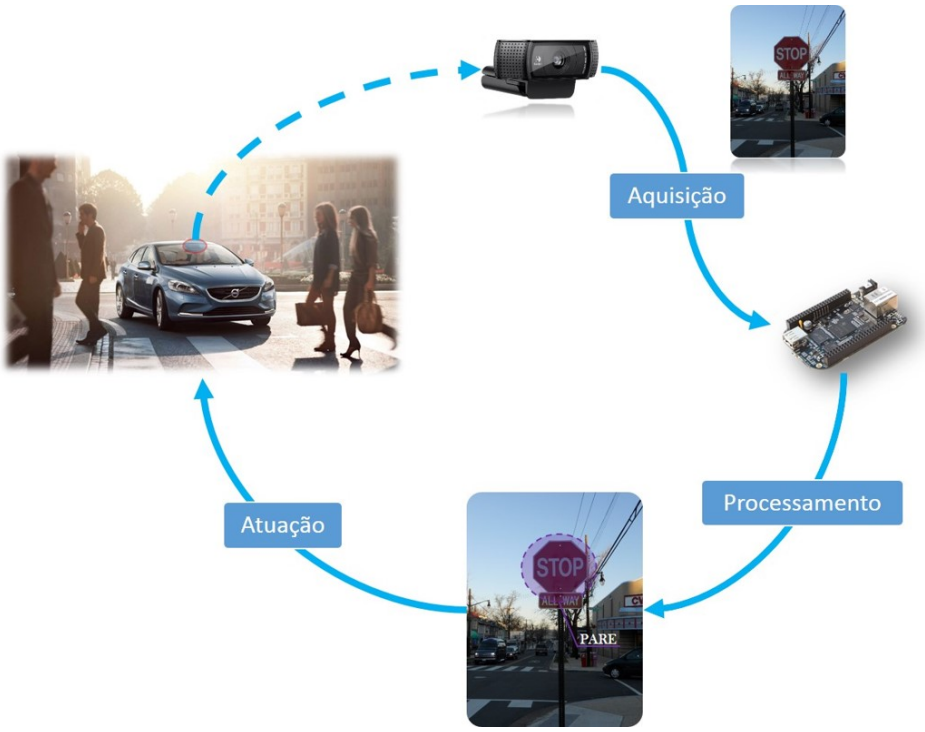


Figura 1: Sistema completo para Identificação de Placas de Trânsito.

As características e significados das placas de trânsito são padronizados por diferentes órgãos regulatórios, sendo projetadas para se destacarem nos ambientes. Além disso, os sinais são rigidamente posicionados e colocados de forma a permitir uma clara visualização pelos motoristas.

Esses fatores reduzem uma parcela da dificuldade em se projetar sistemas de identificação de placas de trânsito. No entanto, o principal desafio para um bom desempenho e robustez de um TSDR é advindo da complexidade dos ambientes.

Entre as diversas dificuldades, as condições climáticas e de luminosidade são um dos problemas corriqueiros envolvidos na analise e que podem variar consideravelmente nos ambientes de trânsito. Adicionalmente, existe a movimentação da câmera, que pode frequentemente gerar *motion blur* e mudanças abruptas no contraste. A forma com que os sinais de trânsito são instalados e o material utilizado para construção das placas podem mudar com o tempo, devido as condições climáticas, acidentes ou mesmo vandalismo, resultando na mudança de posição e degradação da pintura. Não bastasse estes problemas, ainda é necessário lidar com oclusões devido a outros objetos tal como arvores.

Recentemente, sistemas para detecção tem sido incluída em automóveis topo de linha de várias montadoras. No entanto, a maioria das soluções TSDR fornecidas pela indústria, como a de carros fabricados pela Volvo, BMW e Mercedes, além de caras estão limitadas a identificar somente sinalizações de velocidade (HAN et al., 2015). Outra desvantagem destes sistemas é que estão disponíveis apenas para carros novos, e muita das vezes são vendidos em pacotes separados.

Infelizmente ainda existem poucos sistemas acessíveis e baratos para o público geral. Dessa maneira, faz-se necessário a popularização de tais sistemas entre motoristas já que são de custo elevado e limitados a veículos novos. A sua adoção em massa pode aumentar substancialmente a segurança de quem está dentro e fora do veículo , reduzindo acidentes e mortes em decorrência da falta de atenção dos motoristas.

1.1 Objetivos

O objetivo geral deste trabalho é o desenvolvimento de um sistema de visão computacional embarcado, que permita a detecção e reconhecimento de placas de trânsito. Este sistema deverá estar em conformidade com as restrições computacionais presentes no sistema embarcado em que será integrado. O hardware utilizado será de baixo custo, compacto, eficiente energeticamente e acessível. Além disso, as ferramentas e biblioteca

para o desenvolvimento da solução devem ser de código aberto e gratuitas.

1.2 Revisão Bibliográfica

1.2.1 Soluções de Hardware

Recentemente, uma grande variedade de soluções de hardware para TSDR em tempo real foram propostos. Entre elas estão a utilização de computadores de propósito geral, FPGAs (*Field Programmable Gate Arrays*) (HAN et al., 2015; AGUIRRE-DOBERNACK; GUZMAN-MIRANDA; AGUIRRE, 2013; MÜLLER et al., 2010; SALLAH; HUSSIN; YUSOFF, 2011), ASIC (*Application-specific Integrated Circuit*) (PARK et al., 2012), GPUs (*Graphic Processing Units*)(CHEN et al., 2014; DING; YOON; LEE, 2012; GLAVTCHEV et al., 2011) e SBCs (*Single Board Computers*) (KUMARASWAMY et al., 2011).

Por ser um hardware de fácil acesso, os computadores de propósito geral são a plataforma mais usada no desenvolvimento de sistemas TSDR. A estes podem ser adicionadas GPUs, as quais se popularizaram recentemente. Elas aumentam a capacidade de paralelismo, uma característica imprescindível para a aceleração de determinados algoritmos. Apesar disso, estas soluções são pouco compactas e possuem baixa eficiência energética, dificultando a sua integração em sistemas automotivos.

Dentre as soluções citadas, as ASIC são as mais caras e mais complexas. Elas podem atingir altas velocidades, porém são inflexíveis a modificações após a produção dos chips. Dessa maneira poucos trabalhos focam neste tipo de solução.

Com o intuito de redução do tamanho, aumento da velocidade de execução e corrigir as limitações dos ASICs, vários trabalhos propõem a utilização de FPGAs. Elas permitem a implementações em hardware de algoritmos de visão computacional, além de possuírem uma boa eficiência energética. Em Müller et al. (2010), um sistema de detecção de placas de sinalização foi implementado em uma Xilinx Zynq 7020 SoC (*System on Chip*), um hardware com suporte a FPGA.

Devido a capacidade de paralelismo destes sistemas, um múltiplo escaneamento de janelas foi realizado em cada quadro. De acordo com os testes realizados o sistema trabalhou a 83 *frames* por segundo em uma resolução *Full HD* (1920 por 1080 *pixels*). Apesar de permitir taxas maiores de quadros, esta solução de hardware possui uma complexidade intermediaria de desenvolvimento e seu preço ainda é alto se comparado com a solução proposta nesta dissertação.

Resolvendo os problemas de eficiência energética, tamanho e preço, os SBCs surgem com uma solução plausível. Com a popularização dos celulares e surgimentos de projetos com fins educacionais seus preços caíram drasticamente. Não possuem o mesmo poder computacional de outras soluções, porém tem o suficiente para a implementação descrita neste trabalho. Além disso, a execução de sistemas operacionais viabiliza o uso de bibliotecas disponíveis gratuitamente, o que acelera o tempo desenvolvimento de sistemas embarcados.

Kumaraswamy et al. (2011) utiliza uma SBC como solução de hardware. O sistema proposto é baseado em informações de cor e forma, DtBs (*Distance to Borders*), que são classificados através de uma hierarquia de SVMs (*Support Vector Machine*). As informações de cores são utilizadas no processo de segmentação. Já as características DtBs juntamente com a SVM são responsáveis pela classificação do formato da placa de sinalização e do pictograma interno de cada placa de trânsito. Os testes foram conduzidos em vídeos capturados através da apresentação de placas de sinalização impressas em diversas distâncias e orientações. O sistema é capaz de identificar 17 tipos de sinalização e atingiu uma capacidade de processamento de 5 quadros por segundo.

Poucos destes trabalhos se atentam para o equilíbrio entre eficiência e custo (KUMARASWAMY et al., 2011). Além disso, nem todos focam no problema completo (PARK et al., 2012; CIRESAN et al., 2011) ou utilizam bases públicas para os testes (CIRESAN et al., 2011; CHEN et al., 2014), sendo difícil replicar o desempenho reivindicado. Outra limitação é o número de classes reconhecidas (AGUIRRE-DOBERNACK; GUZMAN-MIRANDA; AGUIRRE, 2013; MÜLLER et al., 2010; SALLAH; HUSSIN; YUSOFF, 2011; DING; YOON; LEE, 2012; GLAVTCHEV et al., 2011; KUMARASWAMY et al., 2011). Alguns destes sistemas são criados para tipos específicos de placas de trânsito, como por exemplo placas de velocidade, o que impede sua aplicação generalizada (MÜLLER et al., 2010; GLAVTCHEV et al., 2011).

1.2.2 Técnicas de Visão Computacional

Não existe uma definição universal ou exata sobre o que constitui uma característica extraída de uma região da imagem, a definição geralmente depende do problema ou do tipo de aplicação. Em visão computacional o conceito de detecção de características refere-se ao método cujo objetivo é criar abstrações a respeito das informações de uma imagem e tomar decisões em cada ponto se houver uma característica de terminado tipo ou não. Com tais técnicas é possível se reduzir os pontos a serem analisados, permitindo, assim,

uma maior eficiência computacional.

No problema abordado os objetos de interesse, no caso placas de trânsito, são projetadas com cores e formas discriminantes. Essas características podem ser exploradas no processo de análise. Devido à complexidade do problema, Mogelmose, Trivedi e Moeslund (2012) propõe a divisão do problema em etapas de detecção, reconhecimento e rastreamento. A forma com que estas etapas se relacionam podem ser visualizadas na Figura 2.

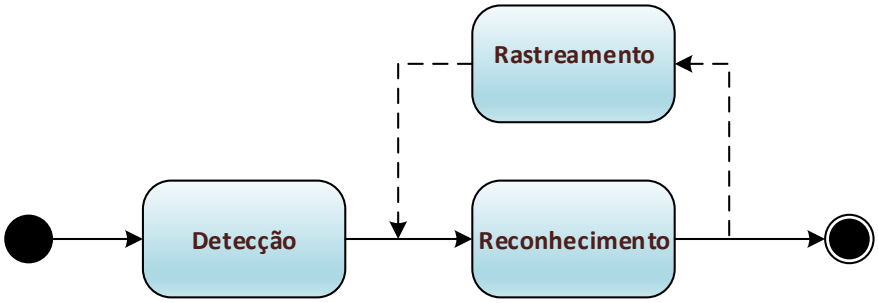


Figura 2: Etapas de um sistema de reconhecimento de placa de trânsito.

Opcionalmente pode-se adicionar uma etapa de segmentação para a redução do espaço de busca necessário na fase de detecção e localização. Devido às restrições computacionais de um sistema embarcado neste trabalho esta etapa não foi ignorada e como poderá ser visto a frente teve grande impacto nos resultados finais.

A segmentação pode ser realizada com imagens coloridas ou em escala de cinza. Quando realizada em imagens coloridas é importante a escolha de um espaço de cores que possua robustez as variações de iluminação presentes em ambientes reais. Isto é possível através de espaço de cores que separam a componente cromática e intensidade, tais como HSI (*Hue, Saturation, Intesity*) (MALDONADO-BASCON et al., 2007; LAFUENTE-ARROYO et al., 2010) e HSV (*Hue, Saturation, Value*) (REN et al., 2009; YAKIMOV; FURSOV, 2015). No entanto ainda assim existem autores que preferem a utilização do espaço de cores RGB (*Red, Green, Blue*) (OHGUSHI; HAMADA, 2009; BROGGI et al., 2007) pois não há a necessidade de conversão, já que grande parte dos sensores CMOS (*Complementary Metal Oxide Semiconductor*) se baseiam neste formato. Gomez-Moreno et al. (2010) faz um estudo detalhado sobre os principais espaços de cores utilizados no domínio de TSD (*Traffic Sign Detection*).

Uma maneira de realizar a segmentação de imagens em escala de cinza é através

da técnica MSER (Maximally stable extremal regions) (GREENHALGH; MIRMEHDI, 2012; WAHYONO et al., 2014). Esta técnica consiste em se realizar diversas segmentações, variando-se o valor de limiar e filtrar regiões que não mantêm suas estruturas. Caso a imagem seja colorida é necessário a conversão em escala de cinza. Em Greenhalgh e Mirmehdi (2012) e Wahyono et al. (2014) foi utilizado uma normalização vermelho/azul para a conversão e destaque das cores presentes nas placas de trânsito.

As regiões resultantes são filtradas, através de restrições dimensionais, e classificadas na fase de detecção e localização. A escolha da técnica de aprendizagem de máquina e característica a ser extraída é de suma importância. Dentre as técnicas de aprendizagem comumente utilizadas estão: SVMs (*Suport Vector Machine*) (GREENHALGH; MIRMEHDI, 2012; MALDONADO-BASCON et al., 2007; TIAN et al., 2016), ANN (*Artificial Neural Networks*) (ZAKLOUTA; STANCIULESCU, 2012; G; PATIL, 2016), *k-nearst neighbor* (OHGUSHI; HAMADA, 2009; REN et al., 2009) e métodos *essemble* como o AdaBoost (CHEN; LU, 2016). Com relação a características são usadas HOG (*Histogram of Oriented Gradients*) (GREENHALGH; MIRMEHDI, 2012; MALDONADO-BASCON et al., 2007; KASSANI; HYUN; KIM, 2016), DtB (*Distance to Borders*) (WAHYONO et al., 2014; MALDONADO-BASCON et al., 2007), SIFT (*Scale-invariant feature transform*) (OHGUSHI; HAMADA, 2009; REN et al., 2009), entre outras.

Outros trabalhos preferem não utilizar técnicas de segmentação, pois a consideram pouco robusta a variação da iluminação. Eles se baseiam somente na informação de forma e técnicas de janelas deslizantes. Uma janela de tamanho fixo percorre a imagem, de forma sobreposta e em várias escalas, classificando cada região como contendo ou não o objeto de interesse. As regiões classificadas como contendo o objeto são agrupadas e delimitadas por uma caixa.

Seguindo esta abordagem, Liu, Chang e Chen (2014) propõe modificações, para a detecção de múltiplas classes, no já conhecido *framework* Viola e Jones (2004). No trabalho são introduzidos novos descritores, MN-LBP (*Multiblock Normalization Local Binary Pattern*) e TMN-LPB (*Tilted MN-LBP*), adequados para o problema de detecção de placas de trânsito. Além disso uma árvore SFC (*Split-Flow Cascade*) é criada, através do algoritmo *CF.Adaboost*, para a avaliação de múltiplas classes.

Posterior a detecção tem-se a fase de reconhecimento (WAHYONO; JO, 2014) na qual é atribuído uma classe de placa de trânsito a cada uma das regiões de interesse. Assim como a fase de detecção a escolha da técnica de aprendizagem e características impacta diretamente no resultado final. Ohgushi e Hamada (2009) extrai descritores locais SIFT das

regiões de interesse e utiliza a técnica Bag of Features para classifica-las. Já Greenhalgh e Mirmehdi (2012) e Wahyono et al. (2014) utilizam descritores HOG e os classificam através de SVMs lineares. Em seu trabalho, Zaklouta e Stanciulescu (2012) avaliou o desempenho de estruturas *KD-tree* e *Random Forest* com características HOG e DT (*Distance Transform*).

Contrariando o usual, Maldonado-Bascon et al. (2007) alimenta uma SVMs, com *kernel* gaussiano, com os próprios *pixels* da imagem. Não é comum a utilização de toda as informações de *pixels* como descritores, pois além de consumirem muitos recursos, não apresentam resultados satisfatórios. No entanto, recentemente, trabalhos baseados em aprendizagem profunda (SERMANET; LECUN, 2011; CIRESAN et al., 2011; JUNG et al., 2016), utilizando dados crus como entrada, tem presentados resultados expressivos.

Esses trabalhos utilizam uma arquitetura de ANN com múltiplas camadas, que utilizam operações de convolução como base de todo o processo. As chamadas Convnets ou simplesmente CNN(*Convolutional Neural Networs*), através do treinamento aprendem filtros que são responsáveis pela extração de característica, diferentemente dos métodos tradicionais onde se utiliza extratores feitos a mão. Esses trabalhos apresentam resultados estado da arte, acima de 99%, superando a performance do ser humano. No entanto os recursos computacionais necessários não estão disponíveis para sistemas embarcados, devido suas restrições.

Por fim uma etapa de rastreamento é responsável por calcular, no quadro seguinte, a provável posição do objeto detectado, liberando assim esforço computacional exigido pela fase de detecção.

Neste trabalho foram abordadas somente as fases de detecção e reconhecimento. Ambas fases possuem, para a metodologia escolhida, uma etapa de treinamento, em que a aprendizagem supervisionada é realizada e uma etapa de teste, em que o desempenho do sistema é medido.

As técnicas para cada uma dessas etapas forma selecionadas levando-se em consideração as restrições do hardware embarcado, uma vez que, técnicas com resultados estado da arte, como as CNN, necessitam de um poder computacional expressivo. Cada fase da solução encontrada foi testada em bases desafiadoras e delas foram extraídos resultados de desempenho e tempo de execução. Além disso, foram realizados testes em vídeo para a fase de detecção. A metodologia proposta foi, então, implementada e integrada a uma SBC de fácil acesso, baixo custo e consumo energético.

1.3 Organização do Trabalho

A fim de explanar o sistema de visão embarcado desenvolvido, este trabalho está dividido da seguinte maneira:

- **Capitulo 2:** trata das técnicas utilizadas na metodologia proposta e ferramentas empregadas no desenvolvimento, assim como conceitos utilizados tanto na integração do sistema de visão computacional ao embarcado quanto para avaliação do desempenho desse sistema.
- **Capitulo 3:** apresenta a estruturação do ambiente transparente de desenvolvimento para se trabalhar com sistemas embarcados e as etapas da metodologia implementada, que são segmentação, detecção e reconhecimento.
- **Capitulo 4:** são exibidas as bases de testes utilizadas para a detecção e reconhecimento. Além disso, através dessas bases é aferido o desempenho da metodologia, proposta tanto para o sistema embarcado quanto desktop, e os resultados obtidos são analisados.
- **Capitulo 5:** conclui o trabalho, resumindo o que foi realizado no trabalho e os resultados obtidos. Também são deixas sugestões de melhorias do sistema de visão computacional embarcado e novos testes que podem ser realizados.

2 Fundamentação Teórica

2.1 Imagem Digital

A visão representa um dos sentidos mais importantes para os seres humanos. Através dela é possível localizar-se no espaço, identificar objetos, reconhecer faces, entre diversas outras tarefas realizadas rotineiramente (JAHNE, 2000). A percepção visual humana depende de uma iluminação apropriada e está limitada a uma pequena faixa do espectro de radiação eletromagnética, conhecida com luz.

O olho humano possui um formato quase esférico com diâmetro médio variando de 2 a 2,5 cm (GONZALEZ; WOODS, 2008). A luz dos objetos do mundo real adentra o olho humano através de uma abertura frontal na íris denominada pupila, atravessando uma lente conhecida como cristalino e finalmente atingindo a retina, que forma a parte posterior do olho. Dessa forma caso haja uma focalização adequada da cena, uma imagem invertida é formada nessa região.

A retina possui dois tipos de fotorreceptores, cones e bastonetes, que são responsáveis pela conversão de energia luminosa em impulsos elétricos, que serão posteriormente interpretados pelo cérebro. Os cones estão localizados principalmente na área central da retina, chamada de fóvea, e são altamente sensíveis a cor, operando apenas em condições suficientes de luminosidade. O número de cones em cada olho varia de 6 a 7 milhões (GONZALEZ; WOODS, 2008). Cada uma desses se conecta à própria terminação nervosa, o que permite aos humanos perceberem pequenos detalhes em uma cena.

Presentes em maior quantidade do que os cones, 75 a 150 milhões (GONZALEZ; WOODS, 2008), os bastonetes estão distribuídos ao longo de toda a retina. Por cobrirem uma área maior e estarem conectados a somente uma terminação nervosa o nível de detalhe captado é menor quando comparado aos cones. Além disso são insensíveis a cores e operam em condições de baixa luminosidade. Dessa forma, o intuito dessas células é apresentar uma visão geral do campo de visão humano.

Com a evolução da humanidade veio a descoberta da fotografia, considerada um grande passo em direção a evolução do registro visual. Inicialmente o registro era realizado de forma analógica, por meio de filmes fotográficos. Com o advento dos computadores e criação das câmeras digitais, grandes avanços foram feitos na forma com que as imagens são armazenadas e manipuladas. Isso foi possível pelo surgimento de técnicas de processamento de imagens digitais, que além de versátil é considerado o método de processamento de imagem mais barato.

Uma câmera fotográfica é uma câmara escura construída para conversão de energia luminosa em sinais elétricos, produzindo uma imagem adequada para um determinado tipo de propósito. Além disso, ela pode ser vista como uma extensão do olho humano. Cada um de seus componentes pode ser associada a uma parte do olho e sua função correspondente. Na Tabela 1 são apresentados algumas dessas analogias.

Tabela 1: Comparativo Câmera Fotográfica e olho humano.

Câmera Fotográfica	**Olho Humano**	**Função**
Obturador	Pálpebra	Permitir ou impedir a entrada de luz.
Diafragma	Iris	Controla a quantidade de luz que entra pela lente.
Lente	Conjunto formado por: Cristalino, córnea (em menor grau), humor aquoso e humor vítreo.	Focalizar a luz, melhorando a nitidez das imagens formadas no plano focal.
Câmara escura	Coroide	Absorver a luz, evitando sua reflexão.
Sensor	Retina	Converter energia luminosa em impulsos elétricos

O processo de aquisição de uma imagem pode ser tanto analógico quanto digital. Devido ao fato de computadores somente manipularem dados digitais existe a necessidade de converter os sinais elétricos analógicos, capturados pelo sensor, em digitais. Dois processos estão envolvidos nesta conversão, conhecida também como discretização. São eles: amostragem e quantização.

O processo de amostragem corresponde a discretização das coordenadas espaciais, en-

quanto que a quantização é a discretização dos valores de brilho (SOLOMON; BRECKON, 2011). Geralmente ambos processos são uniformes implicando na amostragem de uma imagem em pontos igualmente espaçados, distribuídos em uma matriz $M \times N$. Cada elemento dessa matriz corresponde a uma aproximação do nível de cinza, no ponto amostrado, para um conjunto de valores $\{0, 1, \ldots, L-1\}$, onde L corresponde ao valor de brilho, usualmente associado a potência de 2, Equação 2.1.

$$L = 2^k \tag{2.1}$$

Ambos processos acarretam na supressão de informação de uma imagem analógica, sendo que a qualidade de uma imagem digital está intrinsicamente ligado aos valores de M,N e L. Dessa maneira, o número de bits de uma imagem monocromática de dimensões $M \times N$ é dado pela Equação 2.2 .

$$b = M * N * k \tag{2.2}$$

O mesmo procedimento se aplica a imagens coloridas, amostrando de forma idêntica os três planos de uma imagem RGB (*Red, Green, blue*). Cada plano é quantizado em L níveis de brilho, sendo necessário um total de $l_r + l_g + l_b$ *bits*. Utilizando-se 24 *bits*, 8 *bits* para canal é possível representar um total de 16 milhões de cores distintas.

Uma imagem monocromática é uma função continua bidimensional representada por $f(x, y)$, onde x e y são coordenadas espaciais e o valor dessa função em qualquer conjunto de coordenadas é proporcional à intensidade luminosa nesse ponto (QUEIROZ; GOMES, 2006). Ao ser capturada por uma câmera digital e sofrer o processo de discretização, tal imagem, referenciada como imagem digital, é representada em um computador através de arranjos bidimensionais de pontos, conhecidos como *arrays*.

Cada elemento deste *array* é denominado elemento de imagem ou *pixel*. A notação matricial usual para localização de um *pixel*, em uma imagem bidimensional, é apresentada na Figura 3. O primeiro e segundo índice, para uma imagem com M linhas e N colunas, correspondem respectivamente a linha n e a coluna m onde o pixel se encontra. A convenção usualmente utilizada na representação espacial e sentido de leitura de uma imagem digital é da esquerda para a direita.

Nesse modelo de imagem digital, a intensidade luminosa no ponto (x, y) pode ser dividida em 2 componentes: iluminação e reflectância. A componente de iluminação,

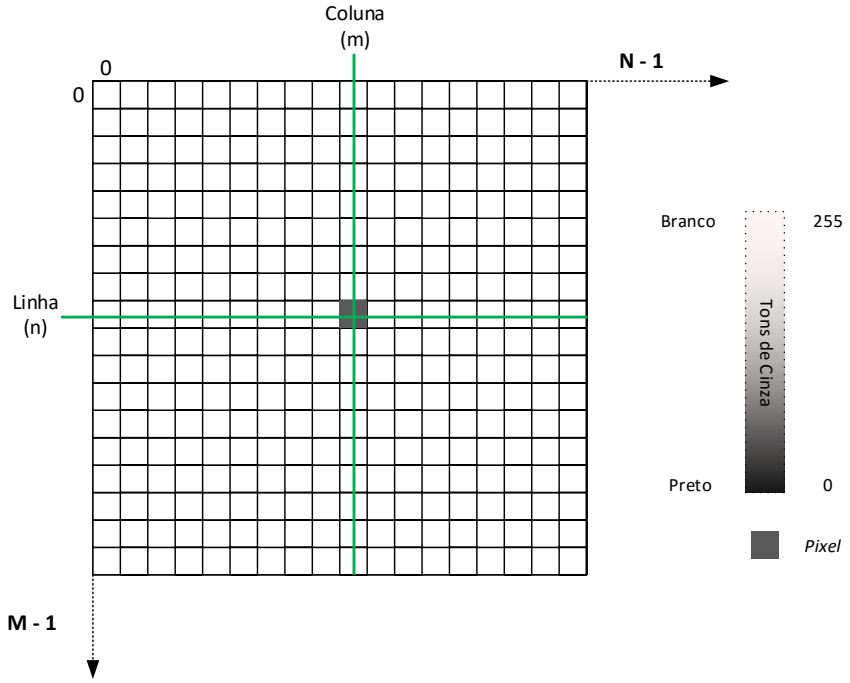


Figura 3: Representação matricial de uma imagem.

$i(x, y)$ está associada a quantidade de luz que incide no ponto (x, y), dependendo das características da fonte de iluminação. Já a componente de reflectância, $r(x, y)$ está associada a quantidade de luz refletida no ponto (x, y), dependendo das características da superfície do objeto. A Equação 2.3 indica que a reflectância está limitada entre 0 (absorção total) e 1 (reflectância total). O produto de ambas resulta em $f(x, y) = i(x, y) \cdot r(x, y)$, onde:

$$\begin{aligned} 0 < i(x, y) < \infty \\ 0 < r(x, y) < 1 \end{aligned} \tag{2.3}$$

Uma imagem digital pode ser do tipo binaria, monocromática ou colorida. A representação binária possui somente dois valores de intensidade para cada *pixel*, 0 ou 1. Normalmente tal tipo de imagem é utilizado para ressaltar algum tipo de característica ou como mascara em algum processo de segmentação.

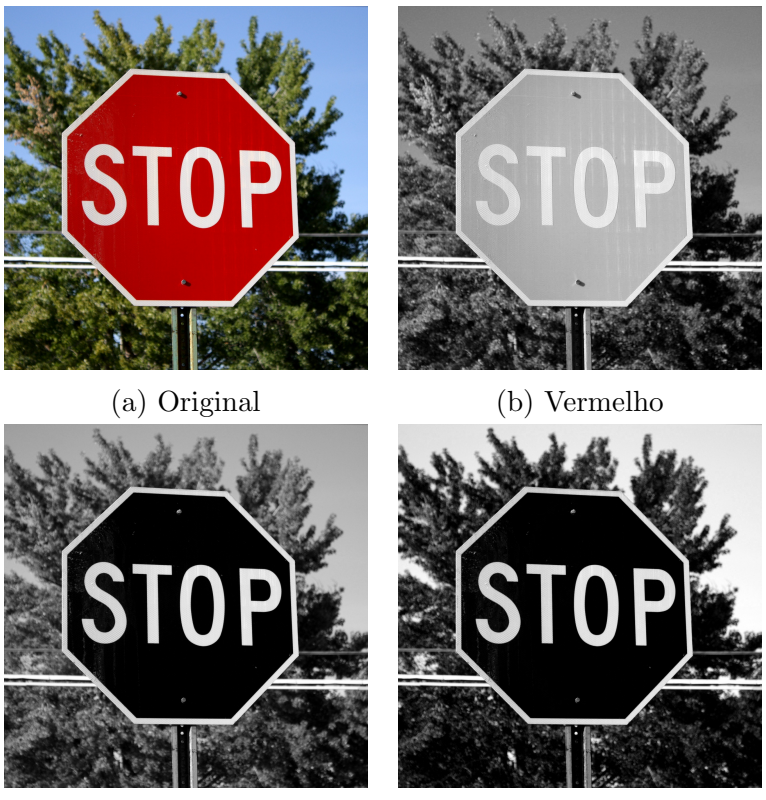
Como visto anteriormente, imagens monocromáticas ou escala de cinza possuem valores de *pixels* em uma escala com valores de intensidade variando de 0 a $L-1$. Geralmente são representados com 8 *bits*, permitindo a representação de 256 valores.

Estendendo o tipo de imagem monocromática, uma imagem colorida é composta por

3 canais cores, representado geralmente por 24 *bits* por *pixel*, sendo 8 *bits* por canal. A ela está atrelado um espaço de cores , que corresponde a uma representação de como armazenar as cores, especificando o número e a natureza dos canais de cores (SOLOMON; BRECKON, 2011).

A soma das cores espectrais torna os objetos que emitem luz visíveis perceptíveis. De acordo com Queiroz e Gomes (2006) um processo aditivo pode ser entendido como a combinação proporcional das componentes monocromáticas nas faixas do espectro associadas as cores verde, vermelho e azul, as quais formam as demais cores percebidas pelo olho humano.

O olho humano possui 3 tipos de receptores de cores, os quais possuem uma absorção limitada e não uniforme para cada um desses comprimentos de ondas. Esse processo de geração culminou na criação do espaço de cores conhecido como RGB, que é baseado numa porção do espectro magnético visível aos seres humanos. O RGB é um espaço de cores comumente utilizado para representação de imagens digitais, uma vez que corresponde aos três tipos de cores que são misturadas para a visualização em um monitor ou dispositivos similares. A Figura 4 apresenta cada um dos três canais monocromáticos para uma determinada imagem.



(a) Original (b) Vermelho

(c) Verde (d) Azul

Figura 4: Canais para um imagem no espaço de cores RGB. FONTE: (PHOTOS..., 2016)

O espaço de cores RGB pode ser representado por um cubo, representado na Figura 5. Os valores de cada um dos eixos R (*red*), G (*green*) e B (*blue*) estão normalizados de tal maneira que seus valores variam de 0 a 1. A cor preta, correspondendo a ausência de todas três cores, está situada na origem do cubo (coordenadas $(0, 0, 0)$) enquanto que a cor branca está localiza no canto oposto (coordenadas $(1, 1, 1,)$), indicando a presença máxima de todas as três cores.

Um dos problemas primários perceptíveis desse espaço de cores é sua não linearidade. Isso quer dizer que se movendo em uma determinada direção neste cubo não necessariamente produz uma cor que é consistente com a mudança em cada um dos canais. Por exemplo, começando no branco e subtraindo a componente azul produz a cor amarela.

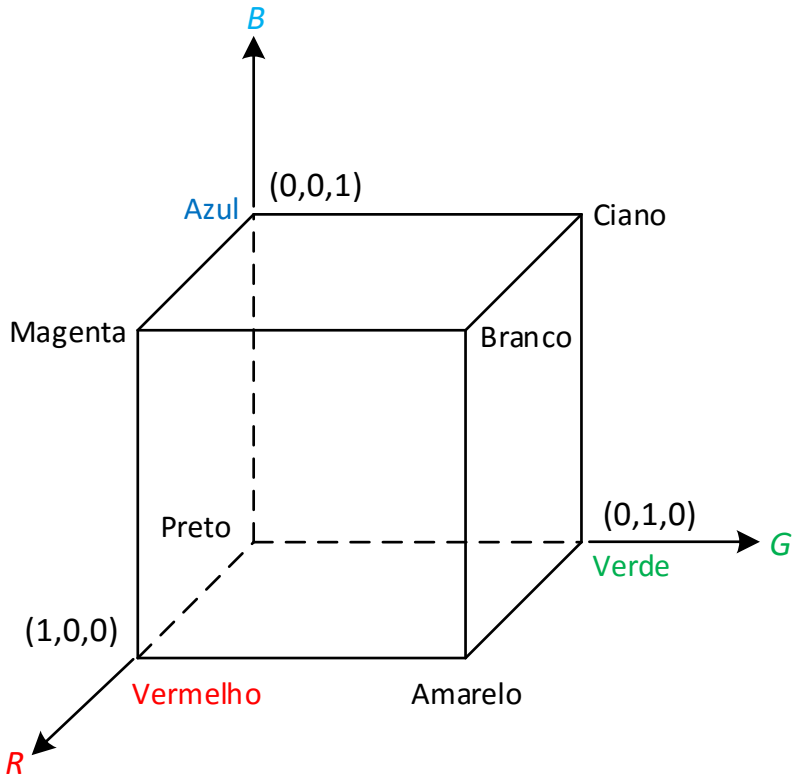


Figura 5: Espaço de cores RGB.

Essa não-linearidade torna difícil para os humanos trabalharem como espaço RGB, uma vez que não é consistente com a forma natural humana de se perceber as cores. Uma forma de se contornar essa limitação é a utilização de representações perceptivas de cores, tal como o espaço HSV.

A conversão do espaço de cores RGB para a escala de cinza é o passo inicial para muitos tipos de algoritmos de análise de imagens. Isso ocorre pois muita das vezes é necessário a redução da quantidade de informação de uma imagem. Apesar de conter

menos informação a escala de cinza preserva a maior parte das características importantes tais como: bordas, regiões, blobs, formas geométricas, entre outras. Uma imagem I_{Cor} pode ser convertida para escala de cinza, I_{Cinza} através da Equação 2.4.

$$I_{Cinza}(m,n) = \alpha I_{Cor}(m,n,r) + \beta I_{Cor}(m,n,g) + \gamma I_{Cor}(m,n,b) \tag{2.4}$$

Onde:

(n, m) é um índice de um pixel em uma imagem em escala de cinza.

(n, m, r) , (n, m, g) e (n, m, b) correspondem a um *pixel* na localização (n, m) para os respectivos canais: vermelho (r), verde (g) e azul (b).

α, β e γ são os coeficientes de ponderação.

Os coeficientes de ponderação são ajustados em proporção a resposta perceptivas do olho humano a cada um dos canais de cores. A conversão do espaço de cores RGB para escala de cinza é uma transformação de imagem não reversível, ou seja, a informação de cor é perdida e não pode ser facilmente recuperada.

O espaço de cores HSV (*Hue, Saturation e Value*) permite superar as limitações do RGB, representando as imagens de uma maneira natural a percepção humana. Cada um desses parâmetros pode ser interpretado da seguinte maneira:

- H (Matiz) corresponde ao comprimento de onda da cor.
- S (Saturação) é a pureza da cor, no sentido de quantidade de luz branca que está misturado.
- V (Value) é o brilho da cor, também conhecido como luminância.

Como é possível perceber, há uma separação entra a parte cromática da iluminação. Isso é um ponto importante em aplicações de analises de imagens, principalmente com relação a segmentação por cor (MENESES; ALMEIDA, 2012). Além disso, em contraste ao RGB mudanças em uma direção dentro do espaço de cores HSV produzem cores consistentes aos valores dos canais. Assim com o RGB o HSV pode ser representado através de uma forma geométrica 3D, neste caso um cone (dentro do espaço RBG), que pode ser visto na Figura 6.

Uma operação comum é a conversão entre os espaços de cores RGB e HSV. Diferentemente da conversão do espaço de cores RGB para escala de cinza, tal conversão é

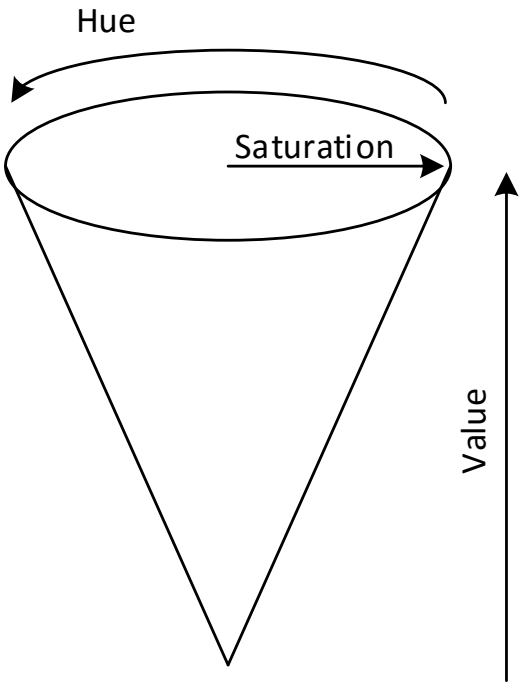


Figura 6: Espaço de cores HSV.

considerada reversível. A Equação 2.5 apresenta a formula para conversão do espaço de cores RGB para HSV.

$$V = \max(R, G, B)$$

$$S = \begin{cases} \dfrac{V - \min(R, G, B)}{V} & \text{Se} \quad V \neq 0 \\ 0 \qquad \text{Senão} \end{cases}$$

$$H = \begin{cases} \dfrac{60 \times (G - B)}{(V - \min(R, G, B))} & \text{Se} \quad V = R \\ 120 + \dfrac{60 \times (B - R)}{(V - \min(R, G, B))} & \text{Se} \quad V = G \\ 240 + \dfrac{60 \times (R - G)}{(V - \min(R, G, B))} & \text{Se} \quad V = B \end{cases} \tag{2.5}$$

Onde:

$\boldsymbol{R, G, B}$ variam de 0 e 1.

H varia de 0 a 360 graus.

S* e *V variam de 0 a 1.

2.2 Equalização de Histogramas

Um histograma consiste em uma representação gráfica da distribuição de um dado numérico. Pode ser interpretado como uma estimativa da distribuição de probabilidade de uma variável continua e foi inicialmente apresentado por Pearson (1894).

No campo de processamento de imagens, histogramas são a base para diversas técnicas de processamento no domínio espacial. Sua manipulação pode ser utilizada para o aprimoramento, compressão e segmentação de imagens, dentre outras aplicações.

Além disso fornecem estatísticas importantes a respeito de uma imagem. Uma das vantagens de sua utilização é a simplicidade do cálculo em software, o que leva a implementações econômicas em hardware, tornando-o uma ferramenta popular em processamento de imagens de tempo real (GONZALEZ; WOODS, 2008).

Para uma imagem digital, um histograma é um gráfico da frequência relativa de ocorrência para cada um dos valores de pixel permitido em uma imagem. Seja uma imagem em escala de cinza o mesmo pode ser construído simplesmente contando-se a o número de vezes que cada valor da escala de cinza (0 a 255), conhecidos como *bins*, aparece dentro da imagem. Cada vez que um determinado valor de intensidade é encontrado, o respectivo *bin* é incrementado. Formalmente, um histograma de uma imagem digital $f(x, y)$, de dimensão $M \times N$, pode ser definido pela Equação 2.6.

$$H_f(C) = \frac{n_c}{M \times N} \tag{2.6}$$

Onde:

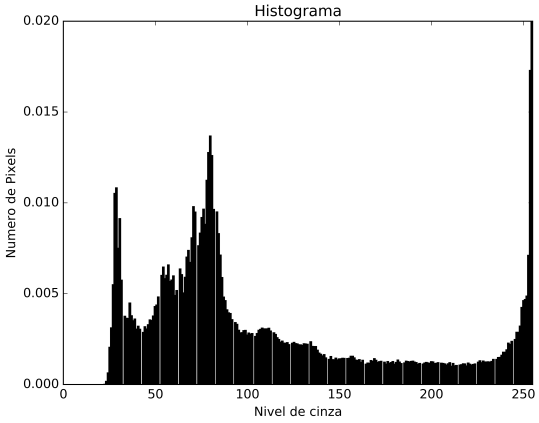
C corresponde aos valores de uma escala de cinza.

$\boldsymbol{n_c}$ corresponde ao número de vezes que o nível de cinza C aparece na imagem.

Através da normalização desse gráfico de frequência, de maneira que a soma de todas as frequências dentro da faixa permitida seja igual a um, é possível tratar o histograma da imagem como uma função de densidade de probabilidade discreta. Tal função define a probabilidade de um determinado valor de pixel ocorrer dentro da imagem e é um conceito base para diversos tipos de analises.

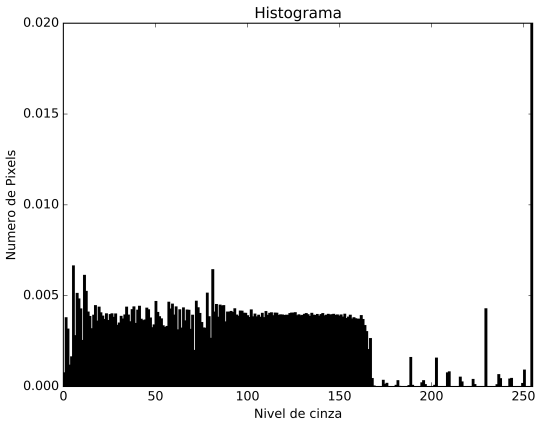
A inspeção visual de um histograma pode revelar informações do contraste básico de uma imagem. Da mesma forma, permite a distinção de qualquer potencial diferença entre a distribuição de cor do plano de fundo e primeiro plano de uma imagem, sendo possível até mesmo a separação destes planos em duas imagens (SOLOMON; BRECKON, 2011). A Figura 7 mostra tipos de contrastes de uma imagem e o respectivo histograma.

(a) Imagem com baixo contraste.



(b) Histograma de (a).

(c) Imagem com alto contraste.



(d) Histograma de (c).

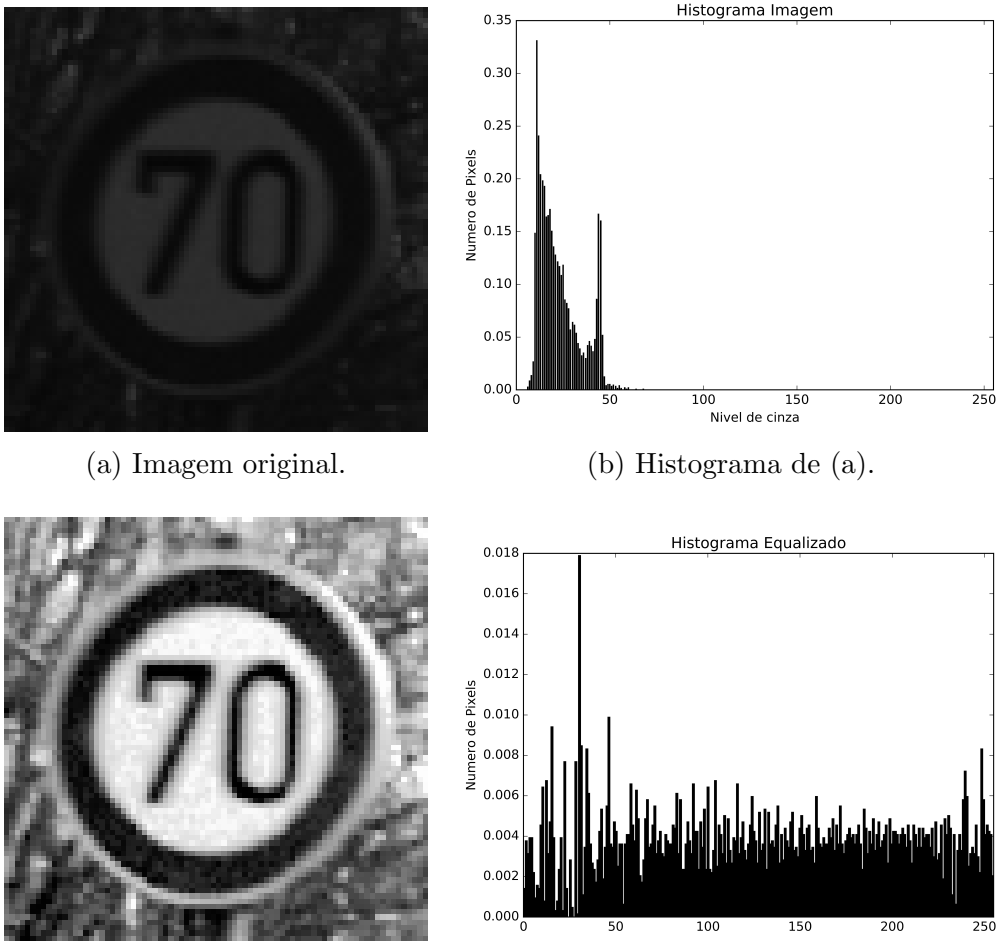
Figura 7: Imagens de contrastes diferentes e respectivos histogramas. FONTE: (HOUBEN et al., 2013)

O realce de contraste é uma operação de pré-processamento bastante comum em aplicações de reconhecimento de objetos. Tem-se que o contraste entre dois objetos em uma imagem pode ser definido como a razão entre seus cinzas médios. Com o intuito de aumentar a discriminação entre eles é necessário a manipulação de seus contrastes através

do remapeamento de cada um dos *pixels* da imagem.

Existem várias técnicas que permite aplicar esse conceito. Alguns exemplos são: expansão de contraste, equalização de contraste, equalização de contraste adaptativa, CLAHE (*Contrast Limited Adaptive Histogram Equalization*) (ZUIDERVELD, 1994), dentre outros. A sua escolha geralmente se dá de forma empírica, porém a análise prévia do histograma pode se demonstrar bastante útil (QUEIROZ; GOMES, 2006).

A equalização de contraste é uma das técnicas mais utilizadas para o melhoramento de contraste de imagens (SOLOMON; BRECKON, 2011). O fato de não necessitar o ajuste de parâmetros, sendo considerado um procedimento totalmente automático, e ser computacionalmente simples de ser executado são dois atrativos desse algoritmo. A Figura 8 ilustra a aplicação da equalização em uma imagem de baixo contraste.



(a) Imagem original.

(b) Histograma de (a).

(c) Imagem equalizada.

(d) Histograma de (c).

Figura 8: Exemplo de aplicação de histograma em uma imagem de baixo contraste. FONTE: (STALLKAMP et al., 2012)

Essa técnica emprega uma função não linear e monotônica que mapeia os pixels da imagem de entrada em uma imagem de saída com PDF (*Probability Density Function*)

uniforme. Esse mapeamento pode ser descrito com uma função da intensidade $s_k = T(r_k)$, onde r_k é o valor da intensidade de um determinado pixel na imagem de entrada em escala de cinza com $L-1$ níveis (0 a 255, para uma imagem de 8-*bit*) e s_k é a intensidade mapeada para o respectivo pixel.

A função de mapeamento s_k é descrita pela Equação 2.7. Esse mapeamento pode ser definido como uma histograma cumulativo, CDF (*Cumulative Distribution Function*), $C(i)$, de tal forma que cada ponto deste histograma corresponde a soma da frequência de ocorrência de cada nível de cinza anterior a um determinado *bin* i, incluindo-o.

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) \tag{2.7}$$

Onde:

$\boldsymbol{P_r}$ é a PDF da imagem.

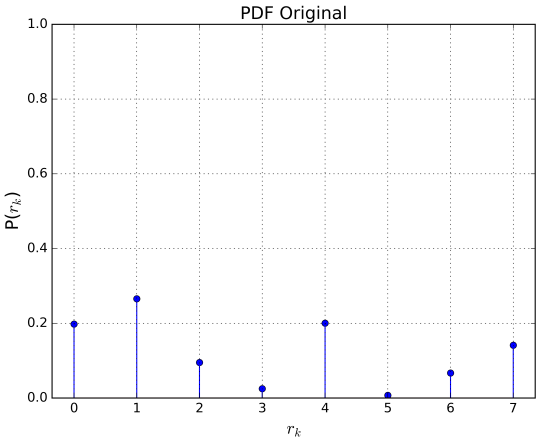
$\boldsymbol{k} = 0, 1, 2 \ldots, L-1$.

Essa equação gera ao final uma lista de correspondência, que pode ser usada como *lookup table* para remapear as intensidades dos *pixels* da imagem original. De forma a exemplificar, considere uma imagem hipotética, em escala de cinza, de dimensão 32×32 *pixels* com 3 *bits* para representar os níveis de cinza. Sendo assim são 4096 *pixels* e o valor de L corresponde a 8 níveis de cinza. A distribuição de cada um desses níveis, r_k, é dada pelos valores de n_k na Tabela 2.

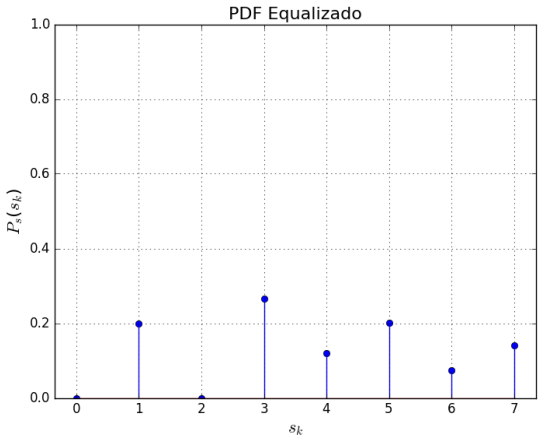
Em seguida os valores de n_k são normalizados, dividindo-os pelo total de pixel, dando origem a função $P_r(r_k)$, PDF, como pode ser observado na Figura 9a. Esses valores são então utilizados na Equação 2.17 para encontrar os valores de mapeamento correspondentes, s_k, a cada um dos níveis, r_k, como descrito na Tabela 2. Essa tabela também contem os valores da distribuição, $p_r(r_k)$, e número de *pixels*, n_k, para cada uma das intensidades do exemplo hipotético. Tais valores foram utilizados para o calculo de s_k. O gráfico da Figura 9c demonstra esse mapeamento, para o dado exemplo.

Tabela 2: Mapeamento (*lookup table*) de uma intensidade r_k para s_k

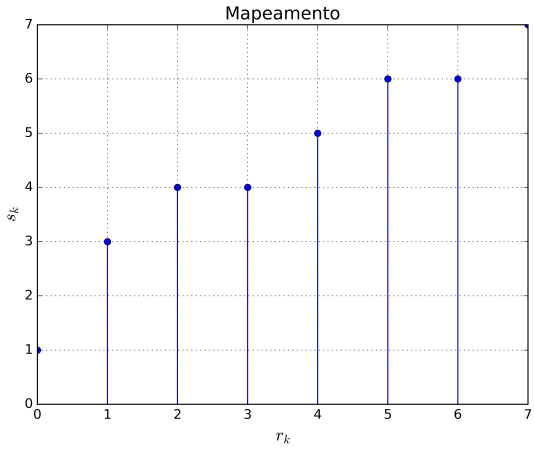
r_k	n_k	$p_r(r_k)$	s_k
0	811	0,20	1
1	1088	0,27	3
2	390	0,01	4
3	103	0,03	4
4	820	0,20	5
5	30	0,08	6
6	275	0,07	6
7	579	0,14	7



(a) Gráfico da PDF da imagem original.



(b) Gráfico da PDF imagem equalizada.



(c) Gráfico do mapeamento de r_k para s_k.

Figura 9: Gráficos de uma equalização de contraste apresentado na Tabela 2.

Como pode-se perceber o PDF da imagem equalizado da Figura 9b não é totalmente

uniforme. Isso ocorre, pois, o histograma da imagem de entrada é uma aproximação para uma PDF. Sendo assim distribuições perfeitamente uniformes são raras em aplicações de equalização de histograma. Diferentemente da sua versão continua, não pode ser provado que a equalização de um histograma discreto irá resultar em uma distribuição uniforme (SOLOMON; BRECKON, 2011).

2.3 Filtro Mediana

Nem sempre uma imagem apresenta as melhores condições para ser manipulada e analisada. De acordo Chan, Ho e Nikolova (2004), ruídos indesejados podem ser introduzidos em uma imagem por diversos fatores tais como: erros de quantização ou transmissão, variação na sensitividade do sensor, variações do ambiente, mal funcionamento do sensor, entre outros. Dessa forma, um ruído de imagem pode ser definido como qualquer corrosão ao sinal de imagem, causado por um distúrbio externo(JASSIM, 2013).

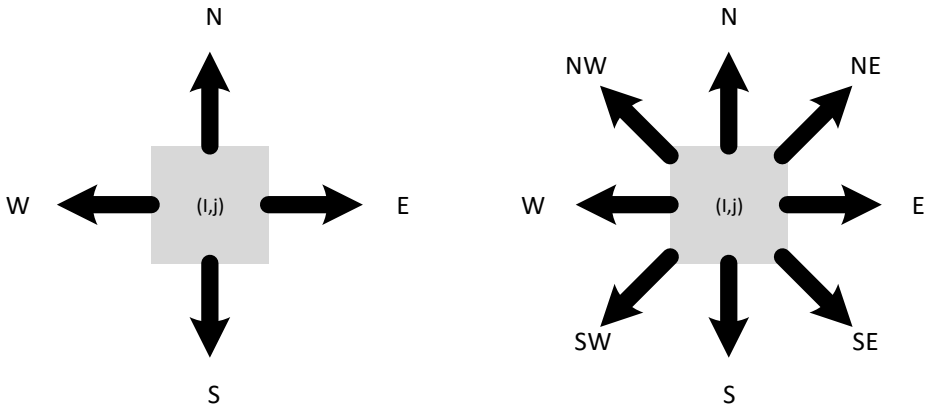
Imagens digitais são frequentemente corrompidas por ruídos de impulso conhecido como *Salt-and-Pepper*. Tais ruídos podem ser provenientes de erros de transmissão, falhas de localizações de memória ou erros de sincronização na conversão analógica para digital (NAIR; REVATHY; TATAVARTI, 2008). Em imagens corrompidas por esse tipo de ruídos, os *pixels* ruidosos somente podem tomar valores máximos e mínimos em uma faixa dinâmica.

Em uma imagem monocromática usualmente altas frequências estão relacionadas a feições (*features*) conhecidas como bordas, enquanto que baixas frequências a áreas uniformes (QUEIROZ; GOMES, 2006). No campo de processamento de imagens digitais, filtros são usados principalmente para suprimir altas e baixas frequências. O objetivo principal desse procedimento é o melhoramento de uma imagem, de tal forma que a informação visual contida nela possa ser analisada de maneira mais clara. As melhorias consistem em remoção de ruídos, aumento da nitidez das bordas da imagem, suavização da imagem, ressaltar características, entre outros.

O processo de filtragem pode ocorrer tanto no domínio da frequência quanto no domínio do espaço. O primeiro consiste em se transformar uma imagem para o domínio da frequência, multiplica-la por uma função de filtro de frequência e por fim transformar o resultado para o domínio do espaço. A função de filtro é projetada para atenuar algumas frequências, enquanto que realça outras. O filtro passa-baixa é um exemplo de uma função simples, onde apresenta o valor 1 para todas frequências menores que uma frequência de corte e 0 para todas as outras.

De forma similar as técnicas de ajuste de contraste, a filtragem no domínio do espaço implica na transformação *pixel* a *pixel*. A diferença está presente na forma com que um *pixel* é alterado. No ajuste de contraste a alteração depende apenas do nível de cinza do *pixel* correspondente na imagem original, enquanto que a filtragem espacial depende dos níveis de *pixel* situado em uma vizinhança.

Muitas operações em processamento de imagens utilizam-se desse conceito de vizinhança,Figura 10, local para definir uma área local de interesse, influência ou relevância. Relacionado a este conceito está a definição da conectividade dos *pixels*. Isso determina quais *pixels* estão conectados uns aos outros. Por exemplo: uma conectividade igual a 4, Figura 10a, somente os *pixels* W, N, E, S estão ligados ao *pixel* central (i, j). A maior parte dos algoritmos utilizam uma conectividade 8 por padrão (SOLOMON; BRECKON, 2011).



(a) Conectividade 4. (b) Conectividade 8.

Figura 10: Possíveis conectividades de um pixel.

Existem duas classes de filtros: lineares e não-lineares. Em um filtro espacial linear um valor novo ou filtrado de um determinado *pixel* é uma combinação linear dos valores dos *pixels* de sua vizinhança. Qualquer outro tipo de filtro é considerado não-linear.

A filtragem espacial se baseia amplamente na operação de convolução (QUEIROZ; GOMES, 2006). A aplicação de um filtro consiste na convolução de uma máscara e uma imagem. Também conhecida como *kernel*, a máscara é definida como uma matriz de dimensões inferiores a imagem a ser filtrada. Geralmente os *kernel* são de dimensões quadradas e seus valores representam pesos a serem aplicados sobre os *pixels* de uma determinada vizinhança de um *pixel* central.

O procedimento geral pode ser visto como uma aplicação de uma série de operações

em uma vizinhança utilizando como base o princípio de janela deslizantes, em que cada um dos *pixels* da imagem de entrada é processado com uma operação realizada numa vizinhança local de $N \times N$ *pixels* O resultado de cada operação local substitui o *pixel* central da vizinhança, compondo, assim, a imagem final. Matematicamente a operação de convolução, no domínio do espaço, é representada pela Equação 2.8.

$$I_s(x,y) = I_e(x,y) * m(i,j) = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} I_e(x-i, y-j) \cdot m(i,j) \tag{2.8}$$

Onde:

$\boldsymbol{I_s}$ é a imagem resultante.

$\boldsymbol{I_e}$ é a imagem de entrada.

$\boldsymbol{m}$ é a máscara de convolução.

$\boldsymbol{x, y}$ são índices da imagem.

$\boldsymbol{i, j}$ são índices da máscara de convolução.

A Figura 11 ilustra a aplicação de uma máscara de convolução fictícia de conectividade 8. Pode-se observar que o processo consiste na multiplicação dos pesos do *kernel* pelos respectivos *pixels* da vizinhança, e na substituição do *pixel* central pelo resultado.

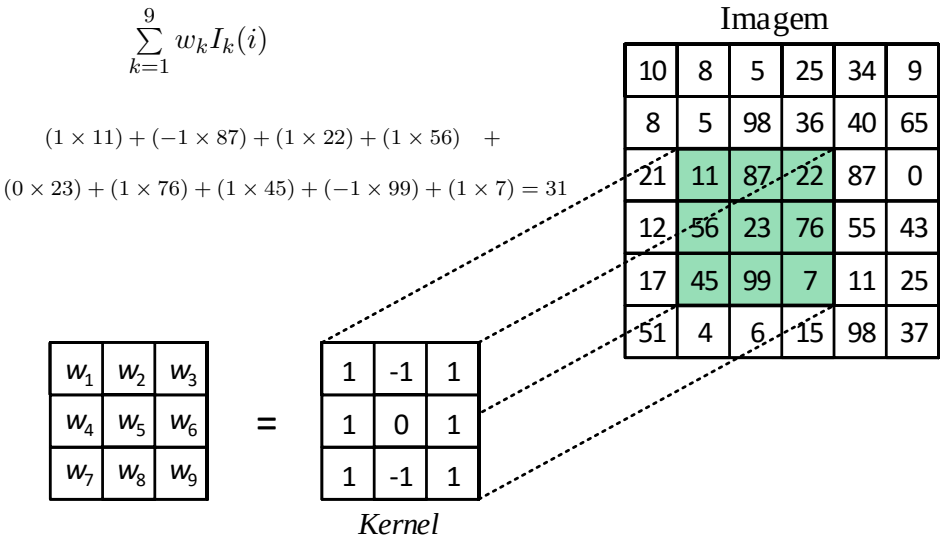


Figura 11: Exemplo do processo de convolução para uma imagem fictícia.

A aplicação do *kernel* em *pixels* da fronteira pode gerar problemas, uma vez, que parte do *kernel* fica fora da imagem. Existem três principais abordagens para se lidar com esse tipo de situação, sendo que o método 2 e 3 são preferíveis por não introduzirem artefatos na imagem final. Tais abordagens compreendem:

1. Não aplicar a máscara nessas regiões de fronteira.
2. Aplicar a filtragem somente nos *pixels* que estão dentro da fronteira.
3. Espelhar os valores da região de fronteira, para que se possa aplicar a filtragem completa.

Pertencentes a classe de filtros não-lineares, os filtros de ordem estatística têm sua resposta baseada na classificação (ordenação) dos *pixels* contidos em uma determinada vizinhança. O filtro de mediana é o filtro mais conhecido nessa categoria (GONZALEZ; WOODS, 2008).

Eles são populares pois para certos tipos de ruídos aleatórios apresentam uma boa capacidade de redução de ruído, principalmente *Salt-and-Pepper*. Juntamente a isso, geram muito menos efeito *blur*, quando comparados com filtros de mesmo propósito, tais como filtro de média e moda(QUEIROZ; GOMES, 2006; GONZALEZ; WOODS, 2008). A Figura 12 apresenta um comparativo entre tais filtros.

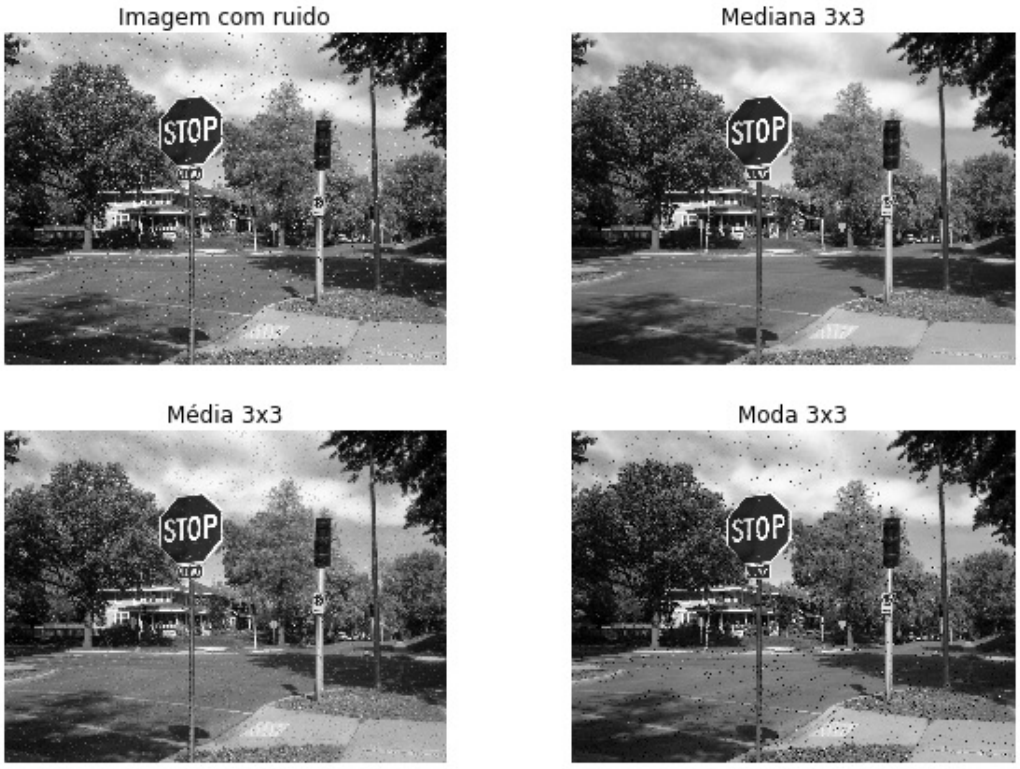


Figura 12: Comparativo da remoção de ruídos, para uma vizinhança 3×3 entre os filtros mediana, média, moda. FONTE:(HUTCHISON, 2016)

Sua aplicação consiste no cálculo da mediana, considerando os valores de intensidade da vizinhança e incluindo o *pixel* central. A mediana é dada através da ordenação dos valores de intensidade dos *pixels*. Caso o número de *pixels* totais seja ímpar, o valor do *pixel* na posição central dessa lista ordenada será a mediana. Se for par, a mediana corresponde à média dos valores dos dois *pixels* na posição central. A Figura 13 ilustra essa operação para uma dada vizinhança 3×3 com conectividade 8.

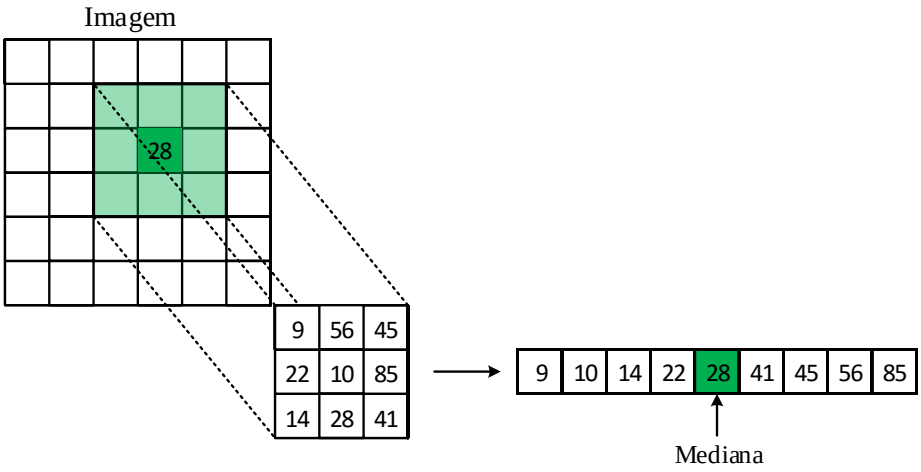


Figura 13: Cálculo do filtro de mediana para uma vizinhança 3×3 com conectividade 8.

O filtro de mediana tende a produzir uma suavização proporcional ao tamanho de sua vizinhança. Porém, como é possível de se observar na Figura 14, em que o filtro é aplicado para tamanhos diferentes de vizinhança, há uma preservação na definição das bordas, mesmo para uma vizinhança 41×41.

Figura 14: Filtro mediana aplicado com três tamanhos diferentes de vizinhança (*kernel*). FONTE:(HUTCHISON, 2016)

2.4 Componentes Conectadas

Uma imagem digital é considerada binária quando possui somente dois valores possíveis para cada *pixel*, 1 ou 0. Geralmente o conjunto de *pixels* de valor 1 sãos referenciados como primeiro plano da imagem (*foreground*), enquanto que o grupo formado por valores 0 são chamados de plano de fundo (*background*). Dessa maneira, um objeto em uma imagem binária é representado por um grupo qualquer de *pixels* conectados (SOLOMON; BRECKON, 2011).

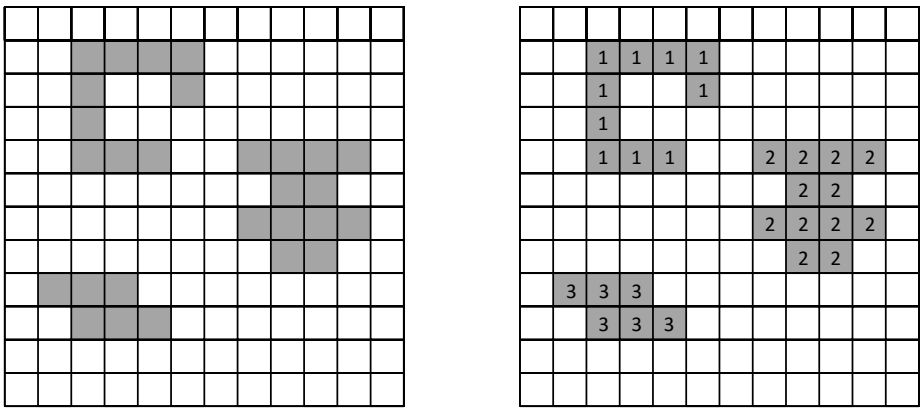
Imagens binárias podem ser utilizadas como entrada para diversos tipos de algoritmos em processamento de imagens. Para Jain, Kasturi e Schunck (1995), mesmo com avanços no poder computacional, algoritmos baseados nesse tipo de imagens continuam muito uteis. Suas propriedades são bem compreendidas e estudadas. Além disso, algoritmos que operam nesse tipo imagem tendem a serem mais rápidos e consumir menos memória do que em imagens coloridas ou em escala de cinza. Entre suas diversas aplicações possíveis pode-se citar: segmentação de imagens, extração de características, contagem de objetos, operações morfológicas entre diversas outras.

A análise de uma imagem consiste na interpretação de uma cena ou dos objetos contidos nela (RAMALHO, 2013). O primeiro passo para muitas analises compreende o isolamento de objetos de interesse em uma determinada cena. Um procedimento possível para tal objetivo é a aplicação da segmentação de imagens.

O produto de tais técnicas são imagens binárias contendo várias regiões de interesse, conhecidas como *blobs*, que devem ser analisadas através de seus respectivos elementos estruturais, tais como área, perímetro, momentos, etc. Para tanto, é necessário a extração de tais regiões desconexas, e uma forma de se realizar isso é através de algoritmos de rotulação de componentes conectadas.

Em teoria dos grafos, uma componente conectada corresponde a um subgrafo, de um grafo não direcional, onde quaisquer dois vértices são conectados através de um caminho. Esse conceito é aplicado ao algoritmo de rotulação de componentes conectadas em imagens, Figura 15, em que um subgrupo de componentes conectadas é unicamente rotulado através de uma heurística.

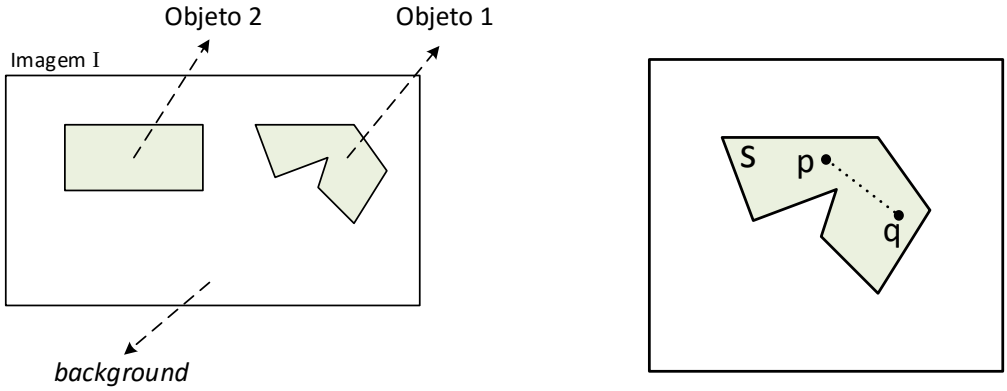
Seja uma imagem I, Figura 16a, onde todos *pixels* iguais a 1 são chamados de primeiro plano e denotados por S. Um *pixel* $p \in S$ está conectado a outro *pixel* $q \in S$, Figura 16b, se houver um caminho de p a q consistindo inteiramente de *pixels* de S. Dessa forma uma componente conectada de uma imagem é um conjunto de *pixels* em que cada *pixel* é



(a) Imagem binária (b) Componentes conectadas

Figura 15: Componentes conectada (a) de uma imagem binária (b).

conectado a todos outros *pixels*.



(a) Imagem binária contendo dois objetos. (b) Caminho entre p e q.

Figura 16: Ilustração da existência de um cainho em uma imagem binária.

Outro conceito importante é o de conectividade, tratado na Seção 2.3 Para uma vizinhança 8 é possível dois tipos de conectividade: 4 e 8. Na conectividade-4 somente os *pixels* acima, abaixo, a direita e a esquerda do pixel central são consideradas. Já na conectividade-8 são considerados todos *pixels* da vizinhança.

Também se define uma adjacência para cada uma desses tipos de vizinhança. Seja um pixel central igual a 1, a adjacência para uma determinada conectividade desse *pixel* corresponde aos *pixels* em sua vizinhança iguais a 1. A Figura 17 ilustra uma adjacência hipotética para as conectividade-4 e conectividade-8.

1	0	1
1	1	1
0	1	0

(a) Adjacência-4.

	0	
0	1	1
	1	

(b) Adjacência-8.

Figura 17: Possíveis conectividades de um pixel.

O tipo de adjacência escolhido pode influenciar na conexão entre duas regiões. Considere a Figura 18, caso a adjacência-4 seja escolhida, as regiões $R1$ e $R2$ estarão desconexas. Em contrapartida, se a adjacência-8 for usada $R1$ e $R2$ estarão conectadas.

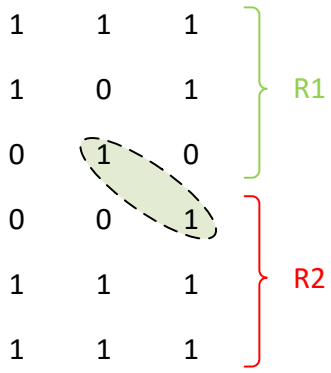


Figura 18: Regiões R1 e R2 só estão conectadas se considerado uma adjacência-8.

Existem diversos algoritmos para tratar o problema de rotulação de componentes conectadas. Em seu trabalho Grana, Borghesani e Cucchiara (2009) faz um comparativo da performance entre vários deles e um histórico da evolução da área. Algum desses algoritmos assumem que a imagem inteira cabe na memória e empregam uma solução recursiva.

Segundo Shah (1997) tais algoritmos funcionam bem e são simples de implementar. Porém, quando executados em computadores com recursos limitados podem facilmente ocasionar estouro de pilha (*stack overflow*). Além disso existem outros algoritmos que foram projetados para se trabalhar em maquinas paralelas e utilizam uma estratégia de propagação paralela (SHAPIRO; STOCKMAN, 2001).

Outra abordagem possível utiliza-se de uma estratégia sequencial, que analisa a imagem, duas linhas por vez, da esquerda para direita e de cima para baixo. Um algoritmo sequencial clássico, baseado na teoria de grafos, foi descrito por Rosenfeld e Pfaltz (1966). Ele consiste em duas passagens pela imagem: o primeiro passo armazena as equivalências e atribui rótulos temporários, enquanto que o segundo substitui cada um dos rótulos

temporários pelo rótulo de sua classe de equivalência (SHAPIRO; STOCKMAN, 2001).

Os passos do algoritmo são apresentados na Figura 19. Uma imagem é escaneada sequencialmente e para cada *pixel* P são analisados seus vizinhos da esquerda, E, e acima, C. Caso um *pixel* P de valor 1 seja encontrado um novo rótulo é atribuído a ele, dependendo das seguintes regras:

- Se ambos *pixels* C e E forem 0, um novo rótulo L é atribuído a P.
- Se E ou C ou ambos tiverem o rótulo L, o mesmo é atribuído a P.
- Se E e C tiverem rótulos diferentes, atribui-se L ao *pixel* P e adiciona uma nova entrada numa tabela de equivalência $L = M$.

Essas regras são ilustradas, de forma simplificada, Figura 19. No segundo passo os rótulos dos *pixels* são modificados para os rótulos de suas classes de equivalência. Suponha que após a primeira passagem a imagem possua rótulos de 1 a 12, e os seguintes rótulos são considerados iguais: (1,2), (12,11), (3,6), (1,7), (2,5), (10,12), (8,6) e (9,11). Através disso é possível se chegar as seguintes classes de equivalência: (1,2,5,7), (3,6,8) e (9,10). Na segunda passagem os rótulos 2,5,7 serão substituídos por 1, os rótulos 6,8 por 3 e os rótulos 10 por 9.

1. Percorra esquerda para a direita, de cima para baixo
2. A Se um *pixel* sem rotulo tem o valor 1, atribua um novo rótulo para ele de acordo com as seguintes regras:

$$\begin{array}{cc} & 0 \\ 0 & 1 \end{array} \rightarrow \begin{array}{cc} & 0 \\ 0 & L \end{array} \qquad \begin{array}{cc} & 0 \\ L & 1 \end{array} \rightarrow \begin{array}{cc} & 0 \\ L & L \end{array}$$

$$\begin{array}{cc} & L \\ 0 & 1 \end{array} \rightarrow \begin{array}{cc} & L \\ 0 & L \end{array} \qquad \begin{array}{cc} & L \\ M & 1 \end{array} \rightarrow \begin{array}{cc} & L \\ M & L \end{array} \quad (L = M)$$

3. Determine a classes de equivalência dos rótulos.
4. Na segunda passagem, atribua o mesmo rótulo a todos elementos dentro de uma classe de equivalência.

Figura 19: Passos de um algoritmo de componentes conectadas. FONTE: Shah (1997).

Uma forma eficiente de se determinar as classes de equivalência é por meio do algoritmo *union-find* (CORMEN et al., 2001), que constrói dinamicamente as classes de equivalência a medida que são encontradas novas equivalências. A sua estrutura de dados é representada por árvores, que permitem uma facilidade na construção e manipulação das classes de equivalência. O proposito dessa estrutura de dados é armazenar uma coleção de conjuntos disjuntos e implementar de forma eficiente as operações de *union*, mesclar dois conjuntos em um, e *find*, determinar qual conjunto um determinado elemento se encontra. A adição dessa estrutura de dados ao algoritmo clássico apresenta grandes melhorias de desempenho (SHAPIRO; STOCKMAN, 2001).

2.5 Detecção

2.5.1 Framework Viola e Jones

O *framework* de detecção Viola & Jones foi proposta em (VIOLA; JONES, 2001) para solucionar o problema de detecção de faces, sendo o primeiro a prover taxas de detecção de objetos em tempo real. Dentre todos os tipos de detectores de faces atuais o mesmo introduzido por Viola e Jones (2001) é o mais conhecido e amplamente utilizado nas mais diversas áreas do conhecimento, tais como: detecção de carros em imagens aéreas (GRABNER et al., 2008), detecção de pedestres (ARAÚJO et al., 2011), detecção de sorriso (DÉNIZ et al., 2008), diagnóstico de câncer (VINK et al., 2013), entre outras.

Diferente de outras abordagens que utilizam janelas deslizantes (WANG; HAN; YAN, 2009) , Viola e Jones (2001) constrói um detector que percorre a imagem de forma eficiente. Através da criação de um classificador em cascata, regiões que não possuam qualquer semelhança com o objeto de interesse são rapidamente rejeitadas. Dessa maneira, os recursos computacionais são destinados primordialmente na análise de regiões com alta probabilidade de conterem o objeto de interesse. Dentre as características que tornam este *framework* um bom algoritmo de detecção pode-se destacar as seguintes:

- Robustez devido à alta taxa de detecção e baixa taxa de falsos positivos.
- O algoritmo pode ser executado em tempo real, dependendo do hardware.
- Invariância a escala e localização.
- Cálculo das características extremamente rápido.
- Esquema de detecção genérico, que pode ser aplicado nos mais diversos problemas.

Apesar dessas vantagens, o *framework* (VIOLA; JONES, 2001) também apresenta suas limitações. Dentre elas pode-se citar:

- Dificuldade em detectar imagens rotacionadas com ângulo de 45 graus;
- Sensibilidade a variações de luminosidade e a grandes oclusões;
- Tempo de treinamento demorado (MCCANE et al., 2005);
- Determinação dos parâmetros de construção da cascata de classificação são dependentes da experiência do utilizador (MCCANE et al., 2005).

De forma simplificada, para um melhor entendimento, o treinamento do *framework* pode ser dividido em quatro blocos, descritos na Figura 20. Esses blocos são: Extração de Características, Seleção de Características, Treinamento de Classificadores, Construção da Cascata de Classificadores.

Figura 20: Módulos do *framework* Viola & Jones

A primeiro bloco consiste na extração de características de uma sub-janela de tamanho pré-determinado. A detecção de objetos é baseada no valor de características simples, semelhantes as funções de base *Haar*, utilizadas no trabalho de Papageorgiou, Oren e Poggio (1998) . Devido a semelhança, estas características também são conhecidas como *Haar-like*.

Na abordagem de Viola & Jones foram utilizados quatro tipos de características *Haar-like* básicos, como vistos na Figura 21. Para uma janela de detecção pondem haver milhares de variações desses tipos básicos, com tamanho e posições diversas O cálculo é realizado através da diferença entre a soma dos *pixels* da região branca e a soma dos *pixels* da região preta.

Apesar de simples, o conjunto de características retangulares, para um detector de tamanho consideravelmente pequeno, pode ser imenso. Com o intuito de acelerar o cálculo foi proposto o conceito de imagem integral.

Uma imagem integral consiste em uma representação intermediaria de uma imagem.

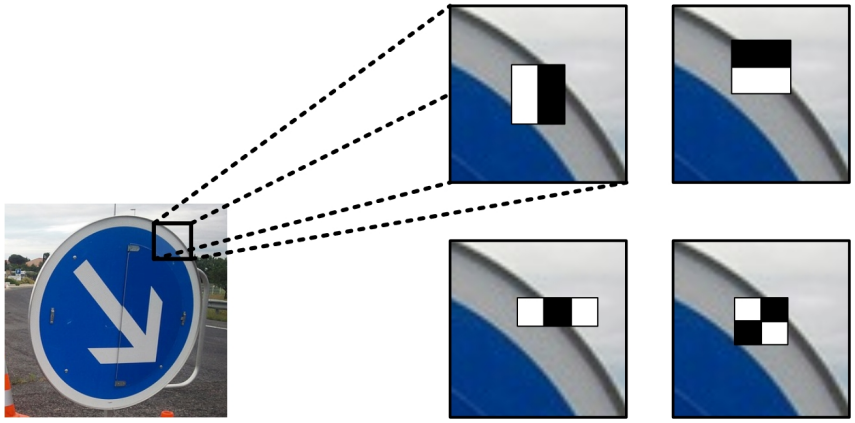


Figura 21: Haar-like básicos para uma janela de detecção.

Seu cálculo é realizado apenas um passo pela imagem original e apenas uma vez. Através desta imagem resultante, a extração de características *Haar-like* pode ser realizada em qualquer escala e em tempo constante.

Uma localização x e y, em uma imagem integral, contém a soma dos *pixels* acima e a esquerda destas coordenadas, incluindo-as, como descrito na Equação 2.9. Essa equação utiliza as recursões dadas pela Equação 2.10.

$$I(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{2.9}$$

Onde:

$\boldsymbol{I(x,y)}$ é a imagem integral.

$\boldsymbol{i(x,y)}$ é a imagem original.

$$\begin{aligned} s(x,y) &= s(x,y-1) + i(x,y) \\ I(x,y) &= I(x-1,y) + s(x,y) \end{aligned} \tag{2.10}$$

Onde:

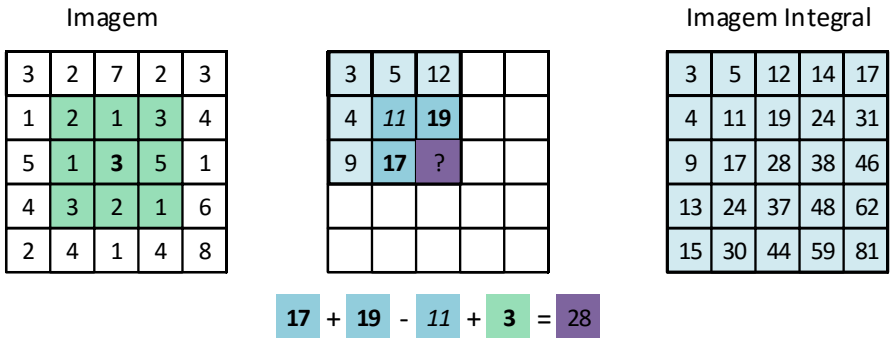
$\boldsymbol{s(x,y)}$ é a soma cumulativa da linha.

$\boldsymbol{s(x,-1)}$ **e** $\boldsymbol{I(-1,y)}$ são iguais a zero.

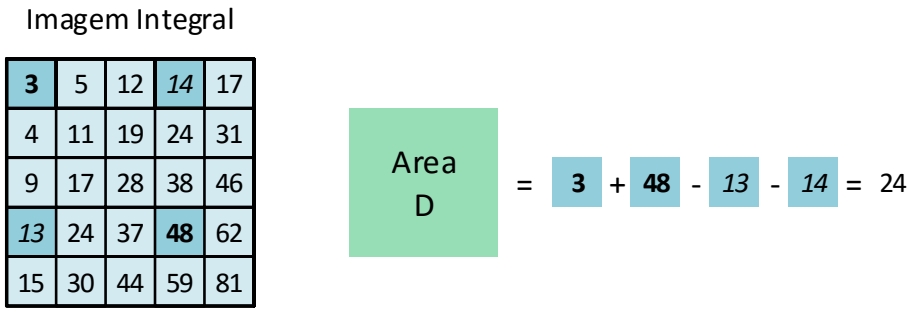
A Figura 22 ilustra um exemplo de imagem integral. Como pode ser observado o cálculo da soma de *pixels* em um determinado retângulo D, Figura 22b, pode ser feita com apenas quatro referências. A diferença entre duas somas retangulares pode ser calculada

através de oito referências. Levando-se em consideração o cálculo das características apresentadas na Figura 21 tem-se a seguinte quantidade de referências necessárias:

- Se estes retângulos forem adjacentes somente seis referências são necessárias;
- Se forem três retângulos são necessárias 8 referências;
- Se forem 4 retângulos são necessárias nove referências.



(a) Criação de uma imagem integral.



(b) Cálculo da área verde na imagem (a), utilizando a imagem integral.

Figura 22: Exemplo de cálculo e uso de uma imagem integral. Valores em negrito soma-se enquanto que, itálico subtrai

O trabalho de Viola e Jones (2001) foi o primeiro a introduzir o conceito de *boosting* na comunidade de visão computacional (SZELISKI, 2010) . O processo conhecido como *boosting* (SCHAPIRE, 1990) envolve o treinamento de uma série de classificadores fracos, que posteriormente são agrupados constituindo um classificador forte. Assim, o *boosting* consiste na construção de um classificador $h(x)$ como a soma de simples classificadores

fracos, como descrito na Equação 2.11.

$$h(x) = sgn\left[\sum_{j=0}^{m-1} \alpha_j h_j(x)\right] \tag{2.11}$$

Onde:

$\boldsymbol{h_j(x)}$ são classificadores fracos.

$\boldsymbol{m}$ corresponde ao número de classificadores fracos.

$\boldsymbol{\alpha_j}$ são pesos.

Cada um dos classificadores fracos $h_j(x)$ é uma função, extremamente simples, da entrada e dessa maneira não contribui muito, de forma isolada, para o desempenho do classificador. Em grande parte das variantes dos algoritmos de *boosting*, os classificadores fracos, são funções de *threshold*, Equação 2.12, conhecidas também como *decision stumps*, que são a forma mais simples possível de uma árvore de decisão, de profundidade igual a um).

$$h_j(x) = a_j[f_j < \theta_j] + b_j[f_j \geq \theta_j] = \begin{cases} a_j, & \text{Se } f_j < \theta_j \\ b_j, & \text{Caso contrário} \end{cases} \tag{2.12}$$

Onde:

$\boldsymbol{f_j}$ é o descritor de caracteristica (*feature*).

$\boldsymbol{\theta_j}$ é o valor de *threshold*.

$\boldsymbol{a_j}$ **e** $\boldsymbol{b_j}$ são escolhidos como ± 1, como por exemplo $a_j = -s_j$ e $b_j = +s_j$. Dessa forma somente f_j, $theta_j$ e $sj \in \pm 1$ precisam ser selecionados.

Dado uma janela de detecção de dimensões 24×24 *pixels* tem-se por volta de 180.000 características retangulares associadas (VIOLA; JONES, 2004). No entanto, somente algumas destas características podem ser combinadas em um classificador efetivo. Dessa maneira, o segundo bloco do *framework* consiste na seleção destas características retangulares, chamados também de classificadores fracos. Com base nesse objetivo, o algoritmo de aprendizagem fraca foi projetado para selecionar uma única característica retangular que melhor separa os exemplos positivos dos negativos.

Para cada característica o classificador fraco (*weak learner*) determina uma função de classificação de *threshold* ótimo, de tal maneira que um número mínimo de exemplos sejam classificados erroneamente. A Equação 2.13 representa essa função de classificação.

$$h_j(x) = \begin{cases} 1, & \text{Se } p_j f_i(x) < p_j \theta_j \\ 0, & \text{Caso contrário} \end{cases} \tag{2.13}$$

Onde:

$\boldsymbol{h_j(x)}$ é o classificador fraco.

$\boldsymbol{f_i}$ é uma característica fraca.

$\boldsymbol{\theta_j}$ corresponde ao *threshold*.

$\boldsymbol{p_j}$ é a paridade, indicando a direção do sinal de desigualdade.

Com o intuito de solucionar tanto o problema de seleção de características quanto o treinamento do classificador, Viola e Jones (2001) utiliza uma variação do algoritmo conhecido como *AdaBoost* (*Adaptative Boosting*).O algoritmo original foi proposto por Freund e Schapire (1995) e consiste na seleção de classificadores fracos para construção de um classificador forte, assim como os algoritmos de *boosting*. No entanto, a variação proposta por (VIOLA; JONES, 2001) se diferencia por alterar os pesos de cada amostra (imagem de treino), através de uma função que verifica se estão corretamente classificados em cada um dos estágios de seleção do classificador fraco. A atualização desses pesos é feita de tal maneira que reduz a contribuição de exemplos que foram corretamente classificados, proporcionalmente ao erro total de classificação.

Na Figura 23 é possível se observar o funcionamento do algoritmo AdaBoost. Inicialmente é selecionado um classificador fraco para a separação de duas classes (bolas azuis e roxas). Esse classificador só consegue classificar corretamente poucos exemplos. Os exemplos classificadores erroneamente tem seu pesos incrementado (aumentam de tamanho).

Então, outro classificador fraco é selecionado e o mesmo tenta evitar cometer os mesmos erros anteriores. Esse novo classificador também classifica exemplos de forma errada e esses tem seus pesos incrementados. O processo de seleção, de um novo classificador fraco e incremento dos pesos dos exemplos classificados de forma errada, é repedido diversas vezes até que se chegue a um erro mínimo esperado. O classificador forte é formado pela combinação linear de todos classificadores fracos selecionados durante o treinamento.

Cada um desses classificadores possui um grau de importância (pesos), baseado em sua taxa de acertos, na formação do classificador forte.

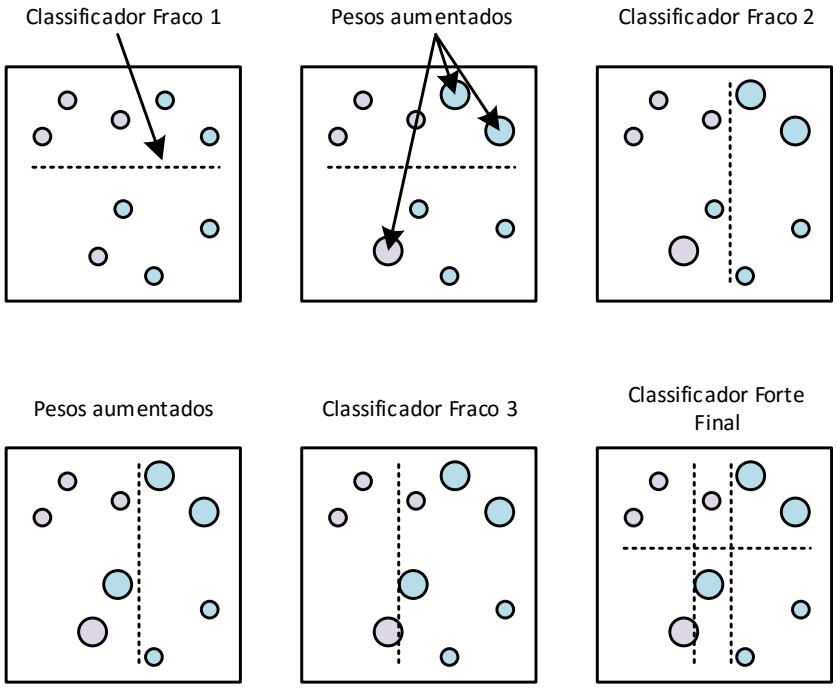


Figura 23: Funcionamento do algoritmo *Adaboost*. Baseado em (SZELISKI, 2010).

Os pesos α_j dos classificadores fracos, presentes na Equação 2.9, são determinados através da média do erro de classificação em cada estágio. A Figura 24 apresenta a variação do algoritmo *AdaBoost* proposto por (VIOLA; JONES, 2001).

Apesar de o classificador final ser extremamente rápido, o tempo de treinamento pode ser extremamente demorado. Isso ocorre devido à grande quantidade de hipóteses de características que necessitam ser avaliadas a cada estágio. O último bloco, que integra todos os outros blocos discutidos, corresponde a criação da cascata de classificadores. Essa cascata pode ser entendida como uma versão degenerada de uma arvore de decisão. Através dela é possível se reduzir ainda mais o custo computacional necessário na detecção de um objeto de interesse.

A hipótese para sua aplicação baseia-se no fato de somente uma pequena parte da imagem corresponde ao objeto a ser detectado (VIOLA; JONES, 2004), ou seja, grande parte da imagem são janelas negativas. Dessa maneira, a cascata é projetada para rejeitar de forma rápida, já nos primeiros estágios, qualquer janela de detecção que não possua

Entrada: n exemplos de treinamento $(x_1, y_1), \ldots (x_i, y_i) \ldots, (x_n, y_n)$ com rótulos positivos, $y_i = 1$, ou negativos, $y_i = 0$.

1. Inicializa pesos $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$, onde m e l são o número de exemplos positivos e negativos, respectivamente.
2. **Para** $t = \{1, \ldots, T(hipoteses)\}$:
   (a) Normalize os pesos, de tal forma que w_t seja um distribuição de probabilidade
   $$w_{t,i} = \frac{w_{i,t}}{\sum_{j=1}^{N} w_{t,j}}$$
   (b) Para cada *feature* j, treine um classificador h_j, restrito a usar somente uma *feature*. Erro com respeito a w_t : $e_j = \sum_i w_i |h_j(x_i) - y_i|$
   (c) Escolha o classificador h_t, com o menor erro e_t.
   (d) Atualize os pesos:
   $$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}, \quad \beta_t = \frac{e_t}{1 - e_t}$$
   onde $e_i = 0$ se o exemplo x_i classificado corretamente, $e_i = 1$ caso contrário.
3. O classificador forte final é
   $$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{caso contrário} \end{cases}, \quad \alpha_t = \log \frac{1}{\beta_t}$$

Figura 24: Algoritmo *Adaboost*. Adaptado de: (VIOLA; JONES, 2004).

semelhança com o objeto de interesse.

Sua construção é realizada, de tal forma, a despender menor esforço computacional nos primeiros estágios de classificação, aumentado o grau de complexidade à medida que se percorre os estágios mais profundos. O número de estágios e o tamanho de cada um deve ser suficiente para atingir um determinado desempenho de detecção, enquanto minimiza o custo computacional (VIOLA; JONES, 2004).

Os classificadores são treinados por meio de uma estratégia baseada no algoritmo *Ada-boost* combinado com outro método de ajuste dos *thresholds* e número de classificadores de cada estágio, como descrito no algoritmo da Figura 25. Além disso, esse processo é semelhante ao de treinamento de árvores de decisão, em que o classificador seguinte é treinado utilizando os exemplos que passaram por todos os estágios anteriores (VIOLA; JONES, 2004).

Os *thresholds* são ajustados em função da taxa de detecção e taxa de falsos positivos em um dado conjunto de validação. Para cada um destes parâmetros é pré-determinado um valor alvo global, obtido ao se percorrer todos os estágios de classificação. Esses valores globais são dados pela Equação 2.15, taxa de detecção global, e Equação 2.14, taxa de falsos positivos global.

$$F = \prod_{i=1}^{K} f_i \tag{2.14}$$

$$D = \prod_{i=1}^{K} d_i \tag{2.15}$$

Onde:

> $\boldsymbol{F}$ e $\boldsymbol{D}$ são, respectivamente, a taxa de falsos positivos e taxa de detecção da cascata de classificadores.
>
> $\boldsymbol{k}$ é o número de classificadores.
>
> $\boldsymbol{f_i}$ e $\boldsymbol{d_i}$ são, respectivamente, a taxa de falsos positivos e taxa de detecção, nos exemplos apresentados, do i-ésimo classificador.

Como exemplo, considere a criação de uma cascata de classificadores, com 10 estágios, que atinja uma taxa de detecção global de 90%. Utilizando-se a Equação 2.14, tem-se que cada estágio precisa atingir uma taxa de detecção de 99% (0.9 é aproximadamente 0.99^{10}).

Entrada:

N: Exemplos Negativos.

P: Exemplos Positivos.

V: Conjunto exemplos de Validação.

$\boldsymbol{f}$: Taxa máxima de falsos positivos aceita por estágio.

$\boldsymbol{d}$: Taxa mínima de detecção aceita por estágio.

$\boldsymbol{F_{alvo}}$: Taxa global para falsos positivos.

Variaveis:

$\boldsymbol{i}$: número de estágios.

$\boldsymbol{n_i}$: número *features* do estagio i.

$\boldsymbol{F_i}$, $\boldsymbol{D_i}$: taxa de falsos positivos e detecção, respectivamente, da cascata de classificadores composta por i estágios.

Inicalização:

$F_0 = 0; D_0 = 0; i = 0$

Enquanto $F_i > F_{alvo}$:

i++; $n_i = 0; F_i = F_{i-1}$

Enquanto $F_i > f \times F_{i-1}$:

$n_i = n_i + 1$

• Treine (*Adaboost*) um classificador com n_i *features*, utilizando P e N.

• Determine F_i e D_i, avaliando a cascata atual em V.

Decremente o *threshold* do classificador do estagio i

Até que a cascata atual tenha $D_i \geq d \times D_{i-1}$

$N = 0$

Se $F_i > F_{alvo}$:

• Avalie a cascata atual em N e coloque qualquer detecção falsa no conjunto N

Figura 25: Algoritmo de criação da cascata de classificadores . Adaptado de: (VIOLA; JONES, 2004).

Enquanto esse valor parece absurdo, para se atingir um baixo valor global de falsos positivos, Equação 2.15, é necessário que em cada estágio tenha uma taxa de falsos positivos de apenas 30% (0.3^{10} é aproximadamente 6×10^{-6}). Atingir este valor é não difícil pois existe uma troca entre a taxa de detecção e falsos positivos, ou seja, valores altos de taxa de detecção levam a valores baixos de falsos positivos.

O funcionamento da cascata de classificadores pode ser observado na Figura 26. Um resultado positivo para o primeiro classificador, que supere o valor de *threshold* encontrado durante o treinamento, dispara o segundo estágio, que por sua vez dispara o terceiro e assim por diante. Caso passe por todos estágios, janela de detecção é classificada como contendo o objeto de interesse. Cada estagio é formado por um classificador forte, composto pela combinação linear de classificadores fracos.

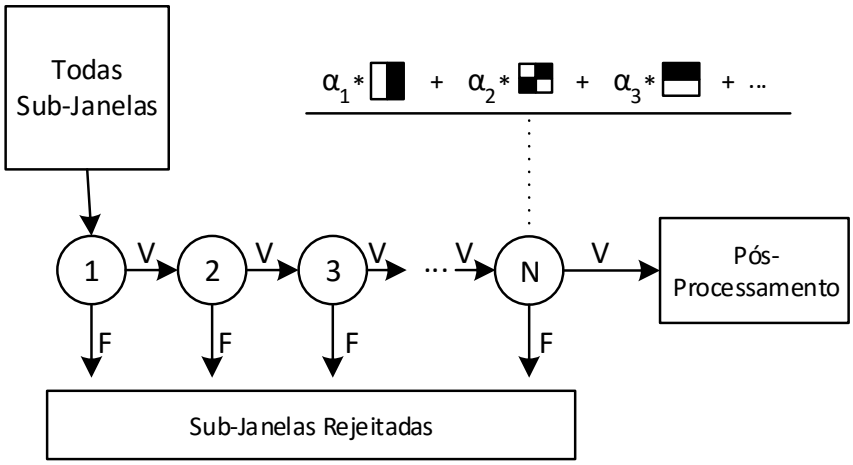


Figura 26: Cascata de classificadores proposto por (VIOLA; JONES, 2001). Sub-janelas negativas (N) são rejeitadas, enquanto que positivas (V) passam para o próximo estagio.

Após o treinamento a cascata de classificadores é utilizada na classificação de janelas de detecção, em um esquema de janela deslizante. O detector final examina a imagem em múltiplas escalas e localizações. Para a análise em várias escalas é realizado o redimensionamento do detector em diversos tamanhos. Isso é aplicável, pois, as características do detector podem ser analisadas em qualquer escala com o mesmo custo computacional. Em (VIOLA; JONES, 2004) obteve bons resultados para um fator de escala de 1,25.

O detector também percorre a imagem em várias localizações. Localizações subsequentes são obtidas pelo deslocamento de uma janela em um determinado número de *pixels*. O número de *pixels* de deslocamento afeta tanto a velocidade do detector quanto a acurácia. No entanto, como o detector é treinado com uma certa variabilidade de translações, uma

boa taxa de detecção é obtida a despeito de pequenos deslocamentos na imagem.

Uma vez que o detector final é invariante a pequenas mudanças em escala e localização, várias detecções irão ocorrer ao redor de um mesmo objeto de interesse, assim como falsos positivos. Dessa maneira, é necessário se retornar apenas uma detecção por objeto. Para tanto, é necessário o pós-processamento das sub-janelas detectadas, combinando detecções sobreposta em uma única detecção final.

Uma maneira simples de agrupamento é a divisão das detecções em dois subconjuntos disjuntos. Isso é feito através da verificação da sobreposição entre duas detecções, onde detecções que possuam sobreposição entre si pertencerão a um mesmo conjunto. Por fim, os vértices, da caixa delimitadora, de cada detecção final, corresponderão a média entre todos os vértices presente em um determinado conjunto.

Várias melhorias foram propostas desde o surgimento do *framework* Viola & Jones. Dentre elas pode-se citar a extensão dos tipos básicos de características *Haar-like* proposto por Lienhart e Maydt (2002), Figura 27. Nessa extensão introduziu-se o conceito de características *Haar-like* inclinadas a um ângulo de 45 graus. A inclusão dessas novas características teve o intuito de melhorar os resultados de detecção e diminuir a taxa de erros.

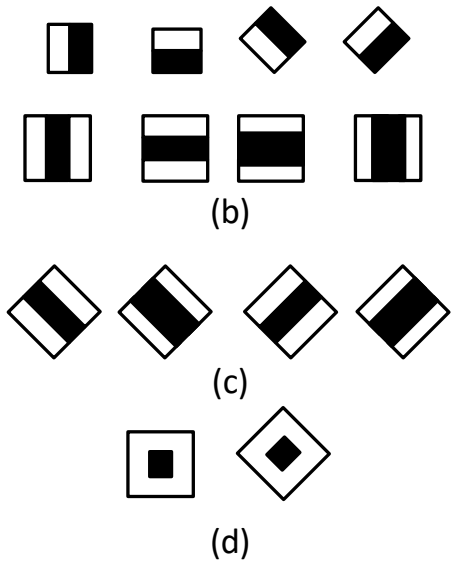


Figura 27: Características *haar-like* extendidas (LIENHART; MAYDT, 2002), onde: (a) são de borda,(b) de linha,(c) de ponto.

Além disso, foram sugeridas outras variações do algoritmo AdaBoost para o treinamento, apresentando melhores resultados, tais como: GentleBoost (VUKADINOVIC; PANTIC, 2005) , Real Adaboost (WU et al., 2004) , entre outras. Também podem ser utilizados outros tipos de características, para uso como classificador fraco, substituindo a Haar-like. Zhu et al. (2006) utilizou características HOG, descritas na Subseção 2.6.1, em conjunto com o framework Viola & Jones para a solução do problema de detecção de pedestre e obteve desempenho superior a trabalhos anteriores. Outro tipo de característica é conhecido como MB-LBP (*Multi-Block Local Binary Pattern*) (GE; WEN; FANG, 2011) e será apresentada na Subseção 2.5.2.

2.5.2 Multi-Block Local Binary Pattern

Em 1994 foi proposto por Ojala, Pietikainen e Harwood (1994) o Padrão Binário Local, conhecido como LBP (*Local Binary Pattern*), com o intuito de descrever texturas em superfícies bidimensionais em imagens. O LBP é um tipo de um descritor visual utilizado em visão computacional nas tarefas de reconhecimento e detecção e é baseado em duas medidas complementares conhecidas como padrões espaciais locais e contraste de escala de cinza.

O descritor em questão tem como finalidade rotular cada um dos *pixels* de uma imagem, em escala de cinza, por meio da comparação de um *pixel* central com sua vizinhança. Essa comparação é realizada com uma vizinhança 3×3 *pixels* e consiste em avaliar se as intensidades dos *pixels* das vizinhanças são maiores ou menores do que a intensidade do *pixel* central. Para isso, caso o *pixel* da vizinhança seja menor do que o *pixel* central é atribuído o valor 1; caso contrário é atribuído o valor 0.

O descritor final é formado pela concatenação dos valores dos oito vizinhos, 0s ou 1s, formando um código binário. Este também pode ser descrito por um número decimal, que é dado pela soma das multiplicações dos valores 0 ou 1, de cada vizinho, por pesos, como visto na Figura 28. Com oito *pixels*, em torno do *pixel* central, é possível representar 256 (2^8) valores para o LBP. Na Figura 28 é apresentado um exemplo do cálculo de um Padrão binário Local (LBP).

Apesar de provado a efetividade do LBP na representação de uma imagem, as características extraídas não permitem uma robustez por serem consideradas muito locais. Para superar as limitações do descritor original, (LIAO et al., 2007) propôs uma extensão conhecido como Padrões Binários Locais de Múltiplas Escala em Blocos (MB-LBP - *Multi-Scale Local Binary Pattern*), aplicando-a ao problema de reconhecimento de faces.

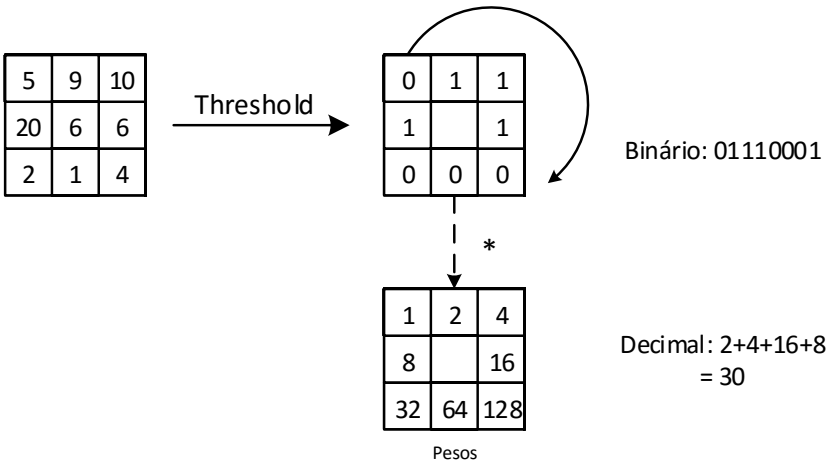


Figura 28: Exemplo de cálculo do descritor LBP. Baseado em (OJALA; PIETIKAINEN; HARWOOD, 1994)

A MB-LBP, diferentemente de Ojala, Pietikainen e Harwood (1994), define a vizinhança como sendo uma região retangular de tamanho variável, dado pela escala $S \times S$, composta por mais de um *pixel*. A Figura 29 apresenta uma vizinhança retangular, de um descritor MB-LBP, onde cada vizinho apresenta 9 *pixels*.

As regiões retangulares, incluindo a região central, são resumidas através da média aritmética de seus *pixels*. Dado a média da intensidade do retângulo central g_c e a média da intensidade sua vizinhança retangular $\{g_1, \ldots, g_{p-1}\}$, a saída do operador MB-LBP, como uma sequência binaria, pode ser descrita pela Equação 2.16.

$$MB - LBP = \sum_{p=1}^{p-1} s(g_p - g_c)2^p, \qquad s(x) = \begin{cases} 1 & , x \geq 0 \\ 0 & , x < 0 \end{cases} \tag{2.16}$$

Onde:

s é uma função sinal.

p é o índice da vizinhança.

Da mesma forma que as características Haar-like, descritas na Subseção 2.5.1, o descritor MB-LBP pode ser calculado de forma rápida e eficiente através de uma imagem integral (ZHANG et al., 2007). Quando comparadas as características *Haar-like*, aplicadas no contexto do framework Viola e Jones (2004), permitem a aceleração do tempo de

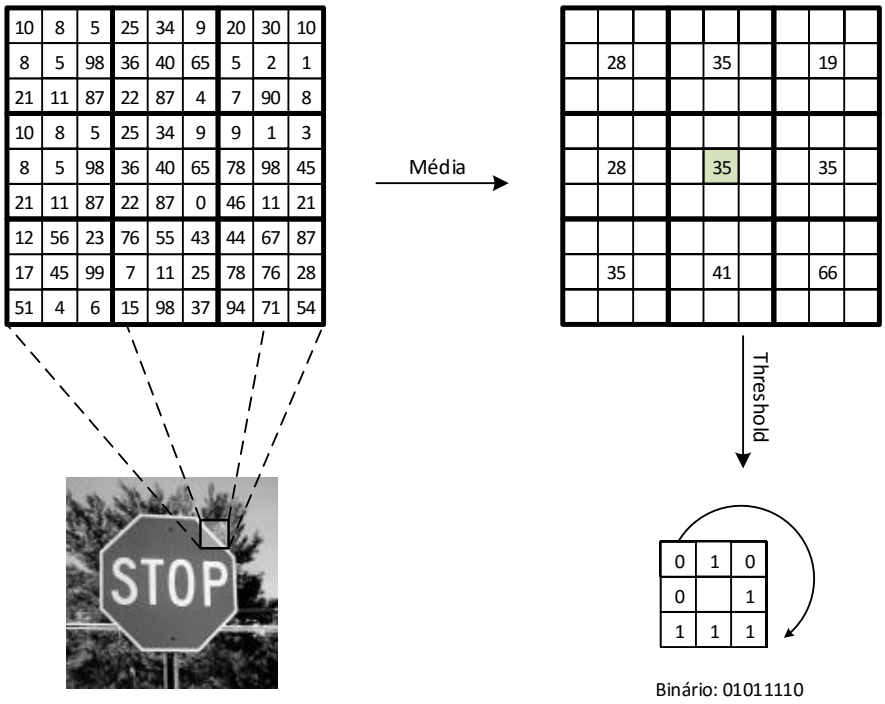


Figura 29: Calculo do MB-LBP para uma vizinhança 3×3.

treinamento, podendo atingir patamares de acurácia semelhantes.

Isso ocorre, pois, o número de combinações de um MB-LBP é aproximadamente 1/20 do número de *Haar-like* para uma sub-janela de tamanho 20×20. A diminuição do número de combinações permite reduzir o tempo do processo de seleção do algoritmo *AdaBoost*, impactando diretamente no tempo de treinamento.

Além disso, o MB-LBP captura mais informações, o que o torna um descritor mais distintivo quando comparado a outros. Também fornece uma melhor representação de uma imagem em relação a proposição original dos Padrões Locais Binários, uma vez que codifica simultaneamente macro e micro estruturas.

Apesar destas vantagens, o MB-LBP não pode ser aplicada diretamente no framework Subseção 2.5.1, pois, seus valores são não-métricos. Sendo assim, é impossível a utilização de um classificador fraco baseado em uma função de *threshold*, assim como utilizado nas características *Haar-like*. De forma a solucionar essa limitação Zhang et al. (2007) adota como classificador fraco, para cada característica MB-LBP, uma árvore de múltiplos ramos. Essa árvore tem o total 256 ramos, e cada ramo corresponde a um determinado valor

discreto de características MB-LBP. O classificador fraco é definido como na Equação 2.17.

$$f_m(x) = \begin{cases} a_0, \ x^k = 0 \\ a_j, \ x^k = j \\ \dots \\ a_{255}, \ x^k = 255 \end{cases} \tag{2.17}$$

Onde:

$\boldsymbol{x^k}$ corresponde ao k-ésimo elemento do vetor de característica x.

$\boldsymbol{a_j}$, com $j = \{0, \dots, 256\}$, são parâmetros de regressão a serem apreendidos.

Esses classificadores fracos são comumente conhecidos como árvores de decisão ou regressão. O melhor classificador fraco, baseado em árvore, pode ser encontrado da mesma forma que se é aprendido um nó em uma árvore de decisão.

2.6 Reconhecimento

2.6.1 Histograma de Gradientes Orientados

O Histograma de Gradientes Orientado ou *Histogram of Oriented Gradients* (HOG) é um descritor de característica utilizado em diferentes problemas de visão computacional e processamento de imagens. Foi proposto inicialmente por Dalal e Triggs (2005) para a solução do problema de detecção de pedestres e posteriormente aplicado a vários tipos de problemas.

A ideia principal é a contabilização do número de ocorrências de orientações de gradientes em determinadas partes da imagem. O intuito do descritor de característica é generalizar a forma e aparência do objeto de tal maneira que objetos semelhantes produzam, sob condições diferentes, vetores de características também semelhantes (DALAL; TRIGGS, 2005).

Isso acontece, pois, o HOG utiliza características globais para descrever um determinado objeto em contraste a outras técnicas que descrevem um objeto como sendo uma coleção de características locais. De maneira simples, isso significa, que uma pessoa inteira, por exemplo, é representada por um único vetor de características em oposição a vários vetores de características representando partes menores de uma pessoa.

O método é similar a outras propostas tais como histogramas de orientações de bordas, descritores SIFT (*Scale-invariant feature transform*) (LOWE, 1999) e contexto de forma (BELONGIE; MALIK; PUZICHA, 2002). Seu diferencial está no uso de um *grid* denso de células uniformemente espaçadas para o cálculo e o emprego de uma normalização de contraste local sobreposta, o que melhora sua precisão.

O descritor possui invariância a transformações geométricas e fotométricas, exceto orientação do objeto. No entanto não apresenta robustez a oclusões e para se adquirir uma invariância a escala é necessário o treinamento do classificador com descritores extraídos em várias escalas para um mesmo objeto (STALLKAMP et al., 2012).

O processo para gerar o descritor é dividido em 4 etapas que tem sua sequência resumidas no Figura 30 e podem ser acompanhadas de forma visual na Figura 31. As etapas são: cálculo de gradientes, votação ponderada, normalização de contraste sob blocos espacialmente sobrepostos e obtenção do descritor.

Na etapa inicial é calculado o gradiente, Figura 31b, para cada *pixel* da imagem de entrada em relação a sua vizinhança. Esse cálculo é realizado através da aplicação de

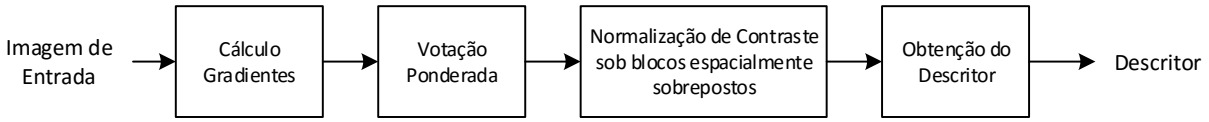


Figura 30: Passos da extração de um descritor HOG.

máscaras de filtragem. Em seu trabalho Dalal e Triggs (2005) foram testados diversos tipos máscaras, sendo que as mascaras unidimensionais centradas foram as que obtiveram os melhores resultados. Outros tipos de filtros também foram testados por Gritti et al. (2008), porém não apresentaram melhoras significativas.

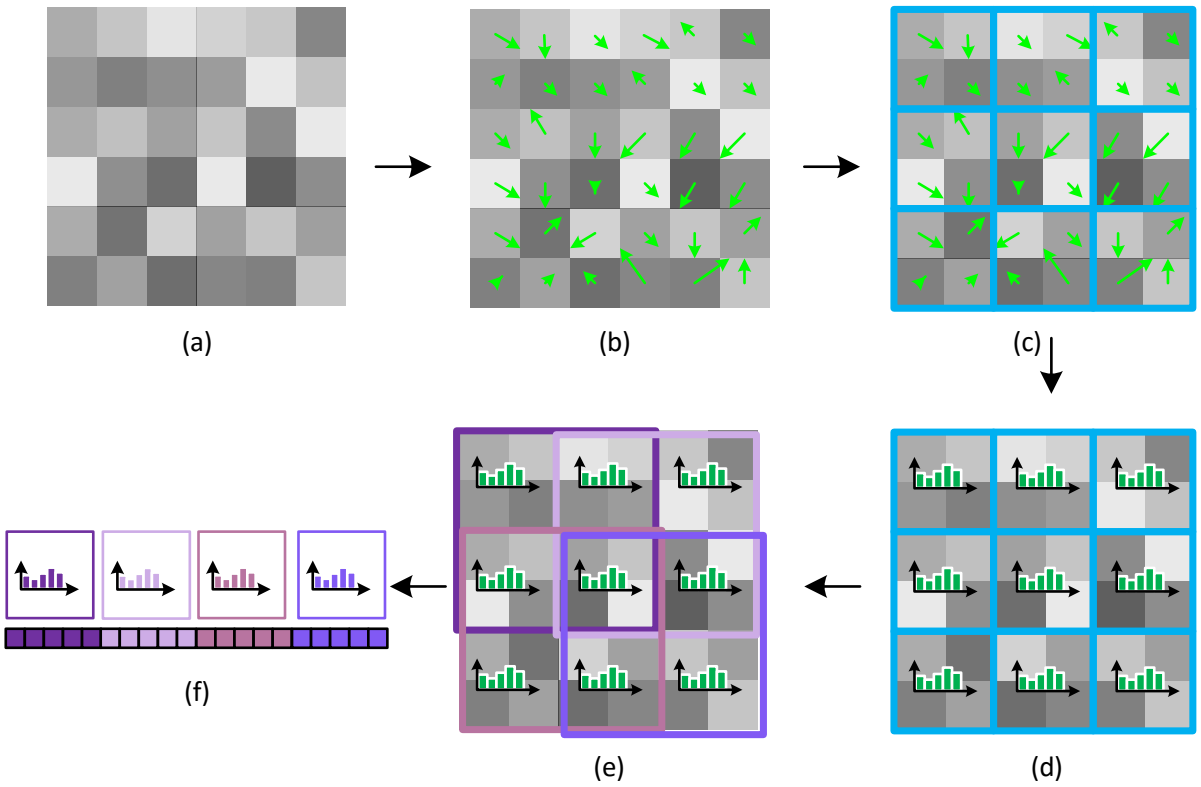


Figura 31: Ilustração das etapas de extração do HOG: (a) imagem de entrada; (b) cálculo do orientação e magnitude do gradiente; (c) agrupamento de *pixels* em células; (d) cálculo do histograma de cada célula; (e) agrupamento de células em blocos sobrepostos e normalização; (f) obtenção do descritor através da concatenação dos histogramas da etapa (e).

Dessa maneira, o cálculo é realizado através de máscaras centradas na direção vertical e horizontal, Figura 32a, como pode ser observado no exemplo da Figura 32b. Os valores resultantes, derivativo horizontal e vertical, são utilizados para o cálculo da orientação e magnitude do gradiente de cada pixel, respectivamente, através da Equação 2.18 e Equação 2.19. No caso de imagens coloridas para cada pixel são armazenados os valores

correspondentes ao canal de maior magnitude.

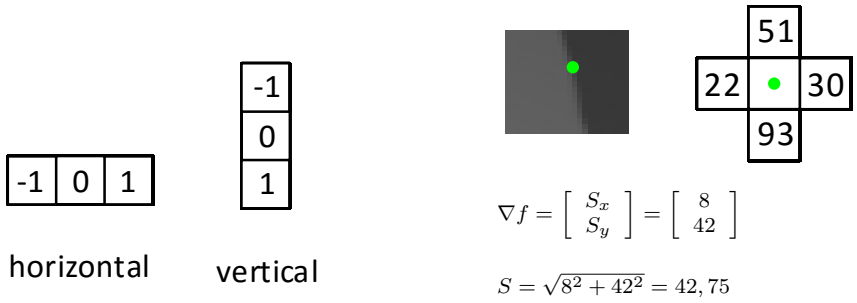
$$\theta = \arctan\left(\frac{s_y}{s_x}\right) \tag{2.18}$$

$$S = \sqrt{S_x^2 + S_y^2} \tag{2.19}$$

Onde:

$\boldsymbol{S_x}$ é o resultado da aplicação da máscara horizontal, direção x.

$\boldsymbol{S_y}$ é o resultado da aplicação da máscara vertical, direção y.



(a) Máscaras centrada. (b) Cálculo da magnitude de um *pixel*.

Figura 32: Cálculo do gradiente.

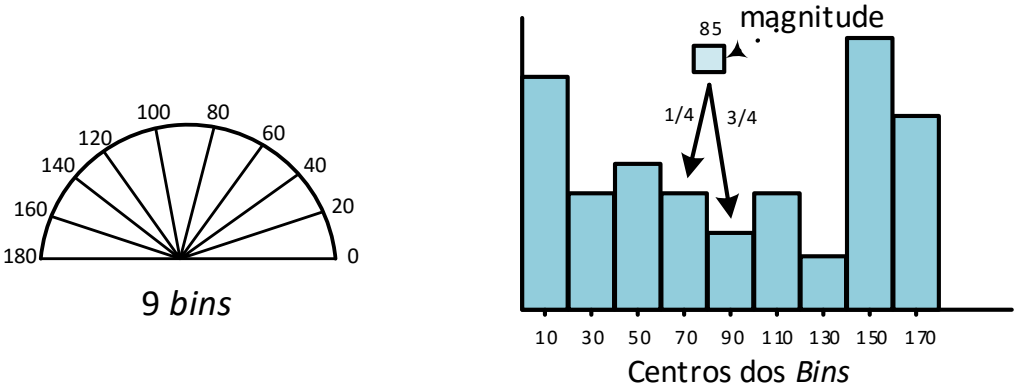
Em seguida os *pixels* são agrupados em células contendo $P \times P$ *pixels*, Figura 31c. Para cada célula é construído um histograma através do processo de votação, Figura 31d. Cada voto é uma função da magnitude do gradiente em cada pixel. O histograma resultante representa a distribuições das orientações do gradiente em cada célula e o voto é dado pela magnitude do gradiente de cada *pixel*.

As orientações são quantificadas em N divisões (conhecidos também como centros ou *bins*) com ângulos de 0 a 180 graus (gradiente não sinalizado) ou de 0 a 360 graus (gradiente sinalizado), como pode observado na Figura 33a. Em seu trabalho Dalal e Triggs (2005) obteve os melhores resultados para 9 divisões e gradiente sinalizado. O voto ocorre numa desta N divisões e para redução do serrilhamento (*aliasing*) os votos são interpolados de forma bi-linear entre os centros vizinhos em ambas orientações e posições.

Por exemplo, suponha uma divisão das orientações em 9 centros de 0 a 180 graus e a votação de um *pixel* com orientação de 85 graus. Esse ângulo não é representado pelos 9

centros, sendo necessário assim a interpolação da magnitude de seu gradiente. Os centros mais próximos são 70 e 90 graus, como pode ser observado na Figura 33b. A distancias para cada um dos centros, respectivamente, são 15 e 5 graus. Dessa maneira a magnitude será distribuída, de forma ponderada, entre ambos os centros, 70 e 90 graus, através da seguinte proporção:

- $\frac{5}{20}$ ou $\frac{1}{4}$ do valor de magnitude para o centro de 70 graus
- $\frac{15}{20}$ ou $\frac{3}{4}$ do valor de magnitude para o centro de 90 graus



(a) Divisão da orientação do gradiente em *bins*.

(b) Exemplo de interpolação linear no processo de votação.

Figura 33: Votação ponderada.

As células são agrupadas em blocos contendo $C \times C$ células e cada bloco possui sobreposição de 50% com os blocos vizinhos, como elucidado na Figura 34. O deslocamento entre blocos é conhecido como *stride*. Em seguida os histogramas de cada bloco são concatenados e normalizados, Figura 31e. A introdução repetida de células em blocos diferentes e a aplicação da normalização permite atenuar problemas de variações locais de iluminação, por conta de sombras ou outras fontes de iluminação, e de contraste.

No método original foram testados vários tipos de normalizações sendo que *L2-Hys*, *L2-Norm* e *L1-norm* melhoraram consideravelmente os resultados de detecção de pedestres. A norma *L2-Hys* corresponde a norma *L2-Norm* seguido de corte do valor máximo de v para 0.2 e renormalização, como descrito em Lowe (2004). A *L2-Norm* é descrita pela Equação 2.20. Nela cada componente de um vetor de característica é dividido pelo valor de normalização.

$$L2\text{-}norm = \frac{v}{\sqrt{\|v_2^2\| + e^2}} \tag{2.20}$$

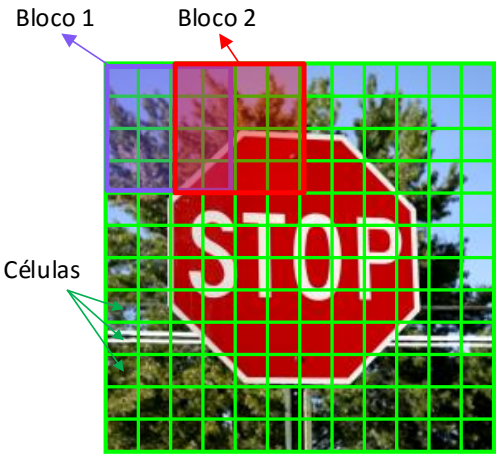


Figura 34: Disposição de blocos e células em uma imagem.

Onde:

$\boldsymbol{v}$ é o descritor de características.

$\|\boldsymbol{v_k}\|$ é a k-norma para $k = 1, 2$.

$\boldsymbol{e}$ é uma constante muito pequena.

Por fim a última etapa, obtenção do descritor Figura 31f, é concatenar os histogramas de cada bloco em um único vetor de característica unidimensional. Este vetor será utilizado para descrever as características de um dado objeto em uma imagem e em conjunto de uma técnica de classificação é possível realizar a tarefa de reconhecimento.

Apesar do HOG descrever a imagem de forma holística muitos padrões presentes nesse descritor são repetidos dentro de uma classe de objeto. Além disso, os vetores de característica HOG podem ser muitos grandes e esse tipo de redundância pouco contribui para melhora da classificação, além de aumentar o tempo de treinamento. No Subseção 2.6.2 será discutido uma maneira eficiente de reduzir a dimensionalidade do vetor de característica HOG e consequentemente o tempo necessário para o treinamento.

2.6.2 Analise de Discriminante Linear

Análise Discriminante Linear, conhecido por LDA (*Linear Discriminant Analysis*), é uma generalização do Discriminante Linear de Fisher (FISHER, 1936). Utilizado em estatística, reconhecimento de padrões e aprendizagem de máquinas tem como objetivo

encontrar uma combinação linear de características que separe duas ou mais classes de objetos ou eventos. Além disso, possui usos práticos como classificador.

Formulado em 1936 por Ronald A. Fisher, o trabalho original foi descrito para um problema de 2 classes. Posteriormente foi generalizado para um número N de classes ficando conhecido como Analise Discriminate Multipla (*Multiple Disciminant Analyisis*), Analise Disciminante Multipla Classes (*Multi-class Linear Discriminant*) ou simplesmente LDA, devido sua popularidade (RAO, 1948).

Esta técnica é comumente utilizada para a redução de dimensionalidade na etapa de pré-processamento em problemas de reconhecimento de padrões e aprendizagem de máquinas (YU; YANG, 2001). A ideia principal desta técnica consiste em se projetar o conjunto de dados em um espaço dimensional menor de tal forma a se obter uma boa separabilidade entre classes, reduzindo assim o *overfitting* (CAO et al., 2003) e custo computacional.

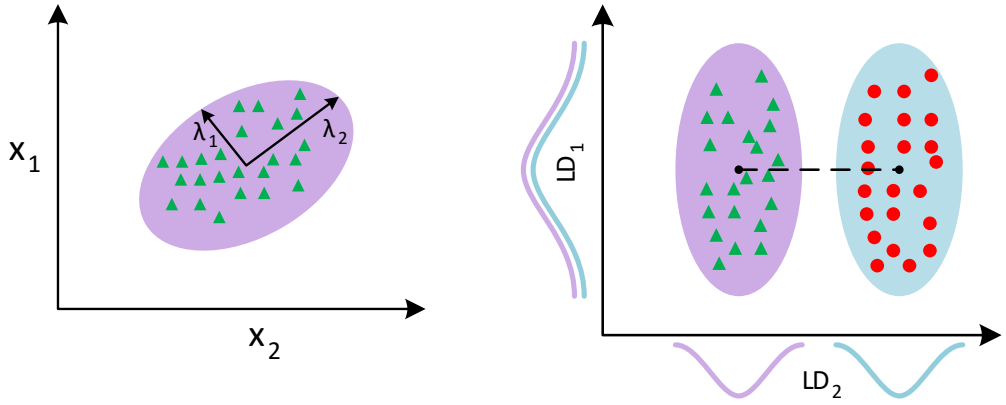
A abordagem da LDA é muito similar a Análise de Componentes Principais ou PCA (*Principal Component Analysis*) (F.R.S, 1901) no que diz respeito em encontrar a combinação de variáveis que melhor explica os dados. O PCA pode ser descrito com uma técnica não supervisionada, uma vez que ignora os rótulos das classes e seu objetivo é encontrar as direções, conhecidas como componentes principais, que maximiza a variância dentro de um conjunto de dados.

Em contraste ao PCA, o LDA pode ser considerado uma técnica supervisionada, já que leva em consideração os rótulos das classes, e calcula as direções, conhecido como Discriminantes Lineares, que irão representar os eixos que maximizam a separação entre múltiplas classes. A Figura 35 elucida essa diferença.

Embora a princípio pareça que o LDA é superior ao PCA, em problemas de multi-classes no qual as classes são conhecidas, isto nem sempre é verdade. Martinez e Kak (2001) demonstrou que o PCA tende a superar o LDA quando o número de amostra por classe é relativamente pequeno.

Deve-se mencionar que o LDA considera uma distribuição normal do conjunto de dados, além de características que são estatisticamente independentes. No entanto, de acordo com Li, Zhu e Ogihara (2006), estas restrições são somente aplicadas para o uso do LDA como classificador sendo que para redução de dimensionalidade pode-se viola-las sem grandes perdas.

O problema do LDA pode ser escrito como a maximização dos pesos w de uma função



(a) PCA - $\lambda_1 e \lambda_2$ maximizam a variância.

(b) LDA - LD_2 é um bom discriminante linear, diferentemente de LD_1.

Figura 35: Diferença entre o LDA e o PCA.

objetivo J, Equação 2.21. A solução para esse para esse problema, como será visto a frente, se baseia na decomposição generalizada de autovetores e autovalores.

$$J(w) = \frac{w^t S_B w}{w^t S_W w} \tag{2.21}$$

A Figura 36 descreve a sequência dos principais passos para o cálculo do LDA. São eles: Cálculo dos vetores de média, Cálculo das Matrizes Dispersas, Decomposição em autovetores e autovalores, Seleção dos Discriminantes Lineares e projeção das amostras no subespaço.

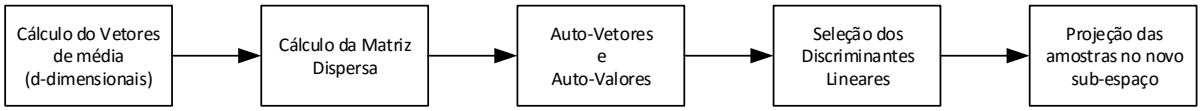


Figura 36: Etapas do cálculo da LDA.

Considere uma base de dados contendo vetores de características (amostras) d-dimensionais compreendendo um número C de classes. O conjunto de vetores de características podem ser expressados por uma matriz X, onde cada linha representa uma amostra e cada coluna determinada uma característica, ou dimensão, desta amostra. Os rótulos das classes de cada amostra são expressos por um vetor y, cuja linhas estão associadas as respectivas

amostras de X. Essa representação pode ser visualizada na Equação 2.22.

$$X = \begin{bmatrix} x_{1_1} & x_{1_2} & x_{1_3} & x_{1_4} & \dots & x_{1_d} \\ x_{2_1} & x_{2_2} & x_{2_3} & x_{2_4} & \dots & x_{2_d} \\ x_{3_1} & x_{3_2} & x_{3_3} & x_{3_4} & \dots & x_{3_d} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n_1} & x_{n_2} & x_{n_3} & x_{n_4} & \dots & x_{n_d} \end{bmatrix}, \qquad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \tag{2.22}$$

Onde:

n corresponde a n-ésimo amostra.

d corresponde a d-ésima dimensão do vetor de característica.

y corresponde ao vetor contendo as classes, que variam de 0 a C-1, para cada uma das amostras.

O primeiro passo do LDA é o cálculo do vetor de médias de cada classe, m_i, utilizando-se a matriz X. A média é calcula com relação a cada coluna da matriz X, através da Equação 2.23, resultando em um vetor m_i, como pode ser visto na Equação 2.23. Para cada classe haverá um vetor de médias, m_i. Após o calculo do vetor de média de cada uma das classes é calculado o vetor média global, m', através da Equação 2.24.

$$m_i = \frac{1}{n_i} \sum_{x \in E_i}^{n} x_k = \begin{bmatrix} m_{i_1} \\ m_{i_2} \\ m_{i_3} \\ \vdots \\ m_{i_d} \end{bmatrix}, \qquad \text{com } i = 1, 2, \dots, C-1 \tag{2.23}$$

$$m' = \frac{1}{C} \sum_{i=0}^{C-1} m_i = \begin{bmatrix} m'_1 \\ m'_2 \\ m'_3 \\ \vdots \\ m'_d \end{bmatrix}, \qquad \text{com } i = 1, 2, \dots, C-1 \tag{2.24}$$

Onde:

$\boldsymbol{m_i}$ é o vetor de média, das amostras, da i-ésima classe.

$\boldsymbol{m'}$ é o vetor média global de todas amostras.

$\boldsymbol{C}$ corresponde ao número total de classes.

$\boldsymbol{n_i}$ corresponde ao número total de amostra na classe i.

$\boldsymbol{E_i}$ corresponde ao subconjunto de amostras pertencentes a classe i.

$\boldsymbol{x_k}$ é k-ésima dimensão de um determinado vetor de característica.

Em seguida são calculadas as matrizes de dispersão intra-classe e inter-classe, respectivamente, S_W e S_B. Ambas matrizes corresponde a estimativas de suas respectivas matrizes de covariância. A matriz de dispersão intra-classe, S_W é calculada pela Equação 2.25. De forma análoga a matriz de dispersão inter-classe, S_B, é calculada pela Equação 2.26.

$$S_W = \sum_{i=0}^{C-1} S_i, \qquad (2.25)$$
$$\text{onde} \quad S_i = \sum_{x \in E_i}^{n} (x - m_i)(x - m_i)^T$$

$$S_B = \sum_{i=0}^{C-1} S_i, N_i(m_i - m')(m_i - m')^T \qquad (2.26)$$

Onde:

$\boldsymbol{N_i}$ é o número de amostra de uma classe i.

De posse de ambas matrizes é, então, solucionado o problema generalizado de autovalores (FISHER, 1936) para a matriz $s_W^{-1} S_B$, Equação 2.27, de tal forma a se obter os discriminantes lineares. Ambos, autovalores e autovetores, fornecem informações sobre a distorção da transformação linear.

$$Av = \lambda v, \qquad A = s_W^{-1} S_B \qquad (2.27)$$

Onde:

$\boldsymbol{v}$ corresponde aos autovetores.

$\boldsymbol{\lambda}$ corresponde aos autovalores.

Os autovetores correspondem a direção desta distorção enquanto que os autovalores descrevem a magnitude da distorção. Para a redução de dimensionalidade os autovetores

são os mais importantes, uma vez que eles representam os novos eixos do subespaço de características. No entanto os respectivos autovalores são particularmente interessantes para a escolha dos novos eixos, já que indicam o quão informativo são os mesmos.

Autovetores que possuem autovalores pequenos contem baixa informação sobre a distribuição dos dados. Esses indicam somente a direção dos novos eixos. Sendo assim, a seleção dos discriminantes lineares é realizada pela ordenação descrescente dos autovalores e respectivos autovetores. Desta lista são escolhidos os k autovetores pertencentes ao topo.

Após a seleção dos novos eixos do subespaço é construído uma matriz W de autovetores de dimensão $k \times d$. Essa matriz irá permitir a redução de dimensionalidade, mapeando um espaço de k-dimensões para um novo de d-dimensões.

A matriz W é construída pela concatenação dos autovetores selecionados. Para se projetar as amostras no novo subespaço é realizado o produto da matriz X pela matriz W, como observado na Equação 2.28.

$$Y = X \times W \tag{2.28}$$

Onde:

$\boldsymbol{X}$ é uma matriz $n \times d$, representando n amostras.

$\boldsymbol{Y}$ são as amostras, de dimensão $n \times k$, transformadas para o novo subespaço.

Através destas etapas são selecionados os discriminantes lineares do LDA. Apesar de sua semelhante com o PCA é possível perceber sua superioridade em problemas onde se conhece as classes. O LDA privilegia a separabilidade das classes em contraste ao PCA que busca maximizar a variância dentro de um conjunto de dados. Sua utilização como método de redução de dimensionalidade permite a redução do custo computacional e aumento da precisão da classificação, através redução do problema de overfitting. Essas caracteristicas são de suma importância para a técnica de aprendizagem discutida n Subseção 2.6.3, no qual as classes são conhecidas.

2.6.3 Máquina de Vetores de Suporte Linear

Em aprendizagem de máquinas, Máquinas de Vetores de Suporte (*Support Vector Machines*) constituem uma técnica de aprendizagem supervisionada utilizada na solução

de problemas de classificação e regressão. Por meio de um conjunto de treinamento rotulado e um algoritmo de aprendizagem é criado um modelo que separa duas categorias de dados. A separação é realizada por meio de um hiperplano com a margem mais larga possível, que separe os exemplos de treinamento.

Esse hiperplano, como pode ser visto na Figura 38, é utilizado posteriormente para a atribuição de uma categoria a novos exemplos. Na classificação ocorre o mapeamento desses exemplos para o novo espaço, encontrado durante o treinamento. Sua categorização é realizada com relação ao lado do hiperplano que se apresenta. Dessa maneira as SVMs são consideradas classificadores lineares binário não probabilístico.

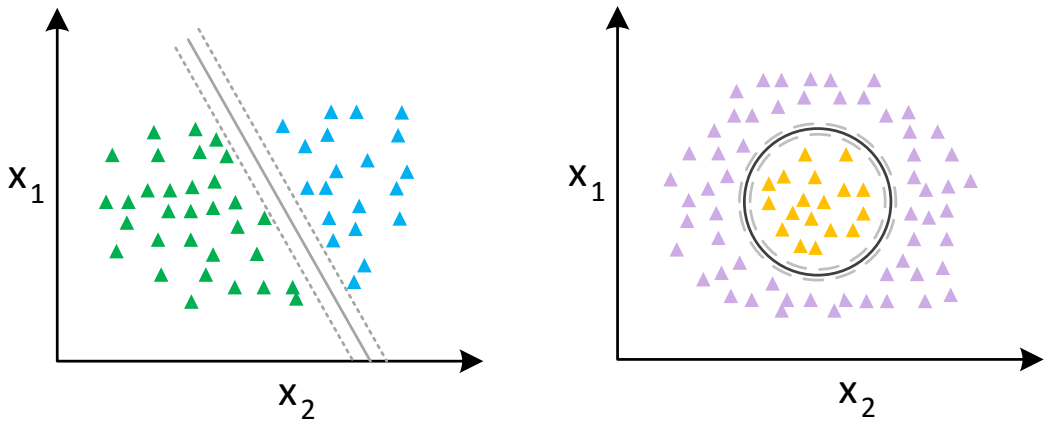
As SVMs podem ser usadas na solução de diversos problemas no mundo real. Os exemplos de aplicações se estendem pelos campos de Visão Computacional, Categorização de Texto, Bioinformática e outras ciências. Em Visão computacional podem se citar exemplos de aplicação bem-sucedidos nas áreas de reconhecimento de faces (OSUNA; FREUND; GIROSIT, 1997), reconhecimento caracteres manuscritos (BAHLMANN; HAASDONK; BURKHARDT, 2002), detecção de pedestres (DALAL; TRIGGS, 2005), entre outras.

Outras técnicas também podem ser utilizadas para o mesmo proposito que a SVM, tais como Redes Neurais Artificiais, Florestas Randômicas e Arvores de Decisão. No entanto, para Lorena e Carvalho (2007) os resultados de sua aplicação muitas vezes são semelhantes ou superiores aos obtidos por outros algoritmos de aprendizagem de máquina, como por exemplo Redes Neurais Artificiais (RNAs ou ANNs) (LORENA; CARVALHO, 2007).

Sua formulação original[1] (VAPNIK; CHERVONENKIS, 2015), conhecida como SVM linear de margens duras (*Hard Margin*), foi desenvolvida por N. Vapnik and Alexey Ya. Chervonenkis, em 1963, para a solução de problemas linearmente separáveis, que podem ser separados por uma reta em um hiperplano. A Figura 37 exemplifica a diferença de problemas lineares e não-lineares através para duas categorias em um plano bidimensional.

Em 1992 Bernhard E. Boser, Isabelle M. Guyon e Vladimir N. Vapnik propuseram uma maneira de se criar um classificador SVM não-linear através do conceito de kernel trick (BOSER; GUYON; VAPNIK, 1992). De forma simples, nesta solução um conjunto de dados linearmente não-separaveis é mapeado para um espaço de características com número de dimensões superior, onde é possível sua separação através de um hiperplano. O padrão atual, SVM linear de margem suave (*soft margin*), foi proposto[2] por Corinna Cortes e Vapnik em 1993 (CORTES; VAPNIK, 1995). Neste trabalho serão tratados apenas

[1] O artigo da formulação original foi publicado em russo e recentemente transcrito para o inglês.
[2] Foi publicado somente em 1995.



(a) Conjunto de dados linearmente separável.

(b) Conjunto de dados linearmente não separável.

Figura 37: Comparativo entre um problema linear e não linear para um conjunto de dados bidimensionais.

o conteúdo referente as SVMs lineares.

Dado um conjunto de treinamento T com n dados $\mathrm{x}_i \in X$ e seus respectivos rótulos $y_i \in Y$, sendo $Y = \{-1, 1\}$. Caso os dados das classes +1 (positivos) e -1 (negativos) sejam separáveis por um plano, T é considerado linearmente separável. Dessa maneira, o classificador gerado a partir de T é considerado linear.

A equação de um hiperplano é dada pela Equação 2.29. Ela divide o espaço em duas regiões que separam exemplos positivos de negativos. Essas regiões podem ser expressadas pela função sinal $g(\mathrm{x})$, Equação 2.30. O termo b presente na Equação 2.29 é conhecido como *bias*. Sem esse termo, o hiperplano sempre está posicionado na origem, dificultando encontrar um hiperplano que maximize o valor de margem.

$$f(x) = \mathrm{w} \cdot \mathrm{x} + b = 0 \tag{2.29}$$

$$g(\mathrm{x}) = sgn(f(\mathrm{x})) = \begin{cases} +1, & \text{se } \mathrm{w} \cdot \mathrm{x} + b > 0 \\ -1, & \text{se } \mathrm{w} \cdot \mathrm{x} + b < 0 \end{cases} \tag{2.30}$$

Onde:

w é o vetor de pesos.

b é o *bias*.

sgn é uma função sinal.

Na Figura 38 é possível se observar como é calculada distancia d entre as margens. Dado um ponto x_1 no hiperplano H_1 e outro ponto x_2 no hiperplano H_2, a projeção de $x_1 - x_2$ na direção do vetor w, perpendicular ao hiperplano H_0, corresponde a distância d. Através de geometria chega-se ao valor de margem com distancia d, Equação 2.31.

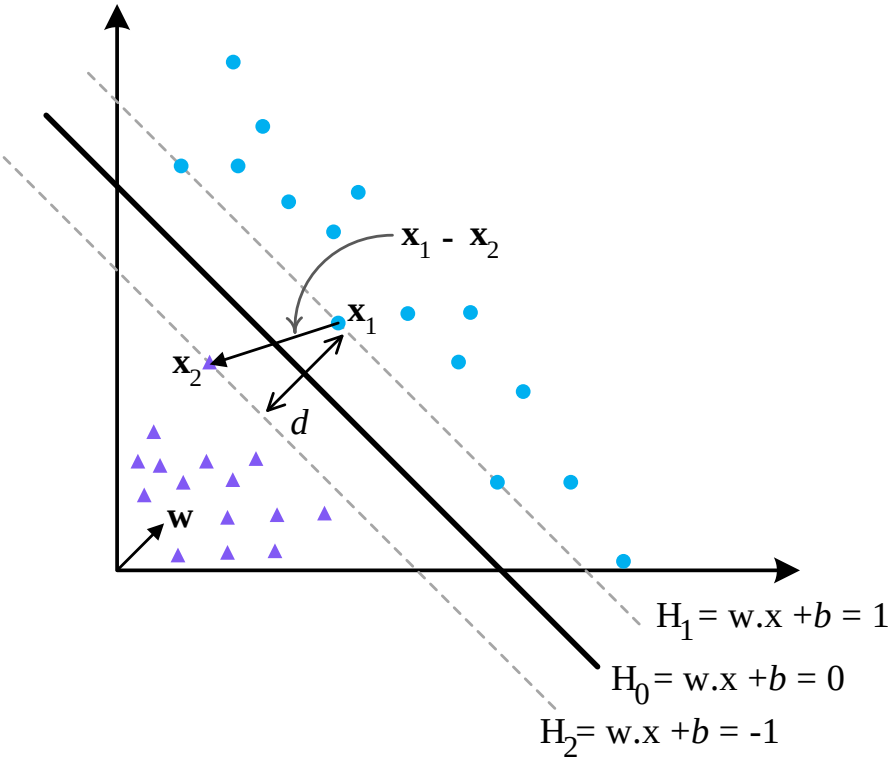


Figura 38: Cálculo da distância entre os hiperplanos H_1 e H_2.

$$d = \frac{2}{\|\mathrm{w}\|} \tag{2.31}$$

Onde:

$\|\mathbf{w}\|$ é a norma do vetor w.

O intuito da SVM é maximizar a distância d. Por conveniência matemática, o problema de maximização da margem de separação dos dados em relação ao hiperplano H_0, pode ser reescrito como um problema de otimização com restrições. Dessa maneira tem-se como objetivo a minimização da Equação 2.32, conhecida como função objetivo.

$$\underset{\mathrm{w},b}{\text{Minimizar}} \quad \frac{1}{2}\|\mathrm{w}\|^2 \tag{2.32}$$

$$\text{Com as restrições: } y_i(\mathrm{w} \cdot \mathrm{x}_i + b) - 1 \geq 0,\ \forall i = 1, \ldots, n$$

Onde:

$\boldsymbol{y_i}$ é o rótulo do exemplo i.

$\mathbf{w}$ são os parâmetros (pesos) a serem encontrados.

$\boldsymbol{b}$ é o *bias*.

$\boldsymbol{n}$ é o número de exemplos de treinamento.

$\mathbf{x_i}$ corresponde ao exemplo i (vetor de características).

A reescrita da Equação 2.32 é realizada para exprimir o problema inicial em um problema de programação quadrática, cuja soluções de otimização são amplamente conhecidas na matemática. Esse tipo de problema pode ser solucionado pela introdução de uma função Lagrangiana, associada aos multiplicadores de Lagrange, α_i. A Equação 2.33 demonstra a reescrita da Equação 2.32.

$$L(\mathrm{w}, b, \alpha) = \frac{1}{2}\|\mathrm{w}\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(\mathrm{w} \cdot \mathrm{x}_i + b) - 1) \tag{2.33}$$

Onde:

$\boldsymbol{\alpha_i}$ são multiplicadores de Lagrange.

Através da Equação 2.33 chega-se ao problema de otimização apresentado na Equação 2.34, também conhecida como forma dual. O resultado final é o classificador $g(x)$, Equação 2.35, onde α_i^* e b^* são as soluções encontradas para a forma dual.

$$\begin{aligned} &\underset{\alpha}{\text{Maximizar}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathrm{x}_i \cdot \mathrm{x}_j) \\ &\text{Com as restrições:} \begin{cases} \alpha_i \geq 0, \, \forall i = 1, \ldots, n \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \end{cases} \end{aligned} \tag{2.34}$$

$$g(\mathrm{x}) = sgn(f(\mathrm{x})) = sgn\left(\sum_{x_i \in SV} y_i \alpha_i^* \mathrm{x}_i \cdot \mathrm{x} + b^* \right) \tag{2.35}$$

Onde:

SV são vetores de suporte, encontrados como solução.

A formulação de SVM anterior,Equação 2.32, é conhecida como SVM linear de margens dura (*hard margin*). Em aplicações reais é difícil encontrar dados que sejam linearmente separáveis. Isso acontece pois nem sempre os dados se encontram imunes a ruídos e *outliers*. Dessa maneira a formulação original (*hard margin*) é estendida para lidar com problemas que possuam essa característica.

Isso é feito pela introdução de variáveis de folgas que relaxam as restrições da Equação 2.35, impostas ao problema de otimização primal, Equação 2.32. Esse procedimento suaviza as margens do classificador linear, permitindo a introdução de alguns dados entre os hiperplanos H_1 e H_2, assim como, alguns erros de classificação. As restrições se transformam, então, nas Equação 2.36.

$$y_i(\mathrm{w} \cdot \mathrm{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \ldots, n \tag{2.36}$$

Onde:

ξ_i é uma variável de folga.

Essa nova restrição suaviza as margens, permitindo que alguns dos dados permaneçam entre os hiperplanos H_1 e H_2, Figura 38. Essa abordagem dá origem as SVMs com margens suaves (*soft margin*s). A função objetivo da Equação 2.32 é reformulado como na Equação 2.37.

$$\underset{\mathrm{w},b,\xi}{\text{Minimizar}} \quad \frac{1}{2}\|w\|^2 + C\left(\sum_{i=1}^{n} \xi_i\right) \tag{2.37}$$

Onde:

C é um hiperparâmetro a ser encontrado.

O parâmetro C controla a troca entre os erros da SVM no conjunto de treinamento e a maximização da margem. Quando C tende ao infinito o resultado é a formulação de margens duras (RYCHETSKY, 2001). E da mesma maneira que a formulação anterior, Equação 2.34 é introduzida uma função Lagrangiana e tem-se como resultado o problema

dual da Equação 2.38.

$$\begin{aligned} &\underset{\alpha}{\text{Maximizar}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathrm{x}_i \cdot \mathrm{x}_j) \\ &\text{Com as restrições:} \begin{cases} 0 \leq \alpha_i \leq C, \forall i = 1, \ldots, n \\ \sum_{i=1}^{n} \alpha_i y_i = 0 \end{cases} \end{aligned} \tag{2.38}$$

O classificador final é o mesmo dado pela Equação 2.35, para a SVM de margens duras. Para ambos tipos de SVMs, os pontos x_i, que participam da formação do hiperplano separador, são conhecidos como vetores de suportes e podem ser considerados os dados mais informativos do conjunto de treinamento. Por isso o nome máquina de vetores de suporte.

A criação de um classificador SVM, em termos práticos, é composta de duas fases: treinamento e teste. Na fase de treinamento é realizada a busca da solução do problema quadrático, enquanto que no teste é aferido a precisão do modelo. O treinamento é realizado a partir de exemplos rotulados, positivos e negativos. Esse conjunto de dados é dividido em 2 partes distintas: treinamento e teste. A escolha dos exemplos em cada um destes conjuntos é realizada de forma aleatória e devem conter dados totalmente distintos.

Antes do treinamento, porém, é necessário a normalização dos dados contidos nestes exemplos. O processo consiste na padronização da faixa de valores dos dados, também conhecidos como características. De acordo com Hsu et al. (2003) este pré-processamento é importante pois impede que atributos com uma faixa numérica grande sobreponham atributos com faixas numéricas pequenas. Hsu et al. (2003) recomenda uma normalização linear de cada uma das característica para faixa numérica de $[+1, -1]$ ou $[0, 1]$. Isso pode ser realizada através da Equação 2.39, conhecida como *minmax*, ou através da norma L2, descrita na Subseção 2.6.1.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2.39}$$

Onde:

- $\boldsymbol{x}$ é o valor original.
- $\boldsymbol{x'}$ é o valor normalizado.

Na SVM de margem suave é necessário a escolha do parâmetro C. Essa pode ser feita de forma manual, com base na distribuição dos dados ou de forma automática por meio do processo conhecido como busca em *grid*. Se o valor de C escolhido for muito grande impõe-se uma alta penalidade para pontos não separáveis, ocasionando o armazenamento de muitos vetores de suporte e consequentemente a ocorrência do *overfitting*. Caso seja muito pequeno pode ocorrer um underfitting (ETHEM, 2004).

Assim como grande parte das técnicas de aprendizagem de máquina, a SVM pode sofrer de problemas com relação a sua capacidade de generalização. Tendo em vista isto, é empregado um processo conhecido como validação cruzada (*Cross-Validation*). Ela consiste em um conjunto de técnicas que avaliam a capacidade de generalização de um modelo para predição. Através disso é possível reduzir o problema de *overfitting* e juntamente com a busca em *grid* consegue-se avaliar o desempenho dos hiperparâmetros do modelo (CAWLEY; TALBOT, 2010). No entanto, para sua aplicação é necessário a divisão do conjunto de treinamento em duas partes: treinamento e validação.

Um dos tipos de validação cruzada empregado é conhecido como k-dobras (k-folds) (HSU et al., 2003). Este método consiste em se particionar o conjunto de treinamento em K partes iguais. Dessas K partes uma é retida como dados de validação para o teste do modelo e as outras $K-1$ partes são utilizadas como dados de treinamento. O processo de validação cruzada é, então, realizada K vezes (número de dobras), onde cada uma das k partes é utilizada apenas uma vez como dados de validação. Os resultados de teste das K dobras são utilizados para o cálculo da média, que produz uma estimativa única do desempenho do modelo.

A Figura 39 ilustra o funcionamento da técnica k-dobras. Neste exemplo, o conjunto de dados (roxo escuro) é dividido, aleatoriamente, em 5 conjuntos disjuntos, que são utilizados para treinar 5 modelos, utilizando uma parte para validação (amarelo) e o restante para treino (roxo claro). A acurácia de cada modelo é extraída do conjunto de teste (vermelho).

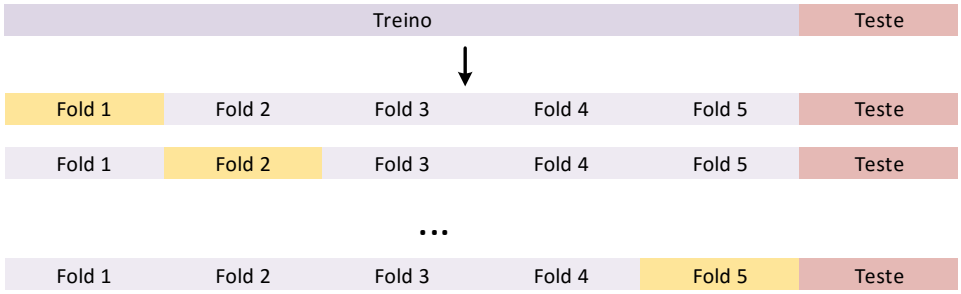


Figura 39: Validação cruzada *k-fold*, com k igual a 5.

Em sua concepção a SVM é um classificador binário. Porém nem sempre os problemas de classificação consistem na separação de duas classes. Com o intuito de superar essa limitação foram propostas estratégias e até mesmo outras formulações de SVM (HSU; LIN, 2002). Uma maneira de se treinar um modelo para várias classes é a divisão de um modelo único em vários classificadores binários. Existem duas estratégias de treinamento conhecidas: um contra todos (*one versus all*) e um contra um (*one versus one*).

A primeira estratégia de treinamento corresponde a treinar N classificadores binários, um por classe, com os exemplos da classe como positivos e todos outros exemplos como negativos. Por outro lado, a estratégia um contra um envolve a construção de um classificador para cada par de classes, resultando em $\frac{N(N-1)}{2}$ classificadores (ANTHONY; GREGG; TSHILIDZI, 2007).

Como visto a SVM é um classificador não probabilístico linear que possui formulações matemáticas para solução de problemas não-linearmente e linearmente separáveis. A formulação que trata de problemas linearmente separáveis é conhecida como SVM linear, possuindo dois formatos: margem duras e margens suaves.

O processo de treinamento é supervisionado e consiste na otimização de uma função objetivo. Além disso, são necessários exemplos positivos e negativos, sendo estes normalmente divididos em três conjuntos: treinamento, validação e teste. Os dados contidos nestes exemplos devem ser normalizados para a padronização da faixa de valores de cada característica.

O conjunto de validação é utilizado juntamente com o de treinamento em um processo conhecido validação cruzada. Através dele é possível se estimar os parâmetros da SVM, assim como reduzir o problema de *overfitting*.

Em problemas onde é necessário a criação de um modelo para classificação de múltiplas classes é possível o treinamento de múltiplos classificadores binários através das estratégias um-contra-todos e um-contra-um.

Por fim o conjunto de dados de teste é utilizado para se aferir o desempenho do classificador gerado. Na Seção 2.8 serão discutidas métricas de avaliação de desempenho nas tarefas de reconhecimento e detecção, que podem ser utilizadas tanto para SVM quanto para outras técnicas de aprendizagem de máquina.

2.7 Sistemas Embarcados

Muitos consumidores podem não estar cientes, mas sistemas embarcados já fazem parte de suas vidas a um bom tempo. De um simples controle remoto até o controle de injeção eletrônico de um carro, esses são apenas exemplos de aplicações utilizados corriqueiramente. Além dessas aplicações podem se citar diversas outras, tais como:

- Computadores de bordo automotivos;
- Controle de temperatura de ar-condicionado e geladeiras;
- Impressoras;
- Roteadores e modems;
- Equipamentos médicos, como sistema de eletro-encefalograma e ultra-som;
- Relógios Digitais;
- Celulares;
- Câmeras digitais;

Um sistema embarcado é a combinação de *hardware* e software projetados em conjunto para desempenhar uma tarefa especifica, ao contrário de sistemas de propósito geral (ALMEIDA, 2016). Geralmente possuem uma especificação de *hardware* em conformidade com a tarefa que foram projetados para executar (OTTLEY, 2007).

A sua especificidade é a principal característica que os diferencia de computadores de propósito geral. Um computador *desktop* (PC) é um exemplo claro de um computador de proposito geral. Através dele é possível realizar diversas tarefas tais como, acessar internet, assistir um vídeo, gravar áudio, etc.

As Propriedades típicas de um sistema embarcado quando comparado com um sistema de propósito geral estão relacionadas a suas restrições e limitações, tais como: baixo consumo de energia, tamanho reduzido, baixo custo por unidade e poder computacional limitado. Tais limitações impostas por um sistema embarcado fazem com que seja necessário um conhecimento não somente do software (denominado *firmware*) que se deseja construir, mas também um bom conhecimento sobre o *hardware* para o qual o software será desenvolvido.

No princípio nenhum sistema operacional era utilizado em sistemas embarcados. Todas empresas construíam seus *software*, que comunicavam diretamente com o *hardware*, com praticamente o mínimo de multitarefa e interação com o usuário. Com o passar do tempo sistemas cada vez mais complexos começaram a surgir. Juntamente com isso, o número de funcionalidades que tais sistemas deveriam suportar foi incrementada (RAGHAVAN; LAD; NEELAKANDAN, 2005).

Esses requisitos fizeram com que tornasse necessário um sistema operacional mínimo, que pelo menos tratasse multitarefas, gerenciamento de memória e processos, etc. Dessa forma vários sistemas operacionais surgiram tais como: Microsoft Windows CE®, QNX ® Neutrino, Red Hat® eCos™, Linux embarcado, etc.

O sistema operacional Linux foi criado em 1991 por Linux Torvalds. O *kernel* do Linux[3] é um projeto que segue o modelo de desenvolvimento *open source* (código aberto). Ele está sob a licença GNU GPL (*General Public License*), que permite o estudo, utilização, modificação e distribuição livremente por qualquer pessoa, de acordo com os termos da licença. Essa característica permitiu que seu código fosse portado para as mais diversas arquiteturas, incluindo os presentes em sistemas embarcados.

Por conta de diversos fatores econômicos e desafios técnicos houve uma forte adoção de Linux em sistemas embarcados em vários segmentos de mercado. Para Hallinan (2007) os motivos para tal adoção e atual crescimento, se dá pelas seguintes características do sistema operacional Linux:

- Suporte a uma variedade de dispositivos de *hardware*.
- Suporte a uma imensa quantidade de aplicações e protocolos de rede.
- O Linux pode ser implantado sem o pagamento de *royalties*, que muitos sistemas operacionais embarcados exigem.
- É escalável, podendo ser aplicado tanto em pequenos sistemas orientados a consumidores até sistemas grandes como roteadores e *switches* de operadoras. O mesmo *kernel* executa em celulares, relógios, roteadores, servidores, etc.
- A atração de uma grande quantidade de desenvolvedores ativos permite um suporte rápido a arquiteturas, plataformas e dispositivos.
- Um número crescente de vendedores de *hardware* e software com suporte ao Linux.

[3] *Kernel* Linux: https://www.kernel.org/

O funcionamento do Linux embarcado pode ser abstraído através de camadas, o que facilita o entendimento de seus principais componentes. Sua arquitetura básica é ilustrada na Figura 40.

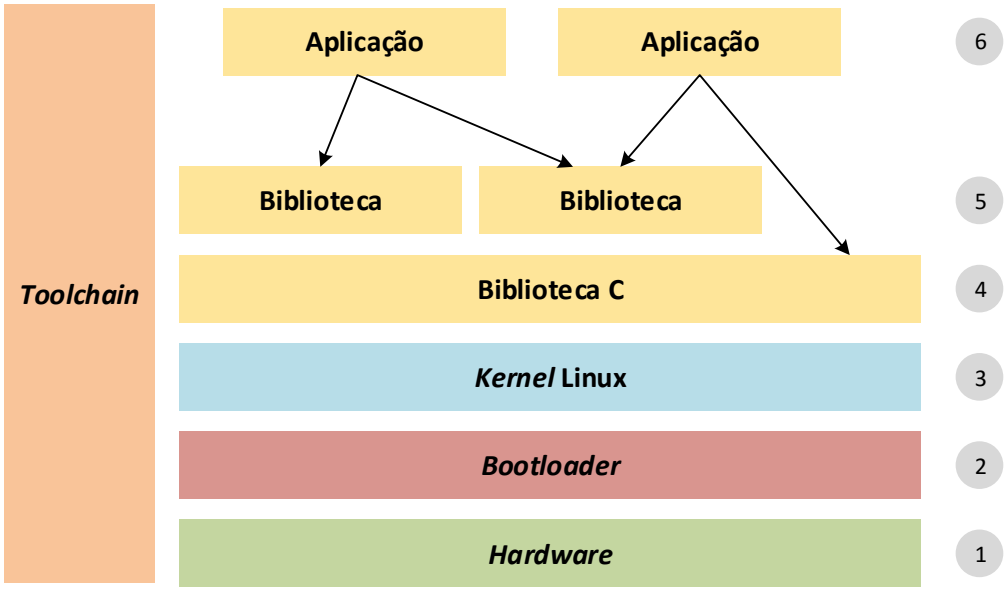


Figura 40: Arquitetura básica do Linux embarcado. FONTE: (PRADO, 2016)

A primeira camada corresponde ao *hardware*, que engloba: CPU, memória RAM e armazenamento, comunicação, entre outros periféricos. O *hadware* inicializa o *bootloader* (segunda camada), que é responsável por realizar a inicialização básica, carregar e executar o *kernel* (terceira camada). Por sua vez o *kernel*, núcleo do sistema, gerencia os recursos de *hardware* (CPU, memória, Entrada/Saída, etc), além de fornece serviços para as aplicações de usuário.

O sistema de arquivos principal, conhecido com *rootfs*, contém diversas bibliotecas. A biblioteca C (quarta camada) prove uma interface entre o *kernel* e as aplicações de usuário, através de chamadas do sistema (*system calls*). Por fim aplicações de usuário (sexta camada) fazem uso tanto de bibliotecas (quinta camada) do sistema, por onde é possível acessar funcionalidades do sistema, quando das Bibliotecas C. Sendo assim, essa arquitetura permite o delineamento de três tarefas durante o desenvolvimento com Linux embarcado:

- Desenvolvimento de BSP (*Board Support Package*).
- Integração.

- Desenvolvimento de aplicações.

A tarefa de desenvolvimento do BSP compreende portar o *kernel*, *bootloader*, *drivers* de dispositivos de *hardware*, etc. Muitas fabricantes de kits de desenvolvimento provem BSPs, com distribuição Linux portada e compilada para a plataforma alvo, permitindo o desenvolvimento direto.

Porém muitas das vezes é necessário a ativação de determinadas funcionalidades de *hardware* (através de configurações do *kernel*), habilitação de periféricos, ou mesmo alteração no processo de inicialização do *bootloader*. Esse procedimento é conhecido como integração e nele são satisfeitas as especificações do sistema embarcado.

Caso o Linux esteja portado e as especificações de *hardware* satisfeitas, resta o desenvolvimento de aplicações Linux. Todas essas tarefas são dependentes da transformação do código fonte em código binário para a arquitetura do *hardware* alvo. Esse processo é conhecido como compilação e é realizado através do conjunto de ferramentas presente na *toolchain*.

Um dos primeiros passos para o desenvolvimento de Linux embarcado consiste na configuração da *toolchain*, necessária tanto para compilação do *kernel* quanto aplicações. A *toolchain* utilizada no desenvolvimento de sistemas embarcados é conhecida como *cross-compiling toolchain* (toolchain de compilação cruzada), composta por um conjunto de ferramentas, binários e bibliotecas.

Esse nome é dado, pois a plataforma de desenvolvimento, conhecida como *Host*, é diferente da plataforma alvo, conhecida com *Target* (RAGHAVAN; LAD; NEELAKANDAN, 2005). Normalmente quando um compilador x86 é utilizado, um código binário para a plataforma x86 é gerado. Porém, é possível, por exemplo, um compilador x86 gerar binários para a arquitetura ARM.

A compilação cruzada é necessária, pois muita das vezes a plataforma alvo não possui os recursos como memória, espaço em disco e poder computacional suficientes para realização do processo de compilação. Outra limitação possível pode ser a inexistência de ferramentas de compilação nativas. Dessa forma a compilação cruzada acontece no *Host*, cujo compilador gera binários para a plataforma alvo (*Target*).

Recentemente diversas opções de *hardware* estão disponíveis a um baixo custo e um considerável poder computacional (SRIRAM; ILLURI, 2014). A Tabela 3 apresenta alguns exemplos de *hardware* e seus respectivos preços.

Tabela 3: Várias placas de desenvolvimento e seus respectivos preços. FONTE: (SRIRAM; ILLURI, 2014)

Placas de Desenvolvimento	**Preço (U$)**
Raspberry pi	35
BeagleBone Black	45
BeagleBoard	125
UDOO	100
Intel MinnowBoard Max	99

A BeagleBone Black (BBB), Figura 41, é uma placa de baixo custo, *hardware* aberto (*open hardware*) e expansível lançada por uma comunidade de desenvolvedores patrocinada pela Texas Instruments (HE; HUANG; WOLTMAN, 2014). Suporta diversos tipos de distribuições Linux tais como Ubuntu[4] e Angstrom[5]. Além disso, possui um tamanho compacto e baixo consumo energético (COLEY, 2013). O fato de ser *hardware* aberto, permite modificações que podem reduzir mais ainda seu tamanho e consumo energético.

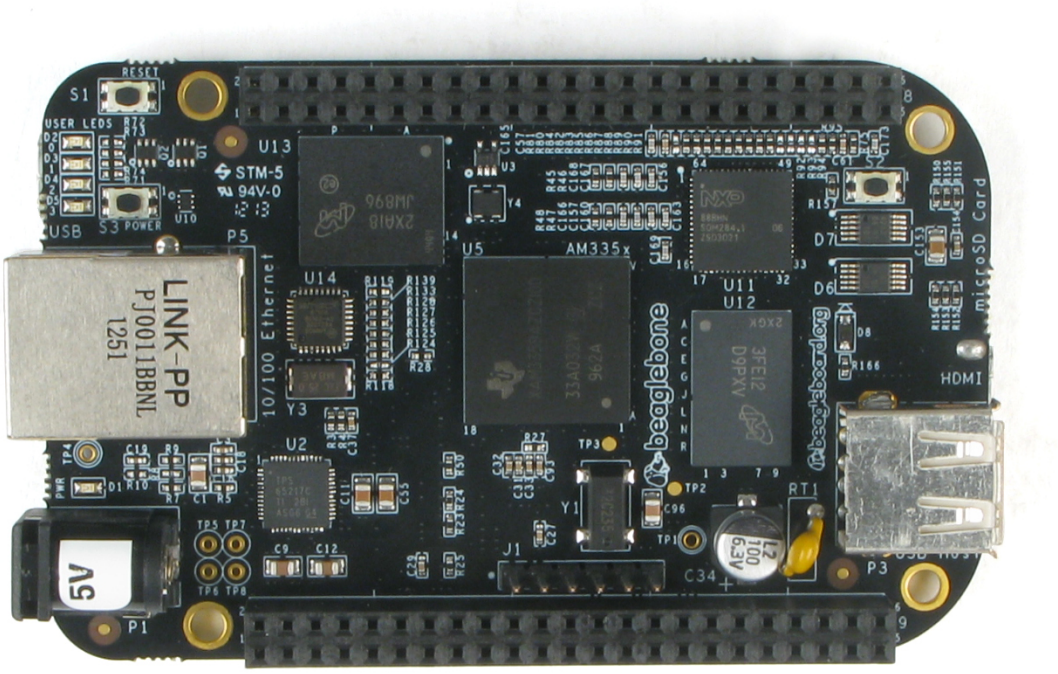


Figura 41: BeagleBone Black.

Sua especificação básica é composta por um poderoso processador de 32 *bits* TI Sitara™ ARM Cortex™-A8 de 1Ghz, 512 MB de memória RAM e 2GB de memória

[4]Ubuntu: http://www.ubuntu.com/
[5]Angstrom: http://www.angstrom-distribution.org/

flash, que serve para hospedar o sistema operacional Linux e outras ferramentas de desenvolvimento de *software* (INSTRUMENTS, 2013). Além disso possui um processador gráfico SGX, suporte a diversos protocolos de comunicação e pinos de GPIO (*General Purpose Input/Output*). Um resumo das funcionalidades é apresentado na Tabela 4.

Tabela 4: Resumo das funcionalidades da BeagleBone Black. FONTE:(HE; HUANG; WOLTMAN, 2014)

Processador	AM3358; 1GHz – Alimentação USB ou DC.
Memória	512 MB DDR3 SDRAM; 2GB eMMC *Flash*.
Conexões	HDMI com audio; USB, 10/100 *Ethernet*; *debug* serial.
Subsistemas	176K ROM; 64K RAM; Motor gráfico 3D; LCD e controladora *touchscreen*; PRU-ICSS. Relógio de tempo real (RTC); 2 Portas USB. *Ethernet*; CAN (*Controller Area Network*); UART (2); McASPs(2); McSPI (2); I2C(2). Conversor analógico-digital; *Enhanced Capture Module* (3); PWM (3); Acelerador de criptografia.

Juntos, *hardware* e *software* abertos formam uma ótima solução para a redução de custo em um sistema de visão computacional embarcado. Além disso a possibilidade de modificações no *hardware* permite uma redução mais drástica de custos, através da eliminação de componentes e redução de tamanho. A *BeagleBone Black* possui estas características, o que a torna uma boa candidata para a integração nesses tipos de sistemas.

2.8 Análise de Desempenho

As áreas de Visão Computacional, Visão de Máquina e Analise de imagens tem suas origens nas pesquisas iniciais de inteligência artificial. Assim, segundo Clark e Clark (1999), três gerações de pesquisadores trabalharam na solução desses problemas. A primeira geração trabalhou com computadores que mal conseguiam manipular imagens digitais, sendo as técnicas desenvolvidas em sua grande maioria baseadas em matemáticas da formação de imagens e analise dos valores de *pixels* em uma região de vizinhança.

A segunda geração de pesquisadores já tinha ao seu dispor computadores um pouco mais potentes, onde podiam processar imagens de forma *online*, visualizá-las e interagir com elas. Grande parte dos algoritmos desenvolvidos nessa época estavam relacionados ao uso de otimização para solução de problemas da área.

Por fim, na terceira geração os computadores se tornaram bastante rápidos e passaram a ter mais espaço de armazenamento de tal forma, que passou a ser praticável o processamento de grandes bases de dados em um tempo hábil. Isso permitiu aos pesquisadores trabalharem em técnicas de visão que aprendem e na avaliação do desempenho desses algoritmos.

Existem poucas ocasiões onde é possível prever o desempenho de um algoritmo, ou sistema de visão computacional, de forma analítica (CLARK; CLARK, 1999). Dessa forma, a maneira mais usual é aferir o desempenho de maneira empírica. Isso é realizado executando-se o programa em um grande conjunto de dados de entrada, em que suas saídas são conhecidas (*ground truth*), e em seguida contando o número de vezes que o programa produz saídas corretas ou incorretas. Nesse teste são possíveis quatro tipos de resultados:

- **Verdadeiro Positivo** (*True Positive*), conhecido como, *true acceptance* ou *true match*, ocorre quando um teste deveria produzir uma saída correta e assim o faz.
- **Verdadeiro Negativo** (*True Negative*), conhecido como, *true rejection* ou *true non-match*, quer dizer que o resultado esperado era incorreto e o teste produziu esse resultado.
- **Falso Positivo** (*False Positive*), conhecido como *false alarm*, ou *type I error*, ocorre quando um teste deveria produzir um resultado correto na verdade produz um resultado incorreto.

- **Falso Negativo** (*False Negative*), conhecido como miss, *false dismissal* ou *type II error*, ocorre quando um teste deveria produzir um resultado incorreto mas produz um resultado correto.

Considere um sistema de visão que deseje detectar a presença ou não de um gato em uma foto. Para testar esse sistema são coletadas imagens, que podem conter ou não o objeto de interesse. A cada imagem está associada uma saída, G para a existência de gato e NG para não existência.

Caso uma imagem de teste contenha um gato (G) e o sistema diga que há um gato (G), tem-se um Verdadeiro Positivo (TP). Caso contrário, se realmente não houver um gato (NG) e o sistema também produzir um resultado dizendo que não há (NG), tem-se um Verdadeiro Negativo (TN).

Se a imagem não tiver um gato (NG) e o sistema disser que há (G), o resultado é um Falso Positivo (FP). Caso contrário, se houver um gato (G) e o sistema responder que não há (NG), tem-se um Falso Negativo (FN). A Tabela 5, conhecida como tabela de contingência, resume esses possíveis casos. Dessa forma, o procedimento de teste consiste em manter o controle destas quatro variáveis.

Tabela 5: Tabela de contingência

		Predição	
		Gato	**Não-Gato**
Real	**Gato**	TP	FN
	Não-Gato	FP	TN

A Tabela 5 representa apenas problemas binários, onde se tem apenas duas classes. A generalização dessa tabela de contingência, para múltiplas classes, é conhecida como matriz confusa (*confusion matrix*). Na Tabela 6 é apresentada uma matriz confusa para um exemplo fictício de reconhecimento de dígitos. Nessa tabela os valores na diagonal representam os números de acertos, TP, para cada uma das classes de dígitos. Já os valores fora das diagonais representam as classificações incorretas. A soma dos valores de cada linha é igual ao total de testes para cada um dos dígitos.

A taxa de erro de um sistema de visão é uma medida de quão bem o sistema resolve o problema para o qual foi projetado (SHAPIRO; STOCKMAN, 2001). A importância dada a cada erro, FN ou FP, depende do tipo de aplicação que está sendo avaliada.

Por exemplo, seja uma aplicação que queira dizer se um paciente tem ou não uma

Tabela 6: Matriz Confusa para um exemplo fictício de reconhecimento de dígitos.

	Predição									
Real	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	20	0	0	6	1	10	0	0	0	0
1	0	21	3	6	8	0	1	1	0	0
2	0	0	27	3	0	10	10	10		10
3	0	0	0	13	0	1	3		0	0
4	0	0	0	0	15	1	1	1	0	0
5	0	0	0	0	2	32	0	0	0	0
6	1	0	2	1	0	0	10	0	0	3
7	0	8	11	1	1	4	1	20	1	
8	5	0	1	1	2	0	12	0	30	
9	0	0	0	1	0	1	0	0	0	16

doença. Se o sistema disser incorretamente o paciente está doente, falso positivo (FP), pode resultar em o paciente tomar remédios sem precisar ou realizar mais testes. Do lado oposto, se o sistema disser incorretamente que o paciente não está doente, falso negativo, pode ter consequências graves. Nesse tipo de sistema talvez seja mais vantajoso reduzir os riscos com o paciente (minimizar FP), aumentando os custos com o paciente (aumento dos FN).

Através do número de TP, TN, FP e FN é possível o cálculo de várias métricas, grande parte originada do campo de recuperação de informação. As mais usuais são: sensitividade/*recall*, especificidade (*specificity*), precisão (*precision*), F1-score e acurácia (POWERS, 2011).

A sensitividade Equação 2.40, conhecida como *recall* ou *true positive rate*, mede a proporção de positivos que foram classificados como positivos. Da mesma forma a especificidade Equação 2.41, ou *true negative rate*, mede a proporção de negativos que foram classificados como negativos.

$$Sensitividade = Recall = \frac{TP}{P} = \frac{TP}{(TP + FN)} \tag{2.40}$$

$$Especificidade = \frac{T}{N} = \frac{TN}{(TN + FP)} \tag{2.41}$$

A interpretação de cada medida varia conforme a área ou contexto da aplicação. Por exemplo, precisão e *Recall* possuem interpretações diferentes no contexto de recuperação de informação. *Recall* é a fração dos documentos, relevantes para busca, que foram recu-

perados com sucesso. Já a precisão Equação 2.42 é a fração dos documentos recuperados que são relevantes para as necessidades do usuário.

$$Precisão = \frac{TP}{(TP + FP)} \tag{2.42}$$

Em aplicações de detecção e localização o objetivo é localizar objetos de interesse de classe O_1 e não muitos outros objetos O_2. A performance de tais sistemas é caracterizada pela sua precisão e *recall*. Tais medidas estão intrinsecamente ligadas, se uma diminui a outra aumenta e vice-versa. A precisão determina se o sistema está detectando muitas coisas que não são o objeto de interesse, quanto mais baixo o valor menos preciso. Já o *recall* diz quantos objetos do total existente foram detectados.

Para ilustrar, considere a imagem da Figura 42 e que o objetivo do sistema seja detectar e localizar estrelas na imagem. Uma caixa tracejada em torno de um objeto representa que ele foi detectado e localizado. Para esse exemplo TP é igual a 2, pois duas estrelas foram detectadas. O número de FP é 2 uma vez que dois hexágonos também foram detectados. Já o FN é igual a 1 pois somente uma estrela foi deixada de ser detectada. Dessa forma tem que o *recall* é $2/(2+1) = 0,66$ e a precisão é $2/(2+2) = 0.5$.

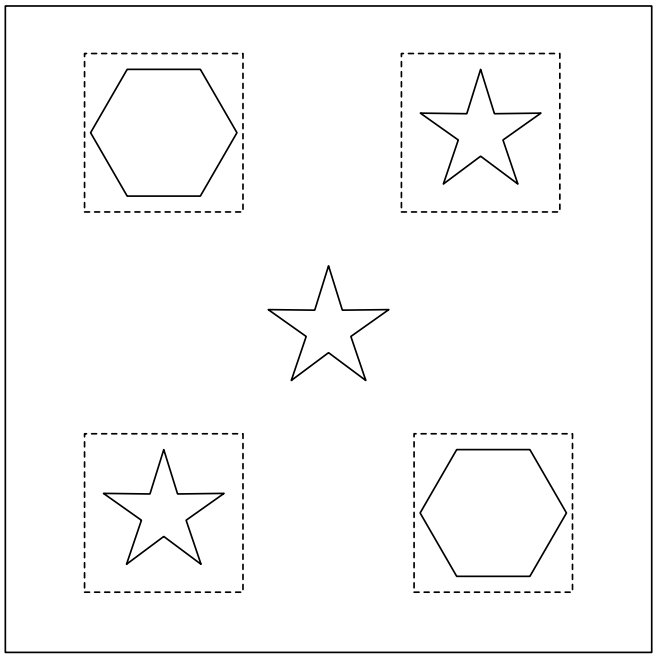


Figura 42: Imagem contendo objetos e resultado de um sistema de detecção, para um exemplo fictício.

O trade-off entre precisão e *recall* pode ser observado através da curva conhecida como

precision-recall (CLARK; CLARK, 1999). Essa curva é obtida variando-se algum parâmetro do sistema e obtendo os valores de precisão e *recall* para cada um destes. Isso permite avaliar o impacto do parâmetro no desempenho e escolher um ponto de operação.

Outra métrica é a acurácia Equação 2.43, que define o número de acertos em relação ao total de testes. Sozinha, a acurácia nem sempre é uma métrica confiável. Em problemas onde existe um desbalanceamento no conjunto de dados de teste, pode acontecer de produzir resultados incorretos.

$$Acurácia = \frac{TP + TN}{P + N} = \frac{TP + TN}{(TP + TN + FP + FN)} \tag{2.43}$$

Por fim, tem-se a métrica *F1-score*, Equação 2.44, que combina a precisão e *recall* em uma média harmônica e é uma medida de acurácia. Ela mede o balanço entre a precisão e *recall* de uma classe e valores próximos de 1 são considerados bons, enquanto que próximos de 0 são ruins.

$$\text{F1-Score} = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.44}$$

Para problemas com múltiplas classes duas métricas, semelhantes a *F1-score*, podem ser calculadas: micro-média (*micro-averaged*) e macro-média (*macro-averaged*). *F-measure*(ÖZGÜR; ÖZGÜR; GÜNGÖR, 2005). Na micro-média, a *F1-score* é calcula de forma global sobre todas as classes. Primeiramente são obtidas todas somatórias de precisão π e *recall* ρ, através da Equação 2.45 e Equação 2.46, respectivamente. Em seguida os valores de π e ρ são usados para o cálculo da micro-média, por meio da Equação 2.47.

$$\pi = \frac{TP}{(TP + FP)} = \frac{\sum_{i=1}^{M} TP_i}{\sum_{i=1}^{M} (TP_i + FP_i)} \tag{2.45}$$

$$\rho = \frac{TP}{(TP + FN)} = \frac{\sum_{i=1}^{M} TP_i}{\sum_{i=1}^{M} (TP_i + FN_i)} \tag{2.46}$$

Onde:

$\boldsymbol{M}$ é o número de classes.

$$F(micro\text{-}averaged) = \frac{2 \times \pi\rho}{(\pi + \rho)} \tag{2.47}$$

Já a macro-média é calculada de forma local para cada uma das categorias e depois feito a média entre todos os valores. A Equação 2.48 resume este procedimento.

$$F(macro\text{-}averaged) = \frac{\sum_{i=1}^{M} F_i}{M}, \quad F_i = \frac{2\pi_i\rho_i}{\pi_i + \rho_i} \tag{2.48}$$

Onde:

$\boldsymbol{\pi_i}$ é a precisão da classe i.

$\boldsymbol{\rho_i}$ é o *recall* da classe i.

Ambas métricas podem ser utilizadas para se ter uma perspectiva geral do desempenho de um classificar em um problema com múltiplas classes. A micro-média representa o desempenho do sistema sobre todo o conjunto de dados, enquanto que a macro-média pode ser utilizada para se medir o desempenho quando se tem classes desbalanceadas (ASCH, 2013).

As métricas e analises apresentadas permitem avaliar quão bom é um sistema de visão computacional em relação a uma base de teste para a metodologia proposta. No Capítulo 4, elas serão utilizadas para se avaliar o método proposto nesse trabalho.

3 Desenvolvimento

Como foi visto, um sistema de visão computacional para identificação de placas de trânsito podem ser divididos em três tarefas básicas. Essa divisão é necessária, pois, o problema é extremamente complexo. Alguns motivos para a complexidade do problema são: quantidade de objetos presentes em uma cena natural, degradação das placas, oclusões parciais, iluminação ruim, etc. Além disso, as dimensões de uma placa são muito pequenas quando comparadas ao tamanho da cena, tornando o problema mais difícil.

A tarefa inicial a ser considerado é a detecção, que tem como intuito reduzir o custo computacional da etapa seguinte. Isso faz com que a etapa de reconhecimento tenha que classificar apenas algumas regiões de interesse, os quais possuem alta probabilidade de conter uma placa.

Alguns sistemas adicionam uma etapa extra conhecida como rastreamento, onde o objeto de interesse reconhecido é rastreado quadro a quadro. Isso torna desnecessário a utilização da etapa de detecção quadro a quadro e reduz assim o custo computacional.

O desenvolvimento deste trabalho se deu em duas fases. Primeiro a metodologia proposta foi implementada e testada em uma máquina *desktop*, e em seguida na placa de desenvolvimento. Isso foi feito pela facilidade em se testar o sistema de visão computacional antes de realizar o *deploy* para o sistema embarcado, já que a configuração do ambiente de desenvolvimento é mais simplificada no *desktop*. Além disso, essa divisão permitiu realizar testes e comparação entre os sistemas.

A primeira etapa para o desenvolvimento do trabalho foi a criação e configuração do ambiente de trabalho. Como estre trabalho trata de sistemas embarcados, esse ambiente *desktop* será tratado como *Host*. Com o intuito de facilitar o desenvolvimento optou-se por utilizar uma máquina virtual como *Host*. Essa escolha se deve ao fato de que durante a configuração do sistema diversos problemas podem ocorrer e uma das vantagens de se trabalhar com maquinas virtuais é a criação de *snapshots*, que permitem reverter um estado passado da máquina virtual. Essa característica é extremamente importe em um

ambiente de desenvolvimento, onde constantemente realiza-se testes.

Como software de virtualização foi utilizado o *VirtualBox*[1], Figura 43, um projeto *open source* regido pelos termos GNU GPL (*General Public License*) e mantido pela Oracle®. Foi criado, então, uma máquina virtual e instalado a distribuição Ubuntu, um sistema operacional de código aberto construído através do *kernel* Linux.

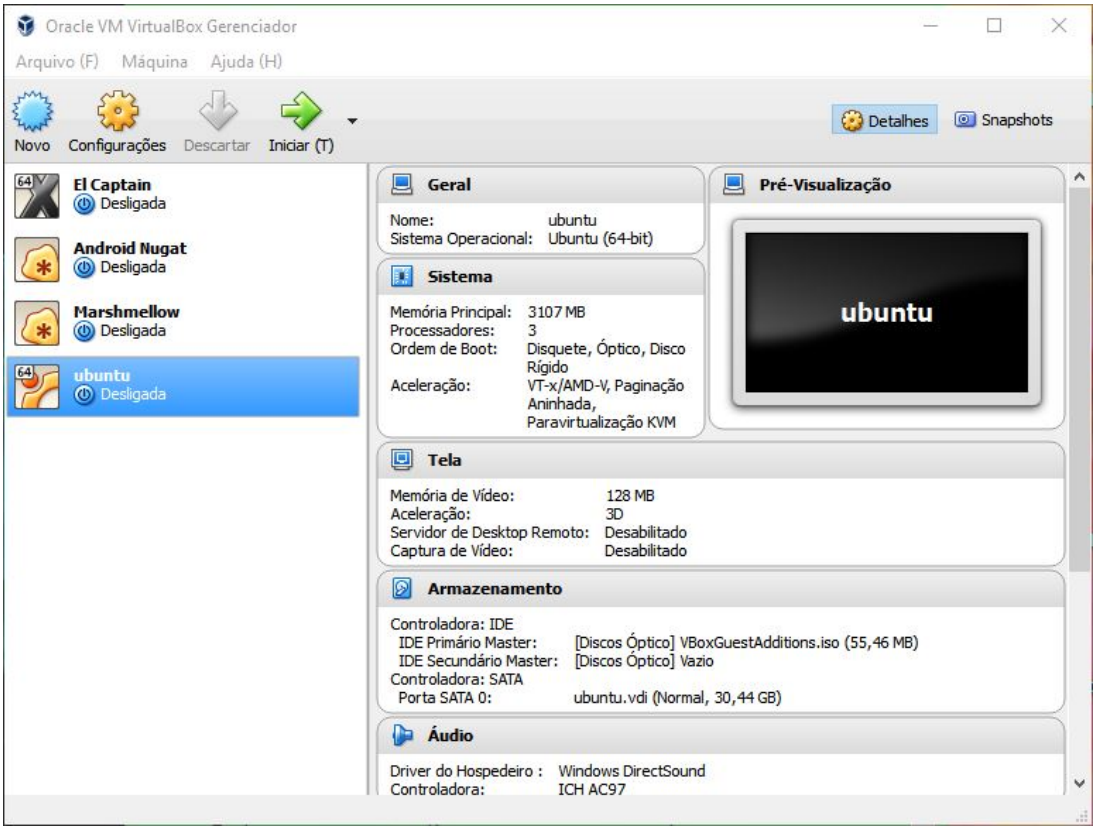


Figura 43: Virtualizador VirtualBox.

Para linguagem de programação optou-se pelo uso do *C++*, que apresenta um desempenho superior a outras linguagens de programação interpretadas, tais como *Python* e *Java*. No entanto, o *C++* não traz produtividade equivalente a linguagem como por exemplo *Python*, que permitem rápida prototipação de ideias. Essa escolha foi necessária, já que o desempenho no sistema embarcado é algo de extrema importância e linguagens interpretadas dificilmente são otimizadas.

A fim de facilitar o tratamento de arquivos e manipular eventos escolheu-se o framework Qt, descrito no Apêndice A.2. Esse *framework* fornece classes para se percorrer o sistema de arquivos de forma simples, o que é necessário para a leitura dos exemplos de treinamento, assim como, os modelos gerados. Outra vantagem é o fato de o *framework* ser *cross-plataform*, isso quer dizer que o mesmo código implementado para desktop pode

[1]VirtualBox: https://www.virtualbox.org/

ser usado para se gerar um binário executável para um sistema embarcado ou até mesmo um dispositivo móvel.

Uma maneira de se obter o framework Qt é através da instalação, a qual foi realizado no *Host*, da SDK (*Sofware Development Kit*), composto por uma IDE (*Integrated Development Environment*) e pelo *framework*. O IDE, Figura 44, que acompanha a SDK é conhecido como *Qt Creator*, que inclui um editor, construtor de *layout* e *forms*, e um *debbuger*. Para compilação, o IDE utiliza-se de um compilador *C++*, que para Linux é o GCC (*GNU Compiler Collection*).

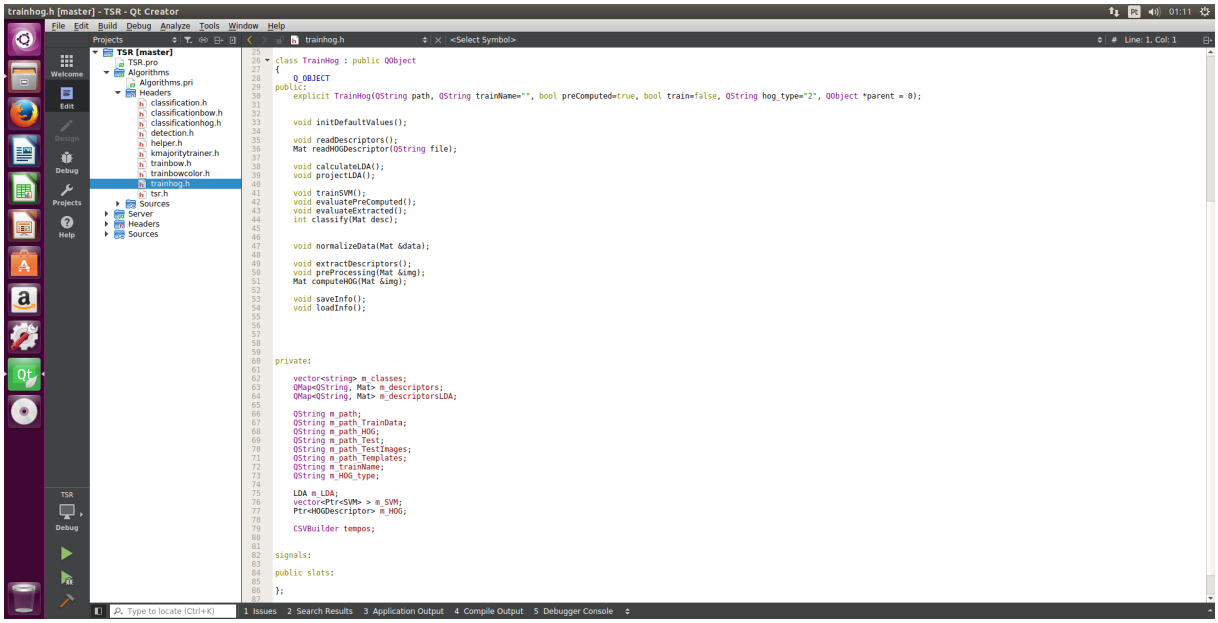


Figura 44: Ambiente de desenvolvimento do *Qt Creator*.

Em conjunto com o *framework* Qt foi instalado a biblioteca OpenCV (*Open Source Computer Vision Library*), que assim como o primeiro é um projeto *open source* e *cross-plataform*. Como pode ser visto no Apêndice A.1, essa biblioteca é composta por módulos, que traz a implementação otimizada de diversos algoritmos utilizados em visão computacional.

Para sua instalação no Ubuntu é necessária, primeiro, a instalação de dependências, através do gerenciador de pacotes *apt-get*, e compilação de seu código fonte, processo que pode levar horas. Assim que terminada, a instalação é feita pelo comando *make install*, que instala os arquivos de cabeçalho (*headers*) e bibliotecas dinâmicas. A etapa final consiste na configuração do *Qt Creator* para que encontre os arquivos *headers* e faça a ligação das chamadas do código fonte com as bibliotecas dinâmicas.

Como é possível perceber, as escolhas se deram principalmente por soluções de código

aberto (*open source*) e *cross-plataform*. Dessa forma o custo total do sistema se reduz apenas ao custo de aquisição do SBC (*Single Board Computer*).

Com o ambiente de desenvolvimento criado, a tarefa de implementação se torna transparente, ou seja, um código para ambas plataformas, *desktop* e embarcado. Nas próximas seções deste capítulo serão descritas as etapas da metodologia implementada. Neste trabalho foram consideradas apenas duas primeiras etapas de um sistema de identificação de placas de trânsito: detecção (Seção 3.2) e reconhecimento (Seção 3.3). Devido às restrições do *hardware* de um sistema embarcado, foi adicionado uma etapa de segmentação (Seção 3.1). Por fim, a última seção (Seção 3.4) trata da segunda fase do desenvolvimento, que corresponde a integração do sistema de visão computacional ao sistema embarcado (*Target*).

3.1 Segmentação

O processo de segmentação tem como intuito isolar na imagem as placas de trânsito, reduzindo, assim, o espado de busca para a etapa de detecção. Devido a forma como são construídas as placas possuem cores que as discriminam de uma grande parte dos objetos em uma imagem. Sendo assim a cor é uma característica a ser explorada no processo de segmentação. Neste trabalho este procedimento foi aplicado somente na fase de teste.

O método consiste em se manipular a imagem original de tal forma que se obtenha uma máscara, utilizada para selecionar objetos que possuam as cores características das placas de trânsito. O processo completo pode ser acompanhado no fluxograma da Figura 45.

Inicialmente são criados duas copias da imagem original. Uma das cópias é convertida para a escala de cinza (Figura 45 - Passo 1) e tem seu histograma equalizado para melhora do contraste (Figura 45- Passo 2), permitindo assim visualizar detalhes escondidos devido as diversas condições de luminosidade encontradas. Esta imagem resultante será utilizada no final processo em conjunto com a máscara.

Em seguida a segunda cópia passa pelo processo de conversão do espaço de cores (Figura 45- Passo 3). A imagem originalmente se encontra no formato RGB (*Red, Green,*

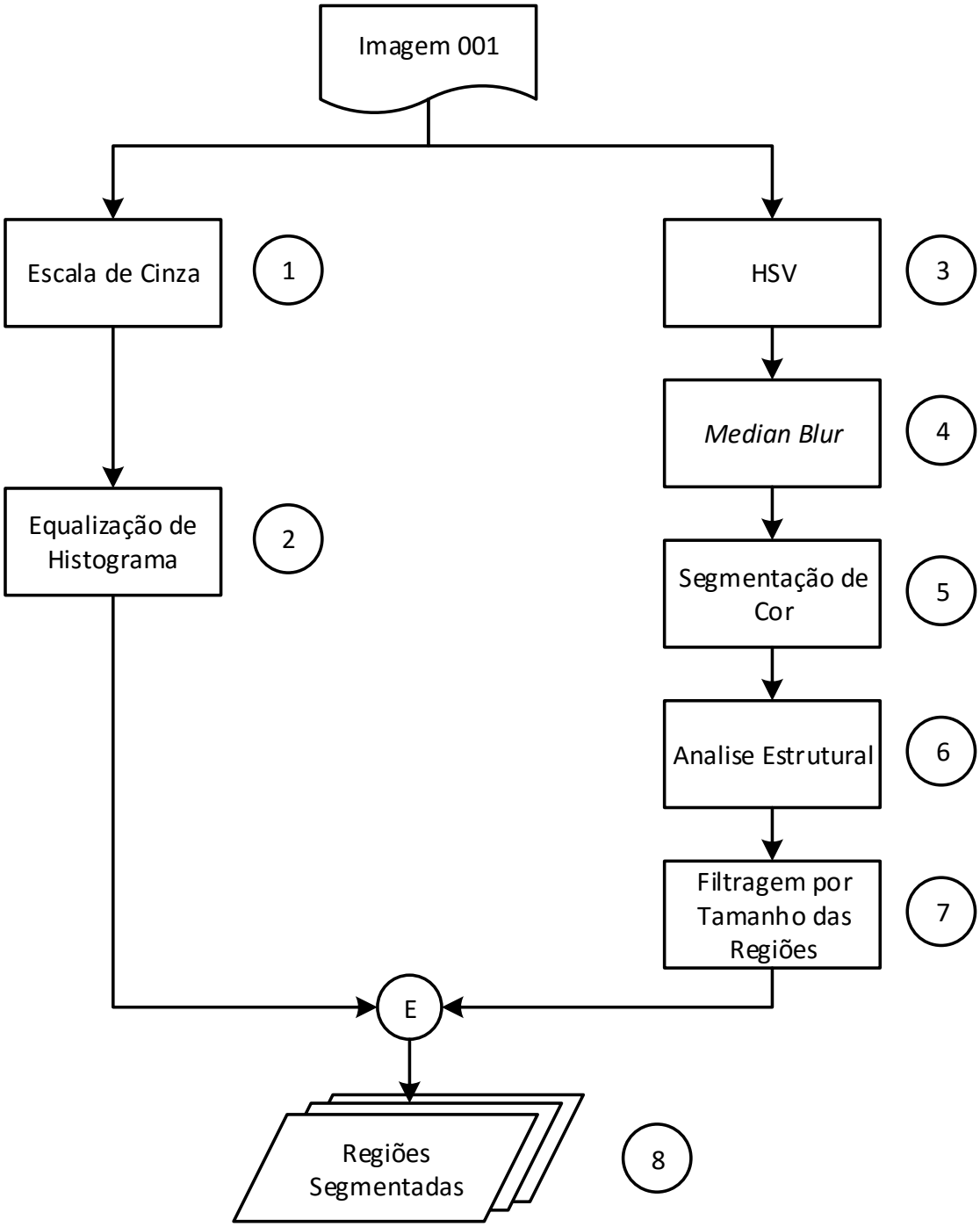


Figura 45: Fluxograma do processo de segmentação.

Blue) e é convertida para HSV (*Hue, Saturation, Value*) de acordo com Equação 3.1.

$$V = \max(R, G, B)$$

$$S = \begin{cases} \dfrac{V - \min(R, G, B)}{V} & \text{Se} \quad V \neq 0 \\ 0 \qquad \text{Senão} \end{cases}$$

$$H = \begin{cases} \dfrac{60 \times (G - B)}{(V - \min(R, G, B))} & \text{Se} \quad V = R \\ 120 + \dfrac{60 \times (B - R)}{(V - \min(R, G, B))} & \text{Se} \quad V = G \\ 240 + \dfrac{60 \times (R - G)}{(V - \min(R, G, B))} & \text{Se} \quad V = B \end{cases} \tag{3.1}$$

Onde:

R, G, B variam de 0 e 1.

H varia de 0 a 360 graus.

S* e *V variam de 0 a 1.

A escolha deste espaço de cores torna a segmentação robusto a variação na luminosidade. Isto acontece, pois, este espaço de cores separa a variável cromática (*Hue*) da luminosa (*Value*), de acordo com Mogelmose, Trivedi e Moeslund (2012). Após a conversão do espaço de cores a imagem é suavizada através de um filtro médio (Figura 45 - Passo 4), para a redução de possíveis ruídos. Desta maneira é então realizada a segmentação por cor (Figura 45- Passo 5) no canal de cores *Hue*.

As cores das placas de trânsito de detectadas são: vermelho e azul. Para cada cor ocorre a binarização conforme a Equação 3.2. Os valores de *threshold* para cada cor foram

encontrados de forma empírica e estão listados na Tabela 7.

$$Pixel(x,y) = \begin{cases} \text{Vermelho, Se } Hue(x,y) \geq ThR_1 \text{ e } Hue(x,y) \leq ThR_2 \\ \text{ou} \\ Hue(x,y) \geq ThR_3 \text{ e } Hue(x,y) \leq ThR_4 \\ \\ \text{Azul, Se } Hue(x,y) \geq ThB_1 \text{ e } Hue(x,y) \leq ThB_2 \\ \\ \text{Outra Cor, Caso contrário} \end{cases} \tag{3.2}$$

Onde:

$\boldsymbol{ThR_1, ThR_2}$ correspondem a primeira faixa do *threshold* para a cor vermelho.

$\boldsymbol{ThR_1, ThR_2}$ correspondem a segunda faixa do *threshold* para a cor vermelho.

$\boldsymbol{ThB_1, ThB_2}$ correspondem a primeira faixa do *threshold* para a cor azul.

A imagem binarizada, Figura 46b, é utilizada para no algoritmo de componentes conectadas para análise estrutural (Figura 45 - Passo 6), que extrai grupos de *pixels* conectados que forma regiões, e através destes são calculadas as caixa delimitadora (*bounding box*). A seguir estas caixas são filtradas (Figura 45 - Passo 7) conforme os seguintes aspectos: proporção entre altura e largura, altura e largura mínima.

Tabela 7: Valores de *Threshold* para as cores vermelho e azul

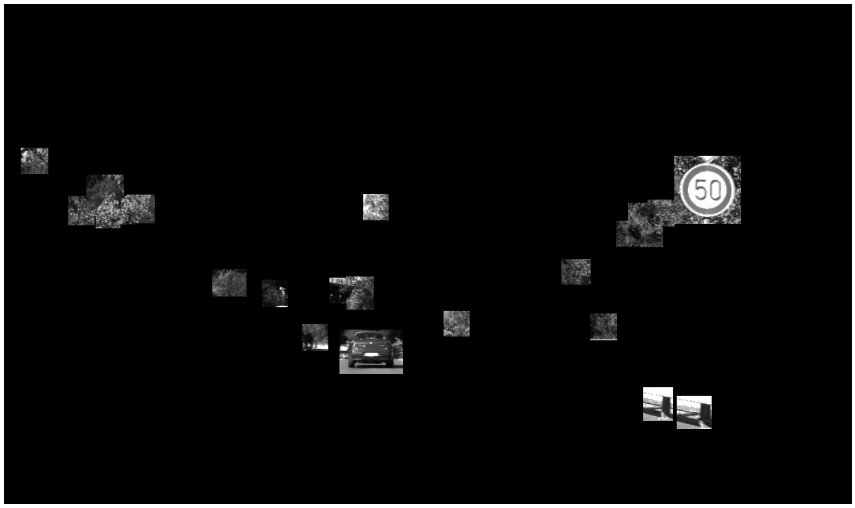
Cor	**Valores de Threshold**
Vermelho	$ThR_1 = 140$, $ThR_2 = 180$ $ThR_3 = 0$, $ThR_4 = 15$
Azul	$ThB_1 = 107$, $ThB_2 = 133$

Os valores para cada um destes aspectos foram encontrados de forma empírica. As caixas selecionadas têm um valor de margem adicionada. Por fim a máscara é utilizada para filtrar, através de uma operação “e logica”, as regiões de interesse na imagem (resultado do Figura 45-passo 8) em escala de cinza equalizada. Na Figura 46c é possível

visualizar o resultado final do processo completo. As regiões segmentadas irão alimentar a etapa de teste na fase de detecção.

(a)

(b)



(c)

(d)

Figura 46: Resultados da detecção. (a) Imagem Original; (b) Segmentação por cor; (c) Regiões de Interesse; (d) Localização da placa detectada.

3.2 Detecção

Como descrito anteriormente a fase de detecção, para este trabalho, é composta de duas etapas: treinamento e teste. A seguir serão descritas ambas etapas, que podem ser acompanhas no Figura 47.

Na etapa de treinamento utilizou-se, como base para a detecção, o *framework* de detecção de objetos proposto por Viola e Jones (2001). Para o treinamento são necessários tanto exemplos positivos, contendo placas de sinalização, como exemplos negativos, imagens que esteja fora do domínio da detecção.

O primeiro passo para a *framework* é a extração das características de cada um dos exemplos positivos e negativos. Para este trabalho foram extraídas características MB-LPB (*Multi-Bloc Local Binary Pattern*) Zhang et al. (2007), as quais exigem um custo computacional menor, culminando assim em um treinamento e execução mais rápido do que a Haar-Wavelets (VIOLA; JONES, 2004).

Extraídas as características MB-LBP, o algoritmo Adaboost (VIOLA; JONES, 2004) é utilizado para a seleção de características e construção da cascata de classificadores, composto de vários classificadores fracos. Esta cascata é construída para que haja um alto nível de rejeição nos níveis iniciais. Sendo assim regiões que não tem qualquer relação com o objeto a ser detectado são rapidamente descartadas.

Na etapa de teste Figura 47 uma imagem é apresentada para o sistema de detecção

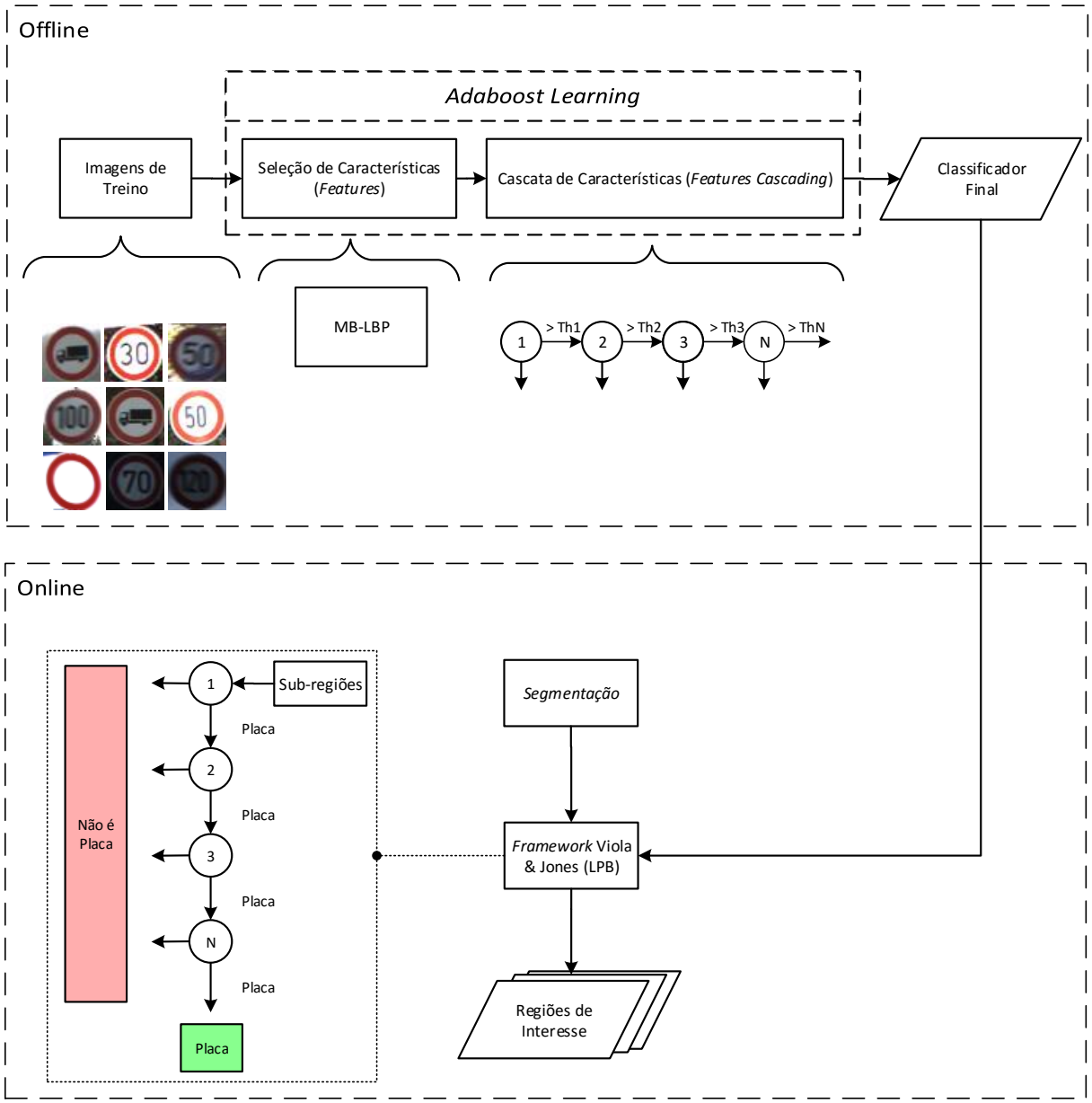


Figura 47: Fluxograma do processo de segmentação.

e o processo de segmentação é realizado, gerando várias regiões segmentadas. A seguir janelas sobrepostas, de tamanho fixo, são deslocadas sobre estas regiões. Cada sub-região é classificada, através da cascata de classificadores, como contendo ou não uma placa de trânsito. Isto é realizado para várias escalas e ao fim os resultados em cada uma das escalas são agrupados em uma caixa delimitadora, com grande probabilidade possuir o objeto de interesse.

3.3 Reconhecimento

Após selecionado as regiões de interesse é necessário classifica-las em relação aos tipos de placas de trânsito que se quer identificar. Dessa maneira, assim como na detecção, esta fase é dividida em duas etapas, que são descritas a seguir e podem ser acompanhadas no fluxograma da Figura 48.

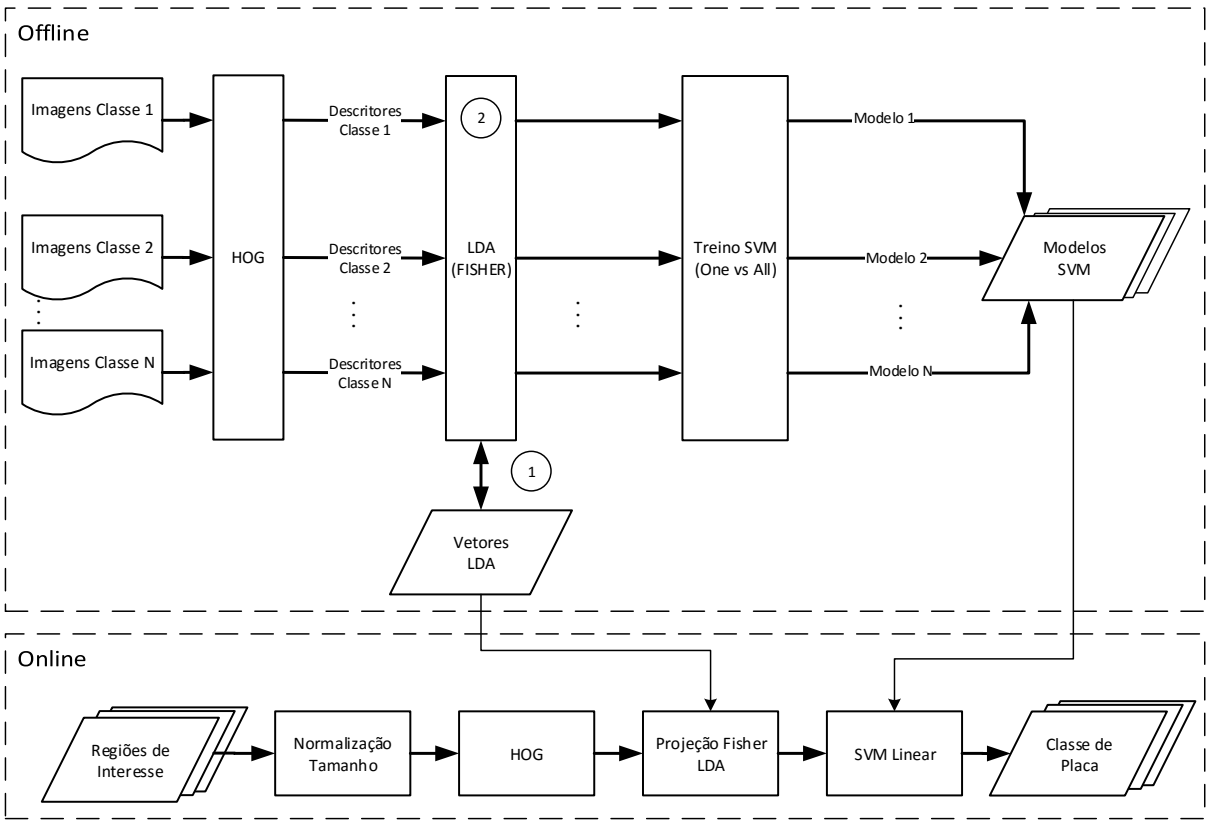


Figura 48: Fluxograma do processo de segmentação.

O processo de treinamento inicia-se pela seleção de exemplos de imagens positivas para cada um dos tipos de placas e redimensionamento para um tamanho fixo. De cada imagem selecionada são extraídos descritores de características HOG (*Histogram of Oriented Gradients*) (DALAL; TRIGGS, 2005). Diferente da fase anterior, as regiões de interesse

não necessitam de uma melhora no contraste, pois, o o próprio extrator da característica HOG realiza a normalização do contraste em uma de suas ultimas etapas, como visto na Subseção 2.6.1.

Este descritor conta a ocorrência de orientações de gradientes em porções localizadas das imagens. O cálculo é realizado em um *grid* denso de células uniformemente espaçadas e utiliza uma normalização de contraste sobreposta para a melhorar a acurácia. A Figura 49 exemplifica como as orientações dos gradientes ficam distribuídos sobre este *grid*. Com é possível observar, as direções dos gradientes seguem aproximadamente o forma geométrica da placa de trânsito.

Figura 49: HOG (Histogram of Oriented Gradients) da placa detectada.

Para este trabalho foram utilizadas células de tamanho 5×5 *pixels*. Cada bloco é formado por 10×10 *pixels*, totalizando por bloco. O deslocamento entre blocos (*stride*) escolhido foi de 5 *pixels* ou uma célula. As células são discretizadas em 8 *bins*, nos quais são contabilizados as magnitudes dos gradientes. A Figura 50 ilustra o cálculo do tamanho vetor de característica HOG. Com o é possível observar essa configuração produz um total de 64 blocos, os quais contem 4 células e estão discretizadas em 8 *bins*. Dessa maneira, o tamanho total do vetor de característica é de 1568 entradas. O tamanho exagerado do vetor torna necessário a aplicação de alguma técnica de redução de dimensionalidade.

Os descritores são combinados em uma matriz, que é utilizado para extração discriminantes lineares (Figura 48 – Passo 1), armazenados para uso posterior, por meio do cálculo do LDA (*Linear Discriminant Analysis*) (SCHOLKOPFT; MULLERT, 1999). Esta técnica é responsável por projetar o conjunto de treinamento em um espaço de menor

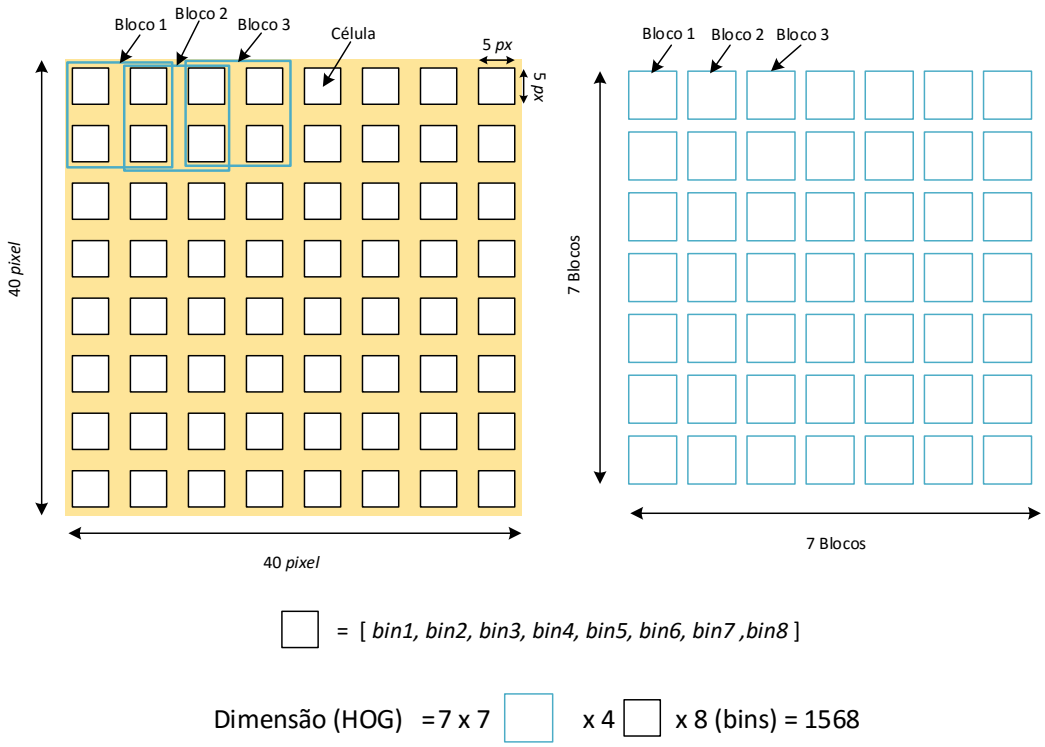


Figura 50: Cálculo da dimensão do vetor HOG, com os parâmetros utilizados.

dimensionalidade com uma boa separabilidade entre as classes.

Dessa maneira, ocorre a redução da dimensionalidade e, consequentemente, a redução no tempo de treinamento e teste. Os vetores de características que antes possuíam 1564 dimensões são transformados para um vetor com apenas 43 dimensões. A aplicação desta técnica impede o problema de *overfitting*, no qual o modelo se ajusta de forma demasiada ao conjunto de treinamento tendo seu desempenho reduzido em imagens de teste.

Após a redução da dimensionalidade dos descritores, é realizado o treinamento utilizando-se a técnica de aprendizagem de máquina supervisionada SVM (*Support Vector Machine*) linear (CHANG; LIN, 2011). Para o ajuste dos hiperparâmetros é utilizado a validação cruzada e os descritores de característica são normalizados por meio da norma L2 *(L2-Norm*). Como descrito na Subseção 2.6.3, a normalização é uma etapa importante antes do treinamento de uma SVM.

O esquema de validação cruzada utilizado foi o *k-fold*, sendo o valor de k escolhido igual a 10. Para se encontrar os hiperparâmetros foi realizado uma busca em *grid* (*grid search*), no qual um número limitados de valores para C, dentro de uma faixa, tiveram seu desempenho aferido durante a validação cruzada.

Em seguida estes descritores são agrupados nas suas correspondentes classes e então

é realizado o treinamento. O método de treinamento utilizado, para um classificador multiclasses, foi o One-vs-All (MIRZA; LIN, 2013; HSU; LIN, 2002). Esta estratégia consiste em treinar um classificador binário por classe, com os exemplos desta classe como exemplos positivos e os outros exemplos como negativos. Os modelos para cada tipo de placa são armazenados para uso na etapa de teste.

As regiões de interesse provenientes da fase de detecção são, em tempo de execução (etapa de teste), classificadas em um dos tipos de placas que o sistema foi treinado. Estas são redimensionadas e tem o descritor de característica HOG extraído. OS parâmetros utilizados para estes processos são os mesmos utilizados na etapa de treinamento.

Em seguida o descritor HOG é projetado no subespaço determinado pelos discriminantes lineares (Figura 48 – Passo 2), armazenados na etapa anterior. O vetor resultante é então avaliado por cada um dos classificadores binários treinados pela técnica SVM.

A estratégia One-vs-All adotada no treinamento requer que cada classificador binário produza uma pontuação de confiança com valores reais para que seja tomada a decisão. Dessa forma os classificadores são treinados para retornarem estimativas de probabilidade, como definido em Wu, Lin e Weng (2004). A avaliação é realizada para cada um dos modelos e os valores de probabilidades estimadas são comparados. O classificador que emitir o maior valor tem sua classe atribuída a região de interesse testada.

3.4 Integração com Sistema Embarcado

Após a implementação da metodologia proposta foi realizada a segunda fase do desenvolvimento, que consistiu na integração do sistema de visão computacional ao sistema embarcado. Grande parte dessa fase foi realizada através da compilação de código fonte. Existe a possibilidade de se compilar código fonte no *Target*, porém o processo pode ser demorado e inconveniente. Assim, optou-se pela compilação cruzada, que permite compilar código fonte utilizando-se uma máquina com mais recursos computacionais e reduzir o tempo necessário para o processo de compilação.

Para se realizar a compilação cruzada são necessárias ferramentas compiladas para a arquitetura do *Host*, que gerem código executável para o *Target*. Para isso, a primeira etapa da fase de integração foi a instalação de uma *toolchain* de compilação cruzada no *Host*. A compilação nativa gera código binário para arquitetura x86, enquanto que a compilação cruzada gera código binário para a arquitetura ARM. A Figura 51 apresenta um ilustração da diferença entre a compilação nativa e cruzada, que já foi discutida na

Seção 2.7.

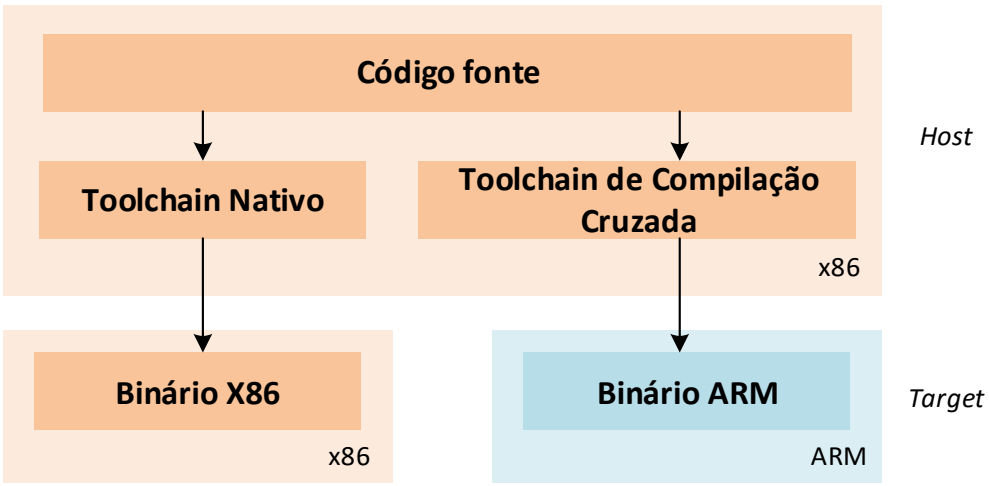


Figura 51: Diferença compilação nativa e cruzada.

Uma toolchain consiste em um conjunto de ferramentas tais como: *binutils*, compilador *C/C++*, blbioteca *C* e um *debugger*. O *binutils* é composto por duas importantes ferramentas: o *assembler*, que converte um código em *assembly* (linguagem de máquina) em um binário, e o *linker*, que liga diversos códigos objetos em uma biblioteca ou em um executável(HALLINAN, 2007). Outras ferramentas podem integrar o *binutils* como, por exemplo, ferramentas para a manipulação de binários e analise.

Existem diversas maneiras de se obter uma *toolchain*. A primeira forma é através de *toolchains* pré-construídas, disponibilizadas por fornecedores ou que acompanhem nas BSP (*Bord Support Package*) do fabricante do *hardware*. O seu uso tem como desvantagem a perda de flexibilidade na escolha das características da *toolchain* como, por exemplo, o tipo de biblioteca *C* utilizadas. Já a vantagem é a facilidade de se iniciar o desenvolvimento sem se preocupar com a criação das ferramentas.

Construir a própria *toolchain*, utilizando ou não ferramentas automáticas, é outra maneira de se obter a *toolchain*. No entanto, tal abordagem pode demandar tempo e exige um conhecimento aprofundado sobre o tema. Além disso, tal abordagem foge do escopo dos objetivos desta dissertação.

Para sistemas embarcados, que utilizam arquitetura ARM, existem dois fornecedores, mais conhecidos, de *toolchain* pré-compiladas: *CodeSourcery*[2] e *Linaro*[3]. A *CodeSourcery* fornece uma solução integrada, que compreende uma IDE com uma a toolchain GNU para desenvolvimento em diversas arquiteturas. No entanto, para o desenvolvimento com

[2]CodeSourcery: http://www.codesourcery.com/
[3]Linaro: http://linaro.org/

processadores ARM é necessário se recorrer a versão comercial do *software*.

Já a *Linaro* fornece *toolchains* otimizadas para os mais recentes processadores ARM de forma gratuita e *open source*. Outra vantagem é a disponibilidade de pacotes para sua instalação na distribuição Ubuntu, o qual foi escolhido como *Host*. Devido a estas vantagens, a *toolchain* fornecida pela *Linaro* foi instalada no *Host* por meio do gerenciador de pacotes.

As ferramentas do *toolchain* possuem um prefixo, que indica suas características. Esse prefixo é composto por várias partes, separadas por hífen, na seguinte ordem: arquitetura, sistema operacional e ABI (*Application Binary Interface*). Para o *hardware* escolhido a arquitetura do sistema embarcado é ARM e o sistema operacional utilizado é o Linux.

A ABI define uma interface binaria de baixo nível entre uma ou mais partes de um programa em uma determinada arquitetura. Isso determina como uma aplicação interage com ela mesma, com outras aplicações e com o kernel. Para sistemas embarcados essa interface é conhecida como EABI (*Embedded-Application Binary Interface*), que estabelece convenções padrões para formatos de arquivos, tipos de dados, uso de registradores, organização do *stack frame* e passagens de parâmetros em funções.

Tanto a ABI quanto a EABI são interfaces estabelecidas para o uso em sistemas *baremetal*. Caso haja a utilização de um sistema operacional Linux a interface é conhecida como GNUEABI. Além disso, existem duas versões da gnueabi: *hard-float* e *soft-float*.

A versão *hard-float* faz uso do hardware de ponto-flutuante, conhecido como VFP (*Vector Floating Point*) em processadores ARM, enquanto que a *soft-floa*t não o utiliza. Neste trabalho, a versão *hard-float* foi utilizada e a justificativa para isso será vista mais à frente. Dessa forma, todas a ferramentas do compilador instalado terão o prefixo *arm-linux-gnueabihf*, como pode ser visto na Figura 52.

Para o desenvolvimento do sistema embarcado foi escolhido como hardware alvo a *BealgeBone Black* (BBB) (COLEY, 2013), um *hardware* de baixo custo, baixo consumo energético e compacto. Custando apenas 45 dólares é um projeto *open hardware*, o que significa que podem ser realizadas melhorias no projeto original, tais como: redução do tamanho da placa e diminuição consumo energético. Estas características permitem a fácil integração do sistema embarcado nos mais diversos tipos sistemas automotivos.

Para o início do desenvolvimento no *hardware* escolhido foi necessário a instalação de um sistema operacional Linux. Uma distribuição Linux é composta pelo *kernel* e o sistema de arquivos raiz (*rootfs*). O *kernel* é parte fundamental de um sistema operacional,

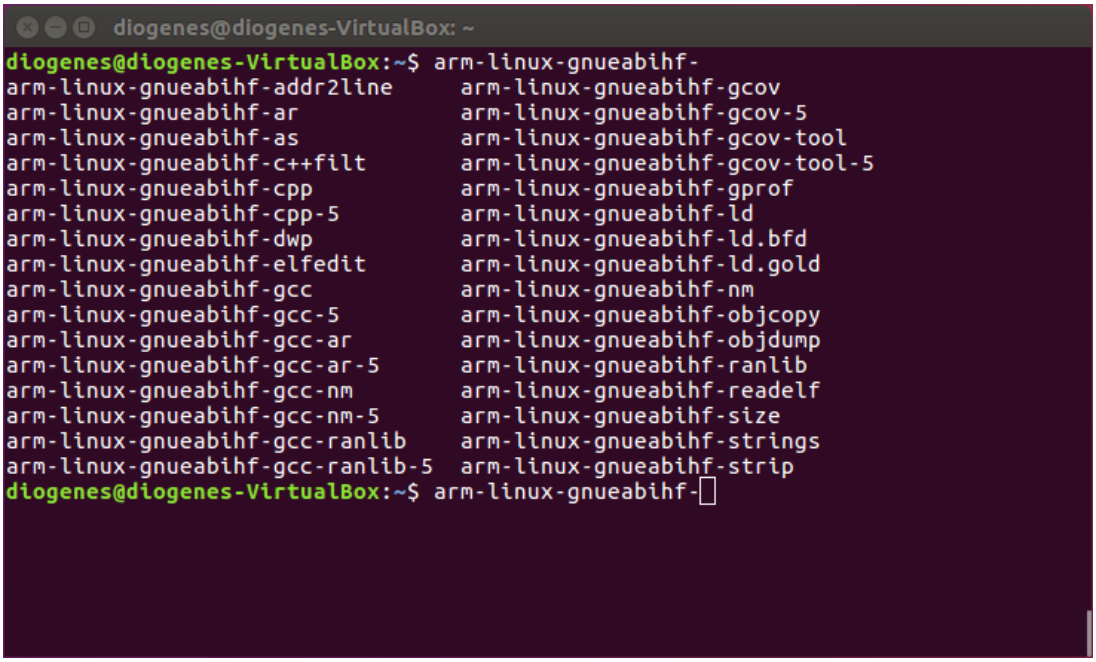


Figura 52: Ferramentas presentes na *toolchain arm-linux-gnueabihf*.

sendo responsável por gerenciar os recursos como memória, processador e dispositivos. Além disso, ele prove uma interface de comunicação, conhecidas como *system calls*, entre aplicações e o *hardware*. A Figura 53 ilustra essas características do *kernel*.

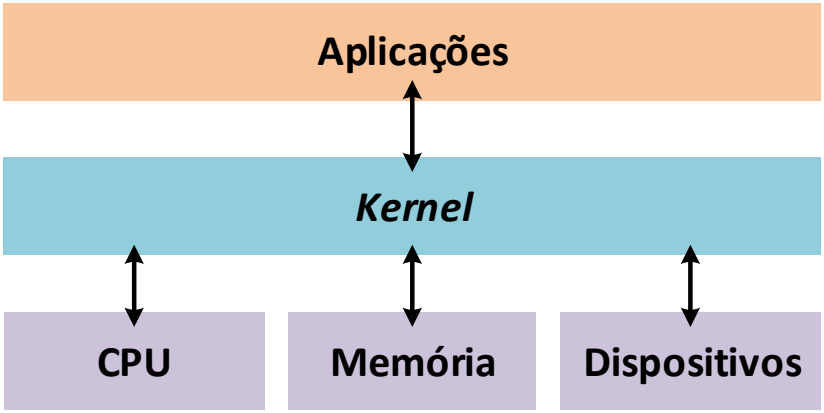


Figura 53: Comunicações feitas pelo *Kernel*.

O *roofs* é um sistema de arquivo contido na mesma partição em que o diretório raiz e nele são montados todos outros sistemas de arquivos durante o processo de inicialização. O seu conteúdo pode variar, mas no mínimo conterá os arquivos que são necessários para o *boot* (inicialização) do sistema e para traze-lo até um estado em que outros sistemas de arquivos possam ser montados. No caso de distribuições Linux é principalmente esse componente que diferencia uma distribuição de outra. Dessa forma, a instalação de um sistema operacional compreende na gravação desses componentes em um cartão SD ou na

memória *Flash*.

A BBB suporta diversas distribuições Linux, como Angstrom, Ubuntu, Debian, Gentoo, entre outras. Devido à popularidade e para manter a mesma distribuição do *Host*, foi instalado a distribuição Ubuntu 12.04 LTS. Optou-se pela não utilização de uma interface gráfica do sistema operacional, para se economizar recursos computacionais. Existem diversas maneiras de se obter a distribuição, sendo a mais simples por meio do download de uma imagem da distribuição para se gravar no cartão SD, que contém o *kernel* e o *rootfs* do Ubuntu. Essa imagem pode ser obtida no site oficial da *BeagleBoard Black*[4], onde também são providas as instruções de instalação.

A integração em sistemas embarcados pode se tornar algo repetitivo, uma vez que é um processo que passa por sucessivos testes. Para agilizar e reduzir a necessidade de gravação da imagem a cada ajuste optou-se por fazer com que o sistema de arquivos raiz resida no *Host*, facilitando as modificações e ao mesmo tempo permitindo uma compilação cruzada transparente, no que diz respeito ao acesso das bibliotecas do *Target*.

Na Figura 54 pode-se observar um esquema usual para se trabalhar com sistemas embarcados. Nele o *Host* e o *Target* estão conectados por duas interfaces: Serial e *Ethernet*. A interface serial é utilizada principalmente para realizar configurações no *U-boot*, sendo possível também terminal de linha de comando do sistema operacional. Através da interface *ethernet* é possível a montagem do sistema de arquivos raiz no *Host*, através do NFS(*Network File System*).

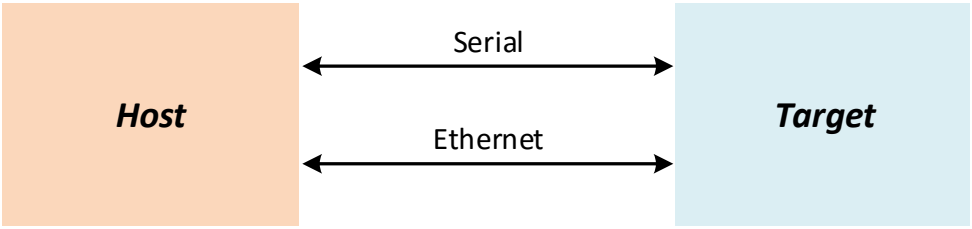


Figura 54: Esquema usual para o trabalho com sistemas embarcados.

Para a montagem do *rootfs* é necessário a configuração do *boot* do sistema embarcado. O processo completo de *boot* da BBB possui quatro estágios. Quando a BBB é ligada, a ROM (*Read-only Memory*) interna é carregada e imediatamente procura pelo arquivo do *bootloader* de segundo estágio, conhecido por *X-loader*.

O *X-loader* é executado, realiza algumas inicializações básicas e carrega o *bootloader* de terceiro estágio. Então, o *U-boot* (terceiro estágio) carrega a imagem do *kernel* e o

[4]BBB: https://beagleboard.org/

DTS (*Device Tree Source*) na memória, e inicializa o *kernel* com alguns argumentos de linha de comando, conhecido como *bootargs*. Por fim o Linux inicia e monta o sistema de arquivos raiz.

Assim, foi configurado o *U-boot*, que já veio gravado de fábrica e com suporte a NFS, através dos argumentos do *bootargs*. Em seguida, configurou-se o Host como servidor para encontrar o *Target* na rede. Implantando esse esquema de trabalho, as etapas relacionadas ao *Target* passam a ser todas realizadas no *Host*.

O último passo para a configuração do ambiente de trabalho foi a compilação cruzada e instalação das bibliotecas Qt e OpenCV no *Target*. No site oficial do OpenCV é disponibilizado o código fonte da biblioteca. Esse foi baixado e descompactado na máquina *Host*. Além disso, foi instalado o CMake no *Host*, pois, o projeto do OpenCV é baseado nele. O *CMake* é um gerenciador do processo de construção de um *software*, utilizando um método que não depende do compilador escolhido.

A compilação do OpenCV exige a instalação de diversas dependências. Essas foram instaladas, através do gerenciador de pacotes *apt-get* no *Target*, antes do processo de compilação. Infelizmente, não existe uma maneira pratica de apontar para o compilador cruzado as bibliotecas do sistema *Target*.

O compilador da *toolchain* somente consegue encontrar as bibliotecas presentes no sistema *Host* e mesmo utilizando diretivas para encontrar as corretas, o compilador acaba confundindo e misturando bibliotecas dos dois sistemas. Dessa maneira, tal abordagem tornou-se impraticável, sendo necessário a pesquisa por outro tipo de solução.

A solução encontrada para resolver-se o problema de dependências, durante a compilação cruzada, foi o uso da aplicação *Scratchbox2*. Esse é baseado no conceito de *chroot*, que consiste em mudar o diretório raiz do processo raiz e de seus filhos para um diretório especificado.

O *Scratchbox2* é composto por um conjunto de ferramentas projetadas para tornar mais simples o desenvolvimento de sistemas embarcados. Através da criação de um ambiente virtual (*sandbox*) é possível enganar o compilador cruzado para encontrar as dependências necessárias. Além disso, essa ferramenta prove uma tecnologia conhecida como *CPU-Transparency*, em que executáveis construídos para o Host e *Target* podem ser executados no *Host*. Os executáveis do *Host* rodam nativamente, enquanto que as aplicações do *Target* são emuladas.

Dessa maneira, foi instalado essa aplicação e configurada para que realize o *chroot*

no *rootfs* do *Target*, montado via NFS. Foram apontados como compiladores do *sandbox* criado, o compilador cruzado instalado no *Host*. O diretório no *Host* que continha o código fonte do OpenCV foi mapeado para um diretório no *sandbox*.

Após a devida configuração processo de compilação na *sandbox*, o projeto do OpenCV foi configurado através do CMake presente no *Host*. Nessa configuração foram ativadas as instruções NEON, uma extensão avançada SIMD (*Single Instruction, Multiple Data*) projetada para aceleração de multimídia e algoritmos de processamento de sinais, como codificação/decodificação de vídeo, jogos, processamento de imagens, entre outros.

O incremento no desempenho, através dessas instruções, é obtido pela paralelização na execução das instruções, permitindo a manipulação de até 16 dados por registrador por vez (ARM, 2016). Para a compilação do projeto com essas instruções, é necessário a utilização da *toolchain hard-float*, instalada anteriormente, que gera código binário com as instruções NEON.

Com o projeto configurado, a compilação ocorreu no ambiente virtual criado pelo *Scratchbox2*. Nesse processo foram utilizados o compilador cruzado e o projeto OpenCV presentes no *Host* e as dependências instaladas no *Target*. Finalizado a compilação, que levou algumas horas, os módulos do OpenCV foram instalados no *Target*.

A instalação do Qt, diferentemente do OpenCV, pode ser realizada através do gerenciador de pacotes do Ubuntu. Porém, a versão do repositório é muito antiga e não permite a ativação da utilização das instruções NEON. Dessa forma, o procedimento realizado para a instalação do Qt, compreende passos similares aos do OpenCV. A principal diferença se dá pela dispensabilidade de instalação de dependências, tornando o processo muito mais simples. Isso, ocorre, pois, o projeto do Qt disponibiliza o código fonte de todas as dependências, que são compiladas junto como o projeto principal.

O código fonte do *framework* Qt foi baixado e compilado utilizando-se esquema de compilação cruzada. O arquivo de configuração do *framework* foi modificado para utilizar as instruções NEON e a *toolchain* de compilação cruzada.

Antes da compilação foi executado uma ferramenta de configuração do projeto, similar ao CMAKE, onde especificou-se todos detalhes da compilação. Então, realizou-se a instalação das bibliotecas dinâmicas no *Target* e a configuração do *Qt Creator* para encontrar tanto as bibliotecas do Qt quanto do OpenCV, presentes no diretório do *rootfs* montado no *Host* via NFS. Outra modificação necessária foi a indicação para a IDE utilizar o gerenciador de bibliotecas (*pkg-config*) do ambiente virtual criado pelo *Scratchbox2*.

Isso é necessário, pois, o gerenciador de bibliotecas do *Host* não sabe onde estão as bibliotecas do Target. O *Qt Creator* utiliza essa aplicação para encontrar as bibliotecas que deve vincular ao código a ser compilado. Se não for feita esta modificação, o código será veiculado a bibliotecas presentes no *Host* ou a compilação não será realizada pela falta das bibliotecas.

Também, instalou-se o *gdbserver* no *Target*, para que no processo de compilação o executável criado no *Qt Creator* fosse enviado e executado diretamente no *Target*, sendo possível realizar a *debug* no próprio *Host*. Isso facilitou a realização dos testes e extração de resultados, tanto no *Target* quanto no *Host*, que será visto no próximo capitulo.

4 Resultados

4.1 Bases de Dados

Grande parte das aplicações de visão computacional do mundo real necessitam de imagens de treinamento. No campo de reconhecimento e detecção de placas de trânsito não é diferente, pois, muitas das técnicas envolvidas estão relacionadas à aprendizagem de máquina. Além de um conjunto de treinamento é necessário um conjunto de teste, onde o desempenho do sistema será aferido, permitindo melhorias e comparações entre trabalhos.

De acordo com Stallkamp et al. (2012) poucas comparações sistemáticas e imparciais entre esses sistemas tem sido feita. Muitos estudos publicados não utilizam as mesmas bases de dados (*datasets*) e tratam diferentes etapas, tornando difícil a comparação entre os mesmos. Além disso, existem trabalhos que focam somente em determinados tipos de classes, como as placas referentes ao limite de velocidade.

Um motivo para tais dificuldades se deve à poucas bases de dados disponíveis publicamente, sendo a grande maioria próprias ou de empresas. Como exemplo tem-se as seguintes bases públicas:

- German TSR Benchmark (GTSRB) (STALLKAMP et al., 2012).
- German TSD Benchmark (GTSDB) (HOUBEN et al., 2013).
- KUL Belgium Traffic Sign DataSet (KUL Data set) (TIMOFTE; ZIMMERMANN; GOOL, 2014).
- Swedish Traffic Signs Data set (STS Data set) (LARSSON; FELSBERG, 2011).
- LISA Dataset (MOGELMOSE; TRIVEDI; MOESLUND, 2012).

Cada uma dessas bases de dados é referente a um pais e possuem um determinado número de classes. As quantidades de exemplos variam de dezenas a dezenas de milhares

e nem todas possuem os exemplos anotados. Em seu trabalho, Mogelmose, Trivedi e Moeslund (2012) fornece um comparativo entre elas com relação a essas e várias outras características.

Uma das bases de dados mais difundidas na área de reconhecimento de sinalização é a GTSRB (*German Traffic Sign Recognition Benchmark*), que foi criada para uma competição realizada na *International Joint Conference on Neural Networks* (IJCNN) em 2011. É uma das maiores bases de dados públicas disponíveis e contem placas de trânsito da Alemanha, que são adequadas para se treinar e testar sistemas que se adequem a convenção de Vienna, que harmoniza sua aparência em 62 países (HOUBEN et al., 2013).

Ela foi criada através de aproximadamente 10 horas de vídeo coletados enquanto se dirigia, de dia, por diferentes tipos de rodovias da Alemanha sob as mais diversas condições de iluminação, oclusões parciais e rotações. É composta por 43 classes contendo mais de 50 mil imagens anotadas, divididas entre treino (39.209 imagens) e teste (12.630 imagens). Na Figura 55 é possível se observar todas as 43 classes disponíveis.

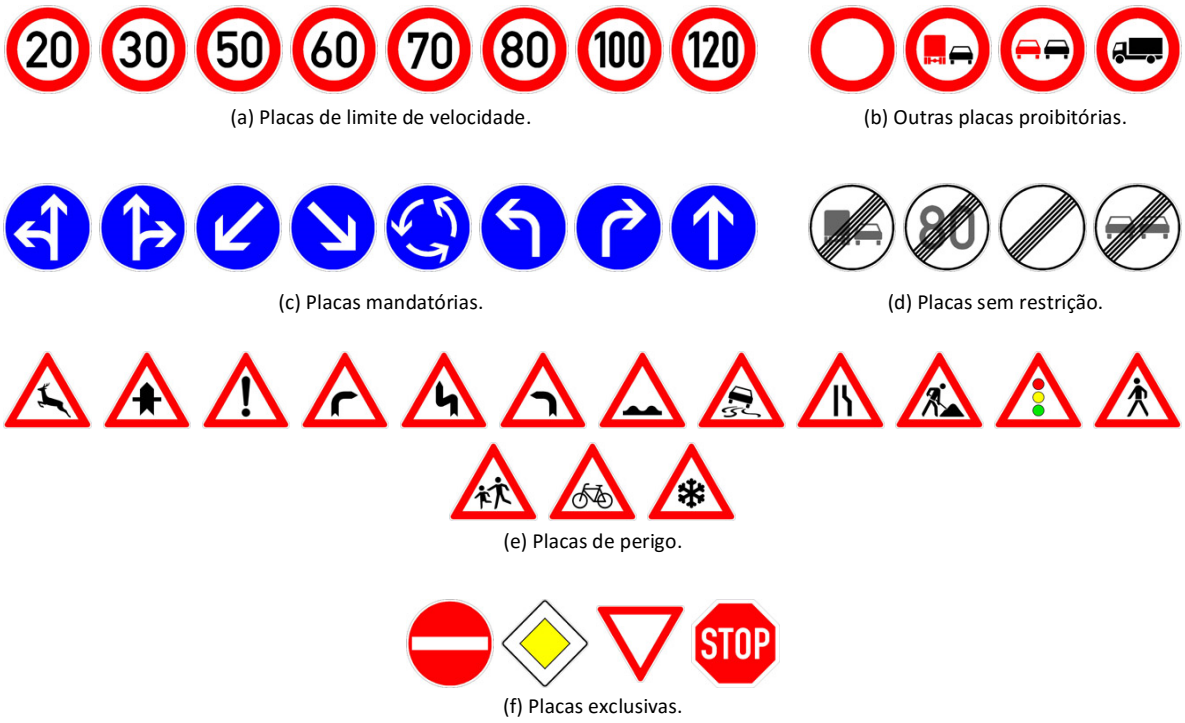


Figura 55: Placas de trânsito presentes na base de dados GTSRB e seus respectivos grupos. FONTE: (STALLKAMP et al., 2012)

Cada classe contém diversas instâncias de placas de trânsito, composta de múltiplas imagens de uma mesma placa. Seus tamanhos variam de 15×15 a 222×193 *pixels*. Essas imagens consecutivas, capturadas em diferentes distâncias, apesar de parecerem

idênticas, podem variar ter variações significantes para com a diversidade da base de dados. Imagens com distâncias grandes possuem baixa resolução enquanto que as mais próximas são propensas ao *motion blur*. A Figura 56 apresenta alguns exemplos de placas de sinalização presentes nessa base de dados.

Figura 56: Exemplos de placas de trânsito, presentes na GTSRB, nas mais diversas condições. FONTE: (STALLKAMP et al., 2012)

A GTSRB é uma base de dados, que para um observador humano, varia muito em termos de qualidade e legibilidade (STALLKAMP et al., 2012). Suas imagens possuem vários artefatos visuais devido à baixa resolução, baixo contraste, *motion blur* e reflexo, originados do processo de aquisição de dados e hardware. Stallkamp et al. (2012) realizou experimentos para se medir o desempenho de um ser humano na classificação de placas referentes a base de dados de teste. A média de acurácia, para um humano, encontrada foi 98,84%, enquanto que o melhor resultado para um indivíduo foi de 99,22% de acurácia.

Outra tarefa importante, em um sistema de identificação de placas de trânsito, é a detecção (*TSD – Traffic Sign Detection*). Diferente da tarefa de reconhecimento, a detecção envolve a localização da placa de sinalização em uma cena natural.

Para Houben et al. (2013) medir o desempenho de um sistema de detecção é mais difícil do que estágio de classificação. Juntamente a isso, as bases de dados públicas existentes em sua grande maioria apresentam pouca variância em relação as condições de iluminação e cenários de condução. Sendo assim, com o intuito de suprimir estas falhas, Houben et al. (2013) propôs a GTSDB (*German Traffic Sign Detection Benchmark*), uma base de dados especifica para o problema de detecção.

A GTSDB baseia-se nas três principais categorias de sinais: Proibitório, Mandatório e Perigo. Cada uma dessas classes é representada por um formato e cor especifico, como pode observar-se na Figura 57.

Figura 57: Grupos de placas, agrupadas por forma e cor, presentes na GTSDB. FONTE: (HOUBEN et al., 2013)

As imagens foram capturadas nos mais diversos cenários (urbano, estradas e rodovias) durante o dia e anoitecer nas mais diversas condições climáticas. Cada placa apresenta tamanhos que variam de 16 e 128 *pixels*, com respeito a sua maior dimensão.

A base de dados compreende mais de 900 imagens de cenas naturais contendo 1206 sinais de trânsito e está dividida entre treino e teste. O conjunto de treino é composto de 600 imagens com 846 sinais, enquanto que o de teste possui 300 imagens e um total de 360 sinais. A Figura 58 mostra alguns exemplos de cenas naturais encontradas na base de teste.

Ambas bases, GTSRB e GTSDB, são desafiadoras, testando um sistema de identificação de placas de trânsito nas mais diversas condições. Neste trabalho elas foram utilizadas para o treinamento da metodologia proposto no Capítulo 3 e serão utilizadas na Seção 4.2, GTSDB, e Seção 4.3, GTSRB, para a extração dos resultados.

Os testes foram realizados em um *desktop* (Intel Core i5 2.5 GHz e 8 Gbytes de memória RAM) e em uma *BeagleBone Black* (ARM Cortex A8 1GHz e 512 Mb de memória RAM). Inicialmente foram treinados modelos para cada uma das fases propostas e posteriormente medidos o desempenho e os tempos de execução em cada uma das plataformas de teste. Todos os treinamentos foram feitos no *Desktop* e o modelo gerado foi copiado para o teste no sistema embarcado.

Figura 58: Exemplo de cenas naturais presentes no conjunto de teste da GTSDB. FONTE: (HOUBEN et al., 2013)

4.2 Resultados Detecção

A etapa de detecção é precedida pela segmentação, que é responsável pela redução do espaço de busca. A segmentação é dependente das condições de iluminação e pode muita das vezes ser influênciada por outros elementos de mesma coloração na cena. O Anexo B mostra alguns exemplos da segmentação proposta (coluna da esquerda).

A base de dados GTSDB fornece poucos exemplos para o treinamento do framework Viola & Jones. Sendo assim foi necessário a criação de exemplos positivos sintéticos. Para cada um dos exemplos fornecidos foram criados outros variando-se a rotação e adicionando-se novos planos de fundo. Este procedimento permite o sistema modelar o problema com maior precisão, conferindo assim uma maior robustez.

O Anexo B apresenta alguns exemplos de cenas onde foram detectadas placas ou houveram falsos positivos. Neste anexo, cada linha apresenta o resultado da segmentação (esquerda) e detecção(direita) para uma determinada cena do conjunto de teste da GTSDB.

Um detector foi criado para cada classe e os testes foram realizados nas imagens fornecidas pela competição. Diferentemente das imagens de treinamento estas são compostas

por cenas naturais. Sendo assim, a objetivo é percorrer a imagem a procura de placas de trânsito, pertencentes a classe, determinando a caixa delimitadora dos mesmos.

Para este trabalho as imagens de teste foram redimensionadas para uma resolução de 640×480 *pixel*. Isto foi necessário para reduzir a carga de processamento no sistema embarcado e tem como implicação a redução da distância máxima com que as placas de trânsito serão detectadas.

Os erros e acerto foram calculados utilizando-se o *ground truth* fornecido pela competição. Ele contém, para cada cena, a posição de cada placa e sua respectiva classe. Cada detector foi testado, em todas cenas, e suas saídas comparado com o *ground truth*. Para ser considerado um acerto e não ser descartado, a caixa delimitadora do detector, com a classe correta, deve ter pelo menos 80% de sobreposição com o resultado esperado, calculado através do coeficiente de *Jaccard* (MATHIAS et al., 2013).

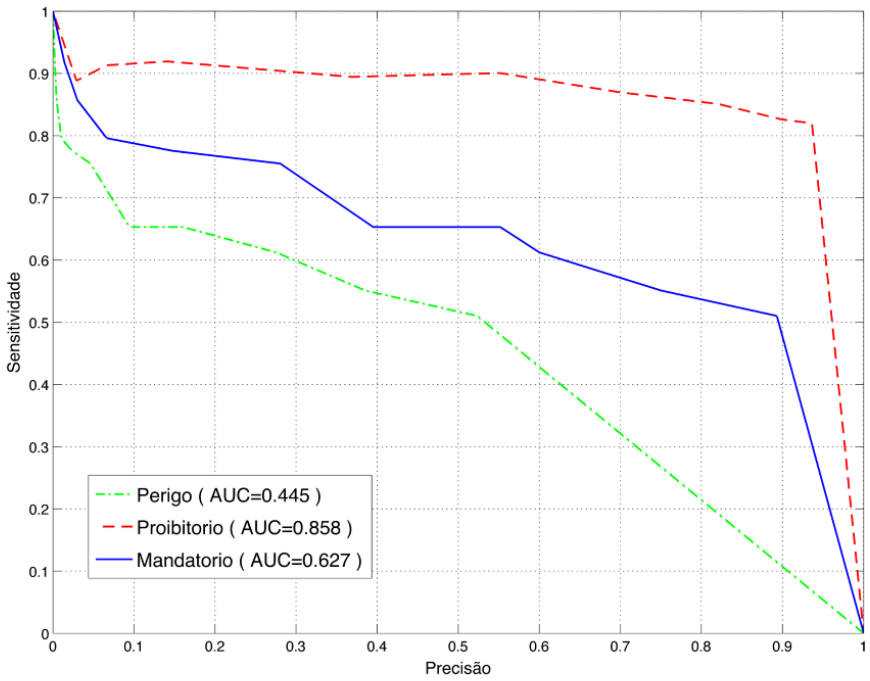
As métricas utilizadas para avaliação do desempenho da detecção foram precisão (*precision*) e sensitividade (*recall*) . A precisão corresponde a fração dos objetos detectados que são relevantes, Equação 4.2. Já sensitividade corresponde a fração dos objetos relevantes que são detectados, Equação 4.1. Para o cálculo destas métricas os erros e acertos são sumarizados em uma tabela contendo os números de verdadeiros positivos (TP), falsos positivos (FP), falsos negativos (FN).

$$Sensitividade = \frac{TP}{(TP + FN)} \tag{4.1}$$

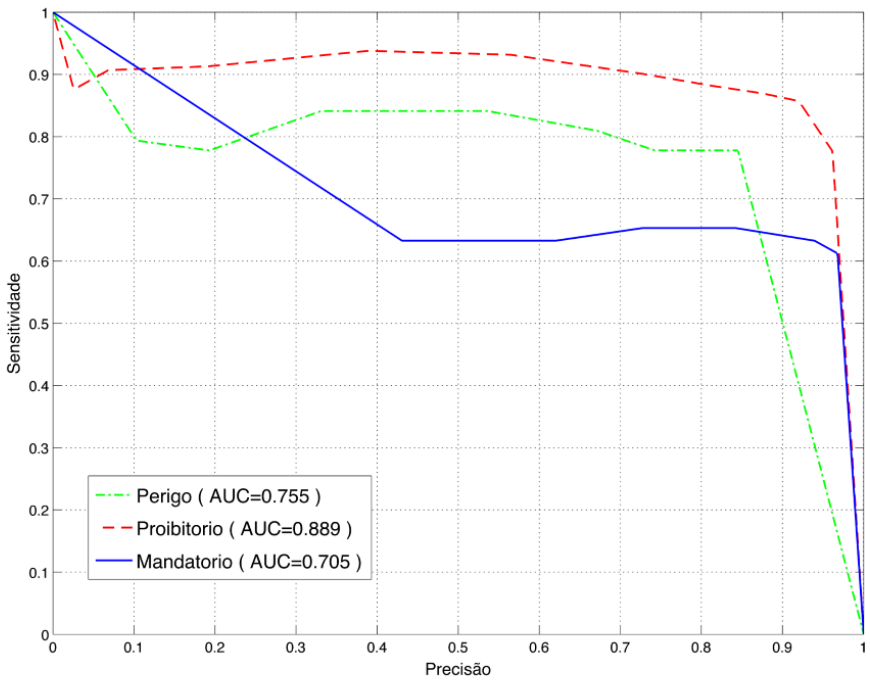
$$Precisão = \frac{TP}{(TP + FP)} \tag{4.2}$$

Por fim, cada detector tem seu número de estágios variado e sua precisão e *recall* calculados. Estes valores são usados para plotar a curva de Precisão versus Sensitividade (*precision vs recall*). Na Figura 59 é possível observar as curvas plotada com e sem o uso de segmentação, respectivamente.

Comparando-se as AUC (*Area Under Curve*) de ambos gráficos é possível observar que o uso de segmentação fez aumentar a eficiência dos detectores das classes proibitórias e mandatória. Também é possível notar suas baixas AUC em relação a classe proibitória. Isto acontece, pois, o número de exemplos fornecidos pela competição é pequeno, além de conter placas de trânsito em condições ruins. Para uma melhora nos resultados uma possível estratégia seria a utilização dos exemplos pertencentes a competição GTSRB.



(a)



(b)

Figura 59: Gráfico comparativo do desempenho da detecção. (a) sem o processo de segmentação; (b) com processo de segmentação.

Neste trabalho, porém, preferiu-se manter os moldes da competição.

A Tabela 8 faz um comparativo da metodologia de detecção proposta com as técnicas apresentas em Houben et al. (2013). Para a extração desses resultados foi considerado uma precisão de 10%. No problema abordado não é necessário uma precisão muito alta, pois é mais importante detectar todas as placa de trânsito em uma cena (alto *recall*) do que a introdução de falsos positivos. Esses podem ser facilmente eliminados na fase de reconhecimento através do treinamento de mais uma classe para rejeita-los. Essa classe pode ser treinada com exemplos positivos provenientes de todas as classes de placas de trânsito e exemplos negativos que não contenham o objeto de interesse.

Tabela 8: Comparativo de taxa de detecção para 10% de precisão. Fonte: (HOUBEN et al., 2013)

Algoritmo	Proibitório	Perigo	Mandatório
Proposto	90,6%	80%	91,2%
HOG + LDA	91,3%	90,7%	69,2%
Hough-like	55,3%	65,1%	34,7%
Viola-Jones	98,8%	74,6%	67,3%

Adicionalmente, foram realizados testes em vídeo para a fase de detecção. Para cada uma das classes foram coletados quatro vídeos diferentes contendo instâncias de placas. Esses foram submetidos ao processo de detecção, sendo contabilizados o número de instâncias detectadas e o de falsos positivos. Então, foram calculados as taxas de detecção para cada exemplo das classes. A Tabela 9 demonstra os resultados obtidos. No Anexo D é possível observar o resultado da detecção para alguns quadros dos vídeos analisados, onde a região da placa localizada está definida por uma caixa delimitadora na cor azul.

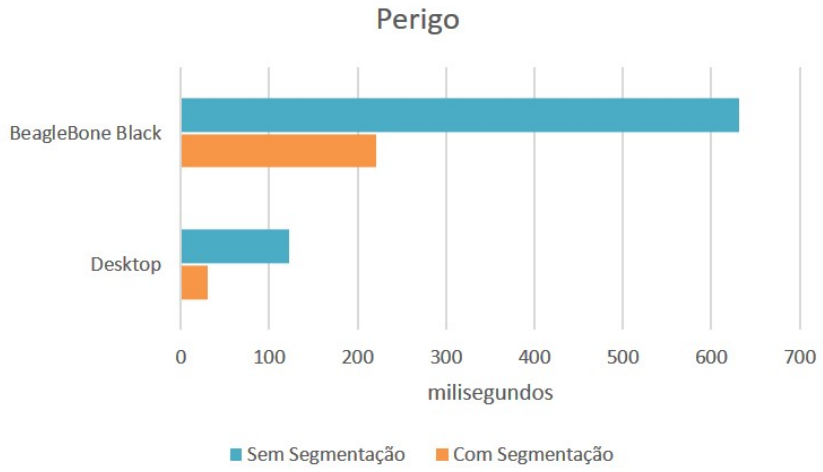
Tabela 9: Resultados do teste em video da fase de detecção realizado para as 3 classes de placas de trânsito

Classe Proibitória				
	Video 1	Video 2	Video 3	Video 4
Número de Instâncias	60	36	51	23
Acertos	57	33	51	20
Falsos Positivos	1	0	0	0
Taxa de Detecção	95%	91,66%	100%	86,95%

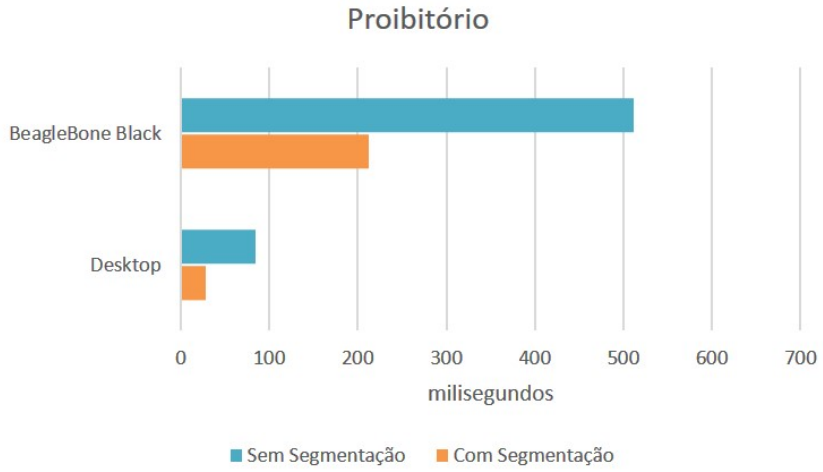
Classe Mandatória				
	Video 1	Video 2	Video 3	Video 4
Número de Instâncias	49	36	33	67
Acertos	49	35	29	63
Falsos Positivos	7	5	8	10
Taxa de Detecção	100%	97,22%	87,87%	94,02%

Classe Perigo				
	Video 1	Video 2	Video 3	Video 4
Número de Instâncias	43	43	27	44
Acertos	40	38	25	40
Falsos Positivos	0	0	0	0
Taxa de Detecção	93,02%	88,37%	92,59%	90,90%

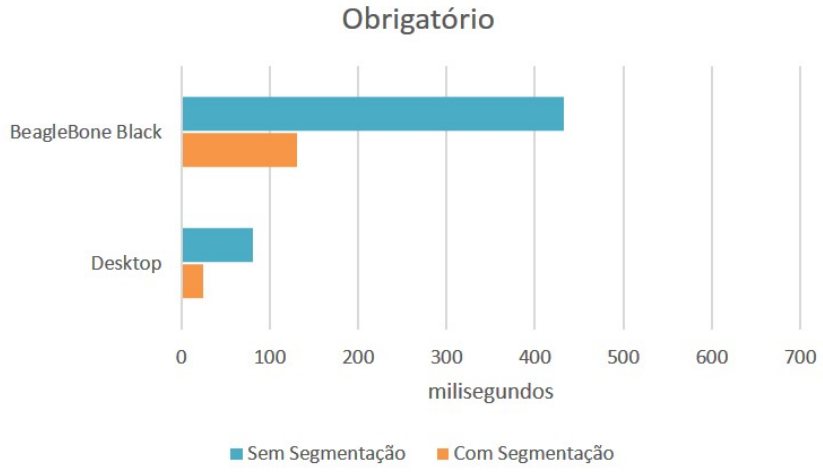
Os gráficos da Figura 60 demonstram os resultados de tempo obtidos para ambas plataformas, com e sem a utilização do processo de segmentação. É possível observar nos gráficos o impacto que a segmentação tem na redução do tempo de execução. Uma média 5 quadros por segundo foi obtida para o sistema embarcado com a aplicação da segmentação, enquanto que no *desktop* superou os 30 quadros por segundo. O tempo obtido para sistema embarcado permite a aplicação para problemas reais, uma vez que várias imagens são capturadas por segundo e isso torna a detecção mais confiável.



(a)



(b)



(c)

Figura 60: Tempo médio (ms) de execução da detecção no Desktop e na BeagleBone Black (Resolução de 640 × 480 *pixels*).

4.3 Resultados Reconhecimento

Os exemplos positivos fornecidos pela GTSRB foram utilizados para o treinamento de 43 classificadores binários seguindo o procedimento do fluxograma da Figura 48. Utilizou-se a validação cruzada para o treinamento e o valor do hiperparâmetro C encontrado foi 10. O sistema de reconhecimento foi testado nas imagens fornecidas pela competição e os resultados foram comparados com o *ground truth* também fornecido. Os erros e acertos foram contabilizados em uma matriz confusa, que pode ser observada na Figura 61. Ela fornece um panorama geral do desempenho dos classificadores e através dela é possível extrair outros resultados.

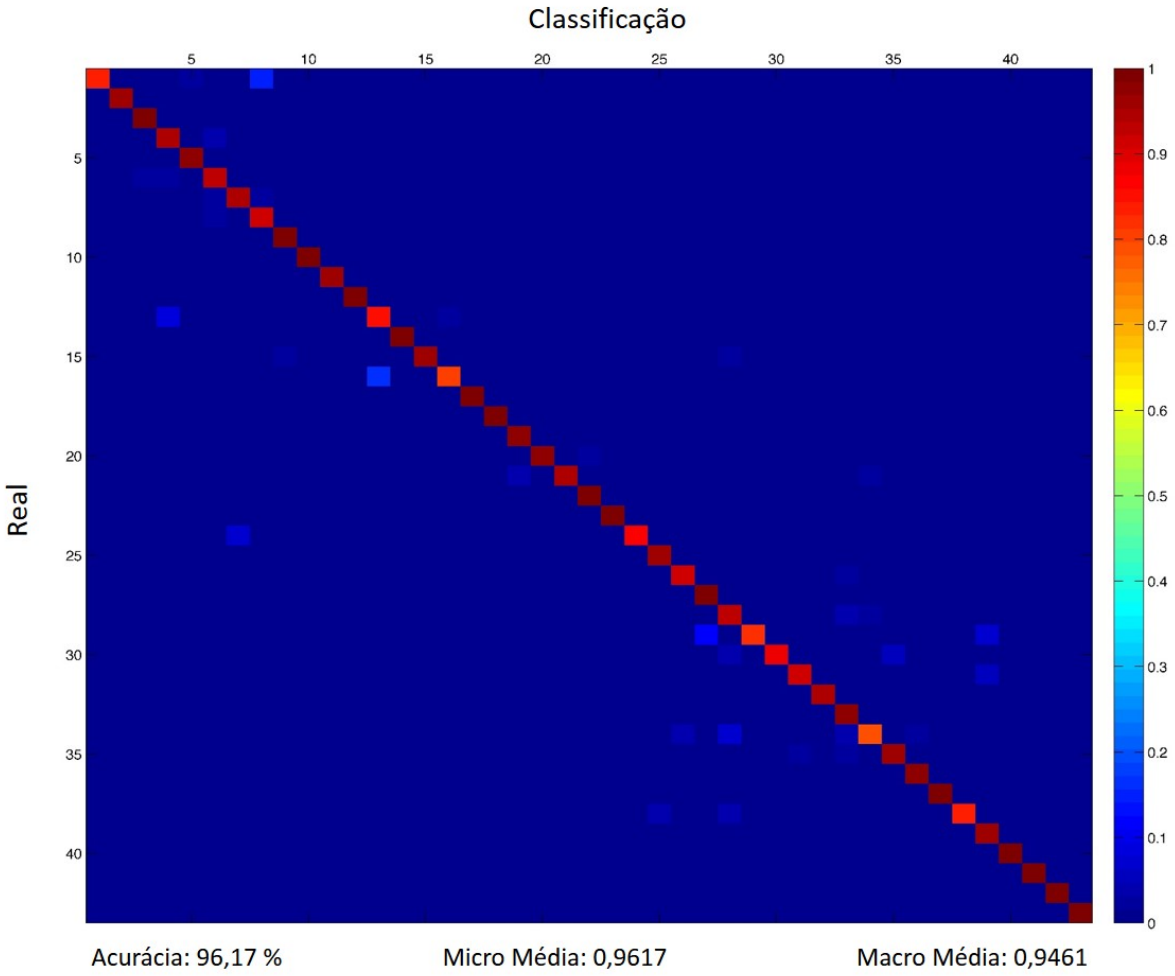


Figura 61: Matriz de confusão da fase de reconhecimento.

A matriz confusa foi mapeada para uma graduação de cores, representando valores de 0 a 1. Como pode-se observar, na Figura 61, a coloração próxima do vermelho se concentra na diagonal, que indica muitos acertos e a cor restante na matriz é próxima do azul, que indica poucos erros. Sendo assim o sistema atingiu uma acurácia ou CCR(*correct classification rate*) de 96,17%, que pode ser considerado um valor alto devido as dificuldades encontradas na base de dados. No Anexo A é possível observar todos valores da matriz confusa encontrados. A soma de cada uma das linhas corresponde ao número total de exemplos por classe. A Tabela 10 fornece um comparativo entre a metodologia proposta e outras técnicas na literatura.

Tabela 10: Comparativo da metodologia de reconhecimento proposta. Fonte: (STALLKAMP et al., 2012)

Método	CCR (%)
Comite de CNNs	99,46%
CNN de multi escala	98,31%
Proposto	**96,17** %
Random Florests	96,14%

No site oficial[1] da GTSRB é fornecido uma ferramenta para análise dos resultados. Para isso é necessário fornecer um arquivo, onde cada linha corresponde a predição emitida pelo sistema, a ser analisado, para cada uma das imagens da base de testes. As predições são valores de 0 a 42, que são o número de classes presentes na GTSRB. Através dessa ferramenta é possível se extrair a acurácia e visualizar erros e acertos do sistema. A Figura 62 demonstra o resultado obtido pelo sistema proposto, citado anteriormente. No Anexo C são apresentados exemplos de acertos (C.1) e erros (C.2) cometidos pela etapa de reconhecimento proposta.

No Anexo C.2 cada linha possui duas imagens extraídas da ferramenta de análise. Cada uma dessas imagens possui 3 colunas: a primeira (*class*) corresponde a classe correta, a segunda (*image*) a imagem da base de teste e a terceira (*Detection*) a classe reconhecida pela etapa de reconhecimento deste trabalho.

Outras métricas extraídas foram a micro média e a macro média. Estes valores são mais confiáveis do que a acurácia, uma vez que levam em consideração o desbalanceamento do número de exemplos de cada classe. Como pode ser visto, estes valores estão próximos da acurácia, o que indica que os números de exemplos de cada classe estão balanceados.

[1]GTSRB: http://benchmark.ini.rub.de/

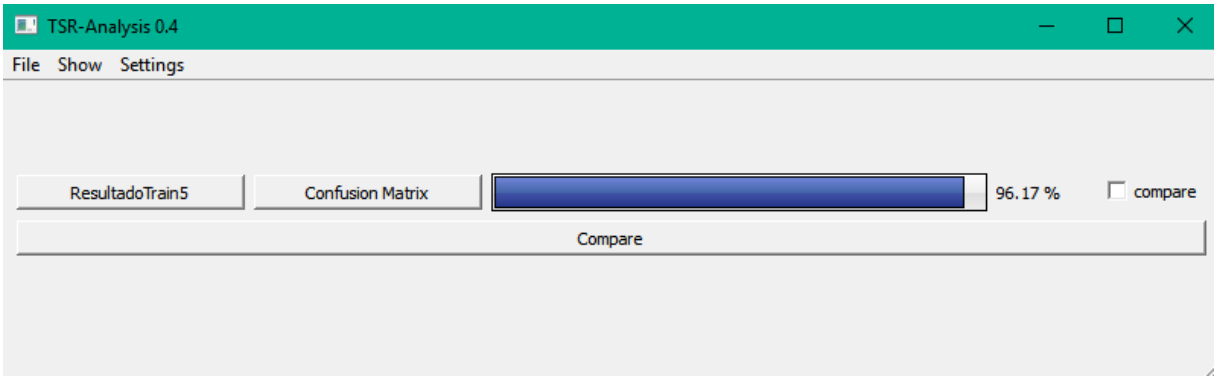


Figura 62: Matriz de confusão da fase de reconhecimento.

Os tempos foram extraídos somente para o sistema embarcada. Como pode-se observar no gráfico da Figura 63, o tempo total foi inferior a 8 milissegundos e as etapas que mais consumiram recursos foram as de extração de característica (HOG) e redução da dimensionalidade (LDA). Sendo assim o tempo de reconhecimento é ínfimo se comparado ao tempo de detecção tendo pouco impacto no tempo total, mesmo quando é necessário a classificação de mais de uma instância. O pouco tempo gasto para a execução de 43 classificadores SVM leva a crer que o sistema possa ser expandido para um número maior de classes sem comprometer muito o tempo total do reconhecimento.

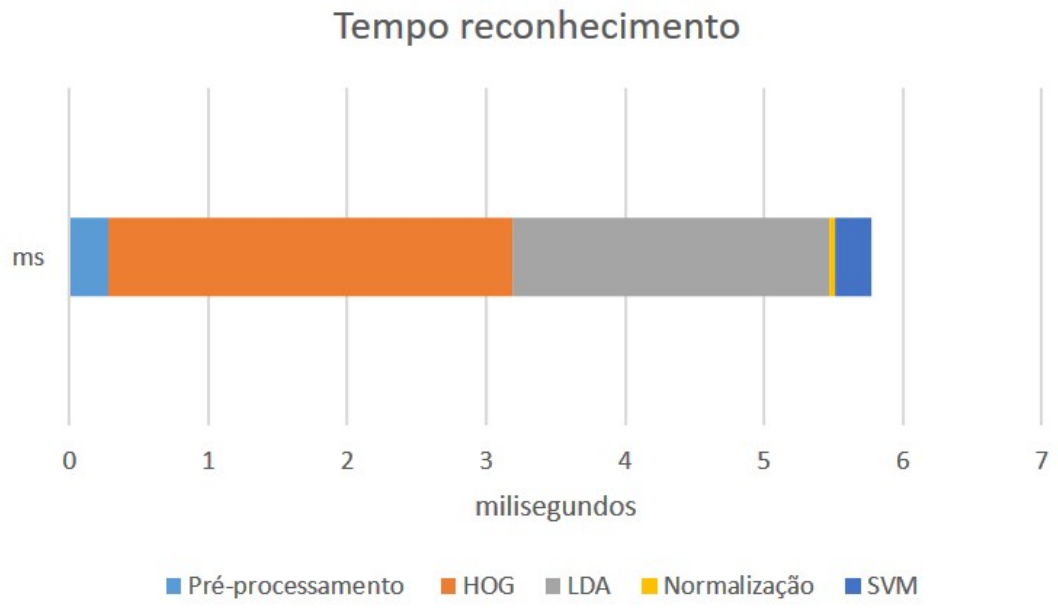


Figura 63: Tempo médio (ms) de execução do reconhecimento na BeagleBone Black.

5 Conclusão

Neste trabalho foi proposto uma metodologia de reconhecimento e detecção de placas de trânsito para um sistema embarcado de baixo custo. Na fase de detecção foi adicionada uma etapa de segmentação por cor, para redução do espaço de busca. Foram introduzidos algoritmos de baixa complexidade, o que adequa o sistema para o uso em tempo real em dispositivos embarcados. No desenvolvimento, utilizou-se, na implementação do sistema, a linguagem *C++* e as bibliotecas OpenCV e Qt.

A aplicação foi gerada para Linux embarcado, através da estratégia de compilação cruzada (*cross-compiling*) do código fonte para a arquitetura ARM. Além disso, foi ativada o uso de instruções NEON para aceleração do sistema de visão no hardware embarcado, em que foi realizado o processo de integração.

Os testes foram feitos em bases públicas GTSRB e GTSDB, conhecidas pela sua dificuldade de obtenção de bons resultados. Para ambas fases, detecção e reconhecimento, foram extraídas métricas para avaliação da acurácia e tempo de execução no sistema embarcado. Além disso, a utilização ou não da etapa de segmentação na detecção foi investigada, seu uso comprovou ser essencial para a redução de tempo computacional e aumento da precisão do sistema de visão.

Obteve-se taxas de sensitividade acima de 80%, em 10% de precisão, para as 3 classes testadas. Já o reconhecimento alcançou uma acurácia de 96,17%, em uma das bases publicas mais desafiadoras. Com relação ao tempo de execução no sistema embarcado, uma taxa aproximada de 5 *frames* por segundo foi atingido. Levando-se em consideração que o processamento não é usualmente exigido em cada *frame*, esta taxa de *frames* pode ser considerada próxima de tempo real, conforme Kumaraswamy et al. (2011). Sendo assim a metodologia apresenta resultados que permitem a sua aplicação em ambientes reais e integração em sistemas embarcado de baixo custo.

5.1 Trabalhos Futuros

O trabalho realizado atingiu os objetivos desejados. No entanto existem inúmeras possibilidades para melhorias ou continuidade do mesmo. A seguir são listados alguns trabalhos futuros que podem ser realizados:

- **Adição de um sistema de atenção visual** (YU et al., 2014). O processo de detecção de instâncias de placas de sinalização gasta a maior parte do processamento da BBB. A busca é feita de forma linear deslizando-se uma janela de tamanho fixo pela imagem. Esta estratégia não é considerada a mais eficiente, ou seja, é necessário a inclusão de um sistema que implemente uma estratégia de busca eficiente fazendo uso de técnicas relacionadas a atenção visual.

- **Treinamento e teste com uma base de placas de sinalização brasileira**. Poucas são as bases de dados disponibilizadas publicamente. Existe diferenças nas sinalizações de cada pais e seria interessante a criação de uma base de dados disponibilizada publicamente para comparar sistemas desenvolvidos exclusivamente para a sinalização brasileira.

- **Integrar o sistema de visão computacional a *GPU SGX530* da BBB**. Grande parte dos algoritmos utilizados são altamente paralelizáveis, devido a sua natureza. Tais algoritmos se tornam viáveis em tempo real no momento que fazem uso de dispositivos como GPUs. Dessa maneira uma implementação ou adaptação dos algoritmos escolhidos para este trabalho poderia acelerar o desempenho do sistema.

- **Adição de um sistema para rastreamento da placa de sinalização** detectada e reconhecida, economizando recursos computacionais. Em um sistema real uma mesma placa identificada aparece em uma sucessão de quadros. Dessa maneira não é necessário o reprocessamento dos *frames* seguintes, sendo que a placa possivelmente estará localizada em uma região próxima a originalmente identificada. Sendo assim pode-se aplicar algoritmos de rastreamento, economizando assim recursos computacionais.

- **Aplicar técnicas para realizar treinamento da etapa de reconhecimento de maneira online**, para que o sistema evolua na medida em que tenha acesso a novas instâncias de placas. O sistema treinado neste trabalho não evolui com os erros realizados ou novas placas identificadas.

APÊNDICE A – Ferramentas de Desenvolvimento

Neste apêndice serão apresentados as principais ferramentas utilizadas no desenvolvimento do sistema de visão computacional proposto no Capítulo 3. Optou-se por trabalhar com ferramentas *open source* para redução dos custos de desenvolvimento do sistema.

A.1 OpenCV

OpenCV [1] (*Open Source Computer Vision*) é uma biblioteca de visão computacional projetada para eficiência computacional e com um foco muito forte em aplicações de tempo-real. Desenvolvida pelo centro de pesquisa da Intel atualmente é suportada por *Willow Garage* e *Itseez*.

Foi lançado sob licença BSD, sendo assim gratuito tanto para uso comercial quanto acadêmico. É um projeto *cross-plataform*, assim, disponível para diversas plataformas entre elas: Linux, Windows, Mac OSX, Android e IOS.

É escrita em linguagem *C* e *C++* otimizada, mas possui compatibilidade com outras linguagens dentre elas: Python, Java, Matlab®, *C#*, entre outras. Através da biblioteca OpenCL pode-se tirar proveito da aceleração de hardware em sistemas de computação heterogêneos, sendo dessa maneira possível aproveitar o poder computacional das modernas GPUs. Entre as áreas de aplicação pode-se citar:

- Sistema de reconhecimento facial.
- Reconhecimento de gestos.
- Realidade aumentada.
- Rastreamento de movimentos.

[1] Site Oficial: http://opencv.org/

- Segmentaçao e reconhecimento de imagens.
- Extraçao de características 2D e 3D.
- Visão estéreo.
- Identificação de objetos.
- Robótica.

A biblioteca está dividida em módulos, que executam determinados tipos de funcionalidades. Os módulos e suas funcionalidades são apresentados a seguir:

- *Core* - Estruturas de dados principais, tipo de dados e gerenciamento de memória.
- *Imgproc* - Filtragem de imagem, transformaçoes geométricas, análise de estruturas e formas.
- *Highgui* - Interface de usuário, leitura e escrita de imagens e vídeo.
- *Video* - Analise de movimento e rastreamento de objeto em vídeo.
- *Calib3d* - Calibração de câmera e reconstrução 3D para múltiplas vistas.
- *Features2d* - Extração de características, descrição e correspondência
- *Objdetect* - Detecção de objetos utilizando *cascade* e classificadores HOG.
- *ML* - Modelos estatísticos e algoritmos de classificação comuns a visão computacional.
- *Flann* - Buscas rápidas em espaços de características hiperdimensionais.
- *GPU* - Paralelização de algoritmos selecionados para execução rápida em GPUs.
- *NonFree* - Implementação de algoritmos patenteados.

Esses módulos são versáteis o suficiente para resolver a maior parte dos problemas de visão computacional. Mesmo no caso de uma pesquisa em que se queira criar novos algoritmos a OpenCV fornece uma arquitetura, gerenciamento de memória e suporte a GPUs que irão auxiliar no desenvolvimento. Facilitando, assim, o desenvolvimento e teste durante a fase de pesquisa de quais metodologias se aplicar para determinado tipo de problema.

A.2 QT

O Qt [2] é um *framework* de aplicaçao *cross-plataform* utilizada de forma abrangente em aplicações que necessitam executar em hardwares ou sistemas operacionais variados, com pouca ou nenhuma mudança no código fonte, e com velocidade de um aplicativo nativo. Atualmente o *Qt* é desenvolvido por *Qt Company* e *Qt Project* sob governança *open-source*.

Ele está disponível com licença tanto comercial quando *open-source* GPL v3, LGPL v3 e LGPL v2. De acordo com esses termos somente é possível a distribuição de um produto comercial caso o código fonte seja disponibilizado de forma aberta ou o produto faça uso de bibliotecas compartilhadas (seção 4.d.1 da LGPL [3]).

O *framework* é utilizado principalmente no desenvolvimento de aplicações com interface gráfica de usuário, conhecida com GUI (*Graphical User Interface*). Apesar disso, programadores podem desenvolver aplicativos de linha de comando ou terminais para servidores. As principais características do *framework* são:

- Sistema intuitivo de classes em *C++*.
- Portabilidade entre sistemas operacionais desktop e embarcados.
- Ferramentas de desenvolvimento integras com IDE *cross-plataform*.
- Alto desempenho em tempo de execução e pequena contribuição no tamanho do executável em sistemas embarcados.

O *Qt* possui seu próprio conjunto de ferramentas para desenvolvimento de aplicações *cross-plataform* e que incluem o *QT Creator* para desenvolvimento, compilação e projeto de interfaces gráficas. O *Qt creator* está disponível para diversas plataformas e através dele é possível se configurar um ambiente de desenvolvimento para sistemas embarcados.

A linguagem utilizada para o desenvolvimento é *C++* possuindo algumas extensões, tal como *signals* e *slots*, que simplificam o tratamento de eventos. Suporta uma vasta quantidade de compiladores entre eles: GCC *C++* e *Visual Studio*. Sua característica *cross-plataform* permite ser executado na maior parte das plataformas *desktop* ou móveis. Algumas das plataformas suportadas atualmente são listadas abaixo:

[2] Site Oficial: https://www.qt.io/
[3] Documento LGPL: https://www.gnu.org/licenses/gpl-3.0.html

- Android
- IOS
- Linux Embarcado
- Windows CE
- Windows RT
- Windows
- OS X
- Entre outros

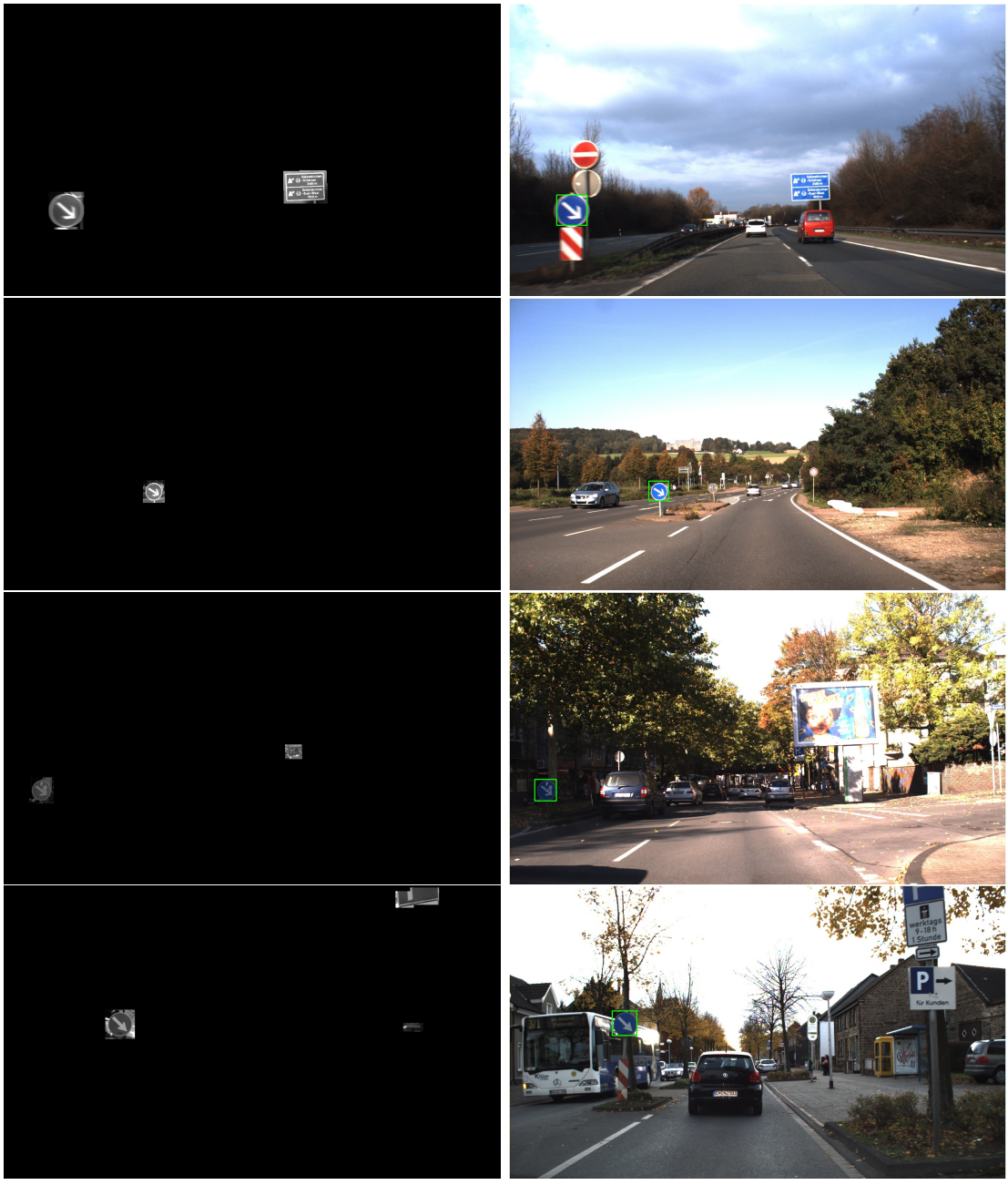
Com o *Qt* é possível se escrever aplicação apenas uma vez e implantar (*deploy*) em vários sistemas operacionais sem a necessidade de reescrever código. O *framework* é dividida em módulos, projetados para desempenhar determinadas tarefas. Abaixo são descritas as principais funcionalidades de alguns módulos disponíveis no pacote *Qt Essentials*:

- *Qt Core* - Contém classes utilizadas por outros módulos, sistema de meta-objetos, estruturas de dados, algoritmos, concorrência e threads, sistemas de eventos, containers, plug-ins e recursos para I/O.
- *Qt GUI* - Módulo central para interface gráfica de usuário.
- *Qt Network* - Camada de abstraçao de rede. Contém diversos protocolos de comunicação.
- *Qt SQL* - Contém classes para integração com banco de dados usando SQL.
- *Qt QML* - Módulo para linguagens QML e *Javascript*.
- *Qt Quick* - Módulo para aplicações GUI escritos em QML 2.
- *Qt Multimidia* - Classes para áudio, vídeo, radio e funcionalidades de câmera.

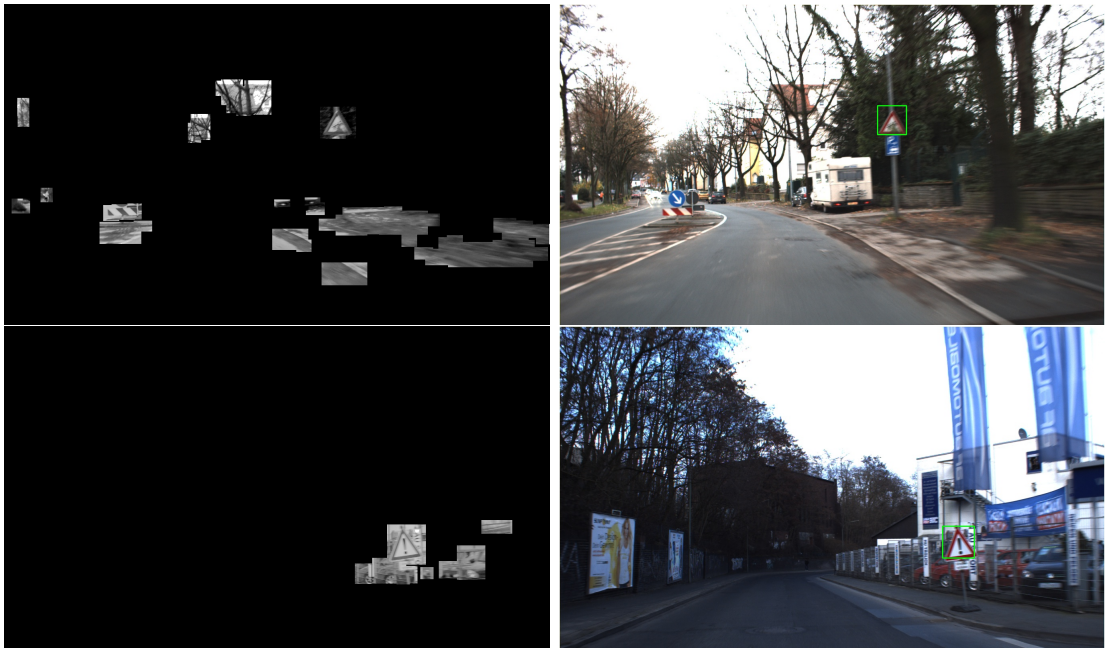
ANEXO A – Valores da Matriz Confusa

50	0	0	0	1	0	0	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	697	11	2	0	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0
0	5	739	3	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	5	424	0	18	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
0	2	3	2	648	1	3	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	4	11	10	0	588	6	3	0	3	0	1	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	2	1	0	1	13	422	8	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	3	6	6	2	8	3	412	5	1	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	475	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	2	0	0	0	651	0	0	0	0	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	3	3	0	0	0	0	0	0	203	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	0	1	1	0	148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	13	2	1	0	0	0	0	0	0	128	0	0	4	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	15	0	1	72	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	209	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	119	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	3	0	0	0	0	1	0	0	0	0	0	1	0	0	0	381	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	118	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	57	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	685	0	0	0	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	78	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	404	3	3	1	1	0	0	0	1	0	0	0	1	1	0	1	2	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	359	2	0	0	1	1	0	7	6	0	5	2	6	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	83	0	0	0	0	3	2	1	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	74	0	0	0	0	0	0	0	0	0	6	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	5	1	106	0	0	0	0	6	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	138	1	0	0	0	0	0	2	8	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	85	0	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	1	0	0	0	472	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	2	7	0	13	0	0	0	0	7	143	1	5	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	58	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	147	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	89	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	7	0	1	7	0	0	2	0	1	1	1	1	1	126	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	1	1	2	0	0	1	0	0	0	0	0	261	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	688	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	718	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	266	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	357

ANEXO B – Exemplos de Detecção




STOP
STOP



ANEXO C – Exemplos de Reconhecimento

C.1 Acertos


STOP


100

C.2 Erros


Class
Image
Detection
20
120
20
70
20
120
20
120
Class
Image
Detection
60
80
60
80
60
50
60
80
Class
Image
Detection
60
80
70
50
70
50
70
100
Class
Image
Detection
80
60
80
60
80
80
Class
Image
Detection
100
100
100
30
100
120
Class
Image
Detection
120
60
120
120
60
120
80


Class
Image
Detection
Class
Image
Detection
Class
Image
Detection
Class
Image
Detection
STOP
100
80
80
80

ANEXO D – Análise em vídeo da Detecção

D.1 Proibitório

Figura 72: Proibitório - Vídeo 1

Figura 73: Proibitório - Vídeo 2

Figura 74: Proibitório - Vídeo 3

Figura 75: Proibitório - Vídeo 4

D.2 Mandatório

Figura 76: Mandatório - Vídeo 1

Figura 77: Mandatório - Vídeo 2

Figura 78: Mandatório - Vídeo 3

Figura 79: Mandatório - Vídeo 4

D.3 Perigo

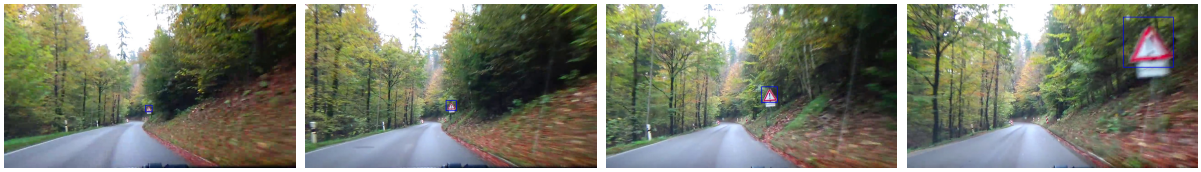


Figura 80: Perigo - Vídeo 1

Figura 81: Perigo - Vídeo 2

Figura 82: Perigo - Vídeo 3

Figura 83: Perigo - Vídeo 4

Referências

AGUIRRE-DOBERNACK, N.; GUZMAN-MIRANDA, H.; AGUIRRE, M. Implementation of a machine vision system for real-time traffic sign recognition on FPGA. In: *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*. [S.l.: s.n.], 2013. p. 2285–2290.

ALMEIDA, R. de. *Programação de Sistemas Embarcados*. Edição: 1ª. [S.l.]: Elsevier, 2016. ISBN 978-85-352-8518-5.

ANTHONY, G.; GREGG, H.; TSHILIDZI, M. Image classification using SVMs: One-against-one vs one-against-all. *arXiv preprint arXiv:0711.2914*, 2007.

ARAÚJO, R. L. C. et al. Classificação de pedestres usando câmera e sensor LIDAR. *X Simpósio Brasileiro de Automação Inteligente (SBAI)*, 2011.

ARM. *NEON - ARM*. 2016. Acessado em 19/10/2016. Disponível em: <http://www-.arm.com/products/processors/technologies/neon.php>.

ASCH, V. V. Macro-and micro-averaged evaluation measures. 2013.

BAHLMANN, C.; HAASDONK, B.; BURKHARDT, H. Online handwriting recognition with support vector machines-a kernel approach. In: *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on*. [S.l.]: IEEE, 2002. p. 49–54.

BELONGIE, S.; MALIK, J.; PUZICHA, J. Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, v. 24, n. 4, p. 509–522, 2002.

BOSER, B. E.; GUYON, I. M.; VAPNIK, V. N. A training algorithm for optimal margin classifiers. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. [S.l.]: ACM, 1992. p. 144–152.

BROGGI, A. et al. Real Time Road Signs Recognition. In: *2007 IEEE Intelligent Vehicles Symposium*. [S.l.: s.n.], 2007. p. 981–986.

CAO, L. et al. A comparison of PCA, KPCA and ICA for dimensionality reduction in support vector machine. *Neurocomputing*, v. 55, n. 1-2, p. 321–336, set. 2003. ISSN 09252312.

CAWLEY, G. C.; TALBOT, N. L. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, v. 11, n. Jul, p. 2079–2107, 2010.

CHAN, R. H.; HO, C.-W.; NIKOLOVA, M. Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization. 2004.

CHANG, C.-C.; LIN, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, v. 2, n. 3, p. 27, 2011.

CHEN, T.; LU, S. Accurate and Efficient Traffic Sign Detection Using Discriminative AdaBoost and Support Vector Regression. *IEEE Transactions on Vehicular Technology*, v. 65, n. 6, p. 4006–4015, jun. 2016. ISSN 0018-9545.

CHEN, Z. et al. A GPU-based real-time traffic sign detection and recognition system. In: *2014 IEEE Symposium on Computational Intelligence in Vehicles and Transportation Systems (CIVTS)*. [S.l.: s.n.], 2014. p. 1–5.

CIRESAN, D. et al. A committee of neural networks for traffic sign classification. In: *The 2011 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2011. p. 1918–1921.

CLARK, A. F.; CLARK, C. Performance characterization in computer vision a tutorial. *) N (Eds.): FBook performance characterization in computer vision a tutorial/(Citeseer, 1999, edn.)*, 1999.

COLEY, G. Beaglebone black system reference manual. *Texas Instruments, Dallas*, 2013.

CORMEN, T. H. et al. Introduction to algorithms second edition. 2001.

CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, v. 20, n. 3, p. 273–297, 1995.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. [S.l.]: IEEE, 2005. v. 1, p. 886–893.

DÉNIZ, O. et al. Smile detection for user interfaces. In: *International Symposium on Visual Computing*. [S.l.]: Springer, 2008. p. 602–611.

DING, D.; YOON, J.; LEE, C. Traffic sign detection and identification using SURF algorithm and GPGPU. In: *SoC Design Conference (ISOCC), 2012 International*. [S.l.: s.n.], 2012. p. 506–508.

ETHEM, A. Introduction to machine learning. *Cambridge MA*, 2004.

FISHER, R. A. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, v. 7, n. 2, p. 179–188, set. 1936. ISSN 2050-1439.

FREUND, Y.; SCHAPIRE, R. E. A desicion-theoretic generalization of on-line learning and an application to boosting. In: *European Conference on Computational Learning Theory*. [S.l.]: Springer, 1995. p. 23–37.

F.R.S, K. P. LIII. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 6*, v. 2, n. 11, p. 559–572, 1901.

G, S. H. S.; PATIL, C. M. An approach towards efficient detection and recognition of traffic signs in videos using neural networks. In: *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. [S.l.: s.n.], 2016. p. 456–459.

GE, K.-B.; WEN, J.; FANG, B. Adaboost algorithm based on MB-LBP features with skin color segmentation for face detection. In: *2011 International Conference on Wavelet Analysis and Pattern Recognition*. [S.l.]: IEEE, 2011. p. 40–43.

GLAVTCHEV, V. et al. Feature-based speed limit sign detection using a graphics processing unit. In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. [S.l.: s.n.], 2011. p. 195–200.

GOMEZ-MORENO, H. et al. Goal Evaluation of Segmentation Algorithms for Traffic Sign Recognition. *IEEE Transactions on Intelligent Transportation Systems*, v. 11, n. 4, p. 917–930, dez. 2010. ISSN 1524-9050.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. [S.l.]: Prentice Hall, 2008. ISBN 978-0-13-168728-8.

GRABNER, H. et al. On-line boosting-based car detection from aerial images. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 63, n. 3, p. 382–396, 2008.

GRANA, C.; BORGHESANI, D.; CUCCHIARA, R. Connected component labeling techniques on modern architectures. In: *International Conference on Image Analysis and Processing*. [S.l.]: Springer, 2009. p. 816–824.

GREENHALGH, J.; MIRMEHDI, M. Real-Time Detection and Recognition of Road Traffic Signs. *IEEE Transactions on Intelligent Transportation Systems*, v. 13, n. 4, p. 1498–1506, dez. 2012. ISSN 1524-9050.

GRITTI, T. et al. Local features based facial expression recognition with face registration errors. In: . [S.l.]: IEEE, 2008. p. 1–8. ISBN 978-1-4244-2153-4.

HALLINAN, C. *Embedded Linux Primer: A Practical, Real-World Approach*. [S.l.]: Pearson Education India, 2007.

HAN, Y. et al. Hardware/Software Co-Design of a Traffic Sign Recognition System Using Zynq FPGAs. *Electronics*, v. 4, n. 4, p. 1062–1089, dez. 2015. ISSN 2079-9292.

HE, N.; HUANG, H.-W.; WOLTMAN, B. D. The Use of BeagleBone Black Board in Engineering Design and Development. 2014.

HOUBEN, S. et al. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In: *Neural Networks (IJCNN), The 2013 International Joint Conference on*. [S.l.]: IEEE, 2013. p. 1–8.

HSU, C.-W. et al. A practical guide to support vector classification. 2003.

HSU, C.-W.; LIN, C.-J. A comparison of methods for multiclass support vector machines. *Neural Networks, IEEE Transactions on*, v. 13, n. 2, p. 415–425, 2002.

HUTCHISON, M. *Brookside traffic lights*. 2016. Acessado em 12/10/2016. Disponível em: <https://insidebrookside.wordpress.com/tag/brookside-traffic-lights/>.

INSTRUMENTS, T. AM335x Sitara^{TM} Processors. *URL http://www. ti. com/-lit/ds/symlink/am3359. pdf*, 2013.

JAHNE, B. *Computer Vision and Applications: A Guide for Students and Practitioners,Concise Edition*. [S.l.]: Academic Press, 2000. ISBN 978-0-08-050262-5.

JAIN, R.; KASTURI, R.; SCHUNCK, B. G. *Machine Vision*. New York, NY, USA: McGraw-Hill, Inc., 1995. ISBN 0-07-032018-7.

JASSIM, F. A. Kriging interpolation filter to reduce high density salt and pepper noise. *arXiv preprint arXiv:1302.1300*, 2013.

JUNG, S. et al. Real-time Traffic Sign Recognition system with deep convolutional neural network. In: *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. [S.l.: s.n.], 2016. p. 31–34.

KASSANI, P. H.; HYUN, J.; KIM, E. Application of soft Histogram of Oriented Gradient on traffic sign detection. In: *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. [S.l.: s.n.], 2016. p. 388–392.

KLAUER, S. G. et al. Distracted Driving and Risk of Road Crashes among Novice and Experienced Drivers. *New England Journal of Medicine*, v. 370, n. 1, p. 54–59, jan. 2014. ISSN 0028-4793, 1533-4406.

KUMARASWAMY, R. et al. SVM based classification of traffic signs for realtime embedded platform. In: *Advances in Computing and Communications*. [S.l.]: Springer, 2011. p. 339–348.

LAFUENTE-ARROYO, S. et al. A decision support system for the automatic management of keep-clear signs based on support vector machines and geographic information systems. *Expert Systems with Applications*, v. 37, n. 1, p. 767–773, jan. 2010. ISSN 09574174.

LARSSON, F.; FELSBERG, M. Using Fourier descriptors and spatial models for traffic sign recognition. In: *Scandinavian Conference on Image Analysis*. [S.l.]: Springer, 2011. p. 238–249.

LI, T.; ZHU, S.; OGIHARA, M. Using discriminant analysis for multi-class classification: An experimental investigation. *Knowledge and information systems*, v. 10, n. 4, p. 453–472, 2006.

LIAO, S. et al. Learning multi-scale block local binary patterns for face recognition. In: *International Conference on Biometrics*. [S.l.]: Springer, 2007. p. 828–837.

LIENHART, R.; MAYDT, J. An extended set of haar-like features for rapid object detection. In: *Image Processing. 2002. Proceedings. 2002 International Conference on*. [S.l.]: IEEE, 2002. v. 1, p. I–900.

LIU, C.; CHANG, F.; CHEN, Z. Rapid Multiclass Traffic Sign Detection in High-Resolution Images. *IEEE Transactions on Intelligent Transportation Systems*, v. 15, n. 6, p. 2394–2403, dez. 2014. ISSN 1524-9050.

LORENA, A. C.; CARVALHO, A. C. de. Uma introdução às support vector machines. *Revista de Informática Teórica e Aplicada*, v. 14, n. 2, p. 43–67, 2007.

LOWE, D. G. Object recognition from local scale-invariant features. In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. [S.l.]: Ieee, 1999. v. 2, p. 1150–1157.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, v. 60, n. 2, p. 91–110, 2004.

MALDONADO-BASCON, S. et al. Road-Sign Detection and Recognition Based on Support Vector Machines. *IEEE Transactions on Intelligent Transportation Systems*, v. 8, n. 2, p. 264–278, jun. 2007. ISSN 1524-9050.

MARTINEZ, A. M.; KAK, A. C. PCA versus LDA. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 23, n. 2, p. 228–233, fev. 2001. ISSN 0162-8828.

MATHIAS, M. et al. Traffic sign recognition—How far are we from the solution? In: *Neural Networks (IJCNN), The 2013 International Joint Conference on*. [S.l.]: IEEE, 2013. p. 1–8.

MCCANE, B. et al. *Optimizing Cascade Classifiers*. [S.l.]: Unpublished, 2005.

MENESES, P. R.; ALMEIDA, T. de. Introdução ao processamento de imagens de sensoriamento remoto. *Brasília: UNB/CNPq*, 2012.

MIRZA, B.; LIN, Z. One-vs-all for class imbalance learning. In: *Information, Communications and Signal Processing (ICICS) 2013 9th International Conference on*. [S.l.]: IEEE, 2013. p. 1–5.

MOGELMOSE, A.; TRIVEDI, M.; MOESLUND, T. Vision-Based Traffic Sign Detection and Analysis for Intelligent Driver Assistance Systems: Perspectives and Survey. *IEEE Transactions on Intelligent Transportation Systems*, v. 13, n. 4, p. 1484–1497, dez. 2012. ISSN 1524-9050.

MÜLLER, M. et al. Design of an automotive traffic sign recognition system targeting a multi-core SoC implementation. In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*. [S.l.: s.n.], 2010. p. 532–537.

NAIR, M. S.; REVATHY, K.; TATAVARTI, R. Removal of salt-and pepper noise in images: A new decision-based algorithm. In: *Proceedings of the International Multi Conference of Engineers and Computer Scientists*. [S.l.]: Citeseer, 2008. v. 2.

OHGUSHI, K.; HAMADA, N. Traffic sign recognition by Bags of features. In: *TENCON 2009 - 2009 IEEE Region 10 Conference*. [S.l.: s.n.], 2009. p. 1–6.

OJALA, T.; PIETIKAINEN, M.; HARWOOD, D. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In: , *Proceedings of the 12th IAPR International Conference on Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision Amp; Image Processing*. [S.l.: s.n.], 1994. v. 1, p. 582–585 vol.1.

OSUNA, E.; FREUND, R.; GIROSIT, F. Training support vector machines: An application to face detection. In: *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. [S.l.]: IEEE, 1997. p. 130–136.

OTTLEY, A. *ECG Compression for Holter Monitoring*. Tese (Doutorado) — Citeseer, 2007.

ÖZGÜR, A.; ÖZGÜR, L.; GÜNGÖR, T. Text categorization with class-based and corpus-based keyword selection. In: *International Symposium on Computer and Information Sciences*. [S.l.]: Springer, 2005. p. 606–615.

PAPAGEORGIOU, C. P.; OREN, M.; POGGIO, T. A general framework for object detection. In: *Computer Vision, 1998. Sixth International Conference on*. [S.l.]: IEEE, 1998. p. 555–562.

PARK, J. et al. A 92-mW Real-Time Traffic Sign Recognition System With Robust Illumination Adaptation and Support Vector Machine. *IEEE Journal of Solid-State Circuits*, v. 47, n. 11, p. 2711–2723, nov. 2012. ISSN 0018-9200, 1558-173X.

PEARSON, K. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, v. 185, p. 71–110, 1894.

PHOTOS Public Domain. 2016. Acessado em 12/10/2016. Disponível em: <http://www.photos-public-domain.com/2010/10/29/stop-sign/>.

POWERS, D. M. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. 2011.

PRADO, S. *Embedded Labworks - Linux embarcado*. 2016. Acessado em 19/10/2016. Disponível em: <https://e-labworks.com/treinamentos/linux-embarcado/>.

QUEIROZ, J. E. R. de; GOMES, H. M. Introdução ao Processamento Digital de Imagens. *RITA*, v. 13, n. 2, p. 11–42, 2006.

RAGHAVAN, P.; LAD, A.; NEELAKANDAN, S. *Embedded Linux System Design and Development*. [S.l.]: CRC Press, 2005.

RAMALHO, G. L. B. *Análise de Imagens Por Meio Da Matriz de Interdependência E Da Transformação Estrutural Multiescala*. Tese (Doutorado) — Universidade Federal do Cerá, 2013.

RAO, C. R. The Utilization of Multiple Measurements in Problems of Biological Classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, v. 10, n. 2, p. 159–203, 1948. ISSN 0035-9246.

REN, F. et al. General traffic sign recognition by feature matching. In: *Image and Vision Computing New Zealand, 2009. IVCNZ '09. 24th International Conference*. [S.l.: s.n.], 2009. p. 409–414.

ROSENFELD, A.; PFALTZ, J. L. Sequential operations in digital picture processing. *Journal of the ACM (JACM)*, v. 13, n. 4, p. 471–494, 1966.

RYCHETSKY, M. *Algorithms and Architectures for Machine Learning Based on Regularized Neural Networks and Support Vector Approaches*. [S.l.]: Shaker, 2001.

SALLAH, S.; HUSSIN, F.; YUSOFF, M. Road sign detection and recognition system for real-time embedded applications. In: *2011 International Conference on Electrical, Control and Computer Engineering (INECCE)*. [S.l.: s.n.], 2011. p. 213–218.

SCHAPIRE, R. E. The strength of weak learnability. *Machine learning*, v. 5, n. 2, p. 197–227, 1990.

SCHOLKOPFT, B.; MULLERT, K.-R. Fisher discriminant analysis with kernels. *Neural networks for signal processing IX*, v. 1, n. 1, p. 1, 1999.

SERMANET, P.; LECUN, Y. Traffic sign recognition with multi-scale Convolutional Networks. In: *The 2011 International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2011. p. 2809–2813.

SHAH, M. *FUNDAMENTALS OF COMPUTER VISION1*. [S.l.: s.n.], 1997.

SHAOUT, A.; COLELLA, D.; AWAD, S. Advanced driver assistance systems-past, present and future. In: *Computer Engineering Conference (ICENCO), 2011 Seventh International*. [S.l.]: IEEE, 2011. p. 72–82.

SHAPIRO, L.; STOCKMAN, G. *Computer Vision*. [S.l.]: Prentice Hall, 2001. ISBN 978-0-13-030796-5.

SOLOMON, C.; BRECKON, T. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. [S.l.]: John Wiley & Sons, 2011. ISBN 978-1-119-95700-3.

SRIRAM, S.; ILLURI, B. Real Time Smile Detection using Haar Classifiers on SoC. *International Journal of Computer Applications*, v. 104, n. 10, 2014.

STALLKAMP, J. et al. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, v. 32, p. 323–332, 2012.

STRAYER, D. L.; DREWS, F. A.; CROUCH, D. J. A comparison of the cell phone driver and the drunk driver. *Human factors: The journal of the human factors and ergonomics society*, v. 48, n. 2, p. 381–391, 2006.

SZELISKI, R. *Computer Vision: Algorithms and Applications*. [S.l.]: Springer Science & Business Media, 2010.

TIAN, B. et al. Robust traffic sign detection in complex road environments. In: *2016 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. [S.l.: s.n.], 2016. p. 1–5.

TIMOFTE, R.; ZIMMERMANN, K.; GOOL, L. V. Multi-view traffic sign detection, recognition, and 3d localisation. *Machine Vision and Applications*, v. 25, n. 3, p. 633–647, 2014.

VAPNIK, V. N.; CHERVONENKIS, A. Y. On the uniform convergence of relative frequencies of events to their probabilities. In: *Measures of Complexity*. [S.l.]: Springer, 2015. p. 11–30.

VINK, J. et al. Efficient nucleus detector in histopathology images. *Journal of microscopy*, v. 249, n. 2, p. 124–135, 2013.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. [S.l.]: IEEE, 2001. v. 1, p. I–511.

VIOLA, P.; JONES, M. J. Robust real-time face detection. *International journal of computer vision*, v. 57, n. 2, p. 137–154, 2004.

VUKADINOVIC, D.; PANTIC, M. Fully automatic facial feature point detection using Gabor feature based boosted classifiers. In: *2005 IEEE International Conference on Systems, Man and Cybernetics*. [S.l.]: IEEE, 2005. v. 2, p. 1692–1698.

WAHYONO; JO, K. H. A comparative study of classification methods for traffic signs recognition. In: *2014 IEEE International Conference on Industrial Technology (ICIT)*. [S.l.: s.n.], 2014. p. 614–619.

WAHYONO et al. Traffic sign recognition system for autonomous vehicle using cascade SVM classifier. In: *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society*. [S.l.: s.n.], 2014. p. 4081–4086.

WANG, X.; HAN, T. X.; YAN, S. An HOG-LBP human detector with partial occlusion handling. In: *2009 IEEE 12th International Conference on Computer Vision*. [S.l.]: IEEE, 2009. p. 32–39.

WU, B. et al. Fast rotation invariant multi-view face detection based on real adaboost. In: *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*. [S.l.]: IEEE, 2004. p. 79–84.

WU, T.-F.; LIN, C.-J.; WENG, R. C. Probability estimates for multi-class classification by pairwise coupling. *The Journal of Machine Learning Research*, v. 5, p. 975–1005, 2004.

YAKIMOV, P.; FURSOV, V. Traffic Signs Detection and tracking using modified Hough transform. In: *2015 12th International Joint Conference on E-Business and Telecommunications (ICETE)*. [S.l.: s.n.], 2015. v. 05, p. 22–28.

YU, H.; YANG, J. A direct LDA algorithm for high-dimensional data—with application to face recognition. *Pattern recognition*, v. 34, n. 10, p. 2067–2070, 2001.

YU, Y. et al. A visual attention based method for detecting traffic signs of interest. In: *Information and Automation (ICIA), 2014 IEEE International Conference on*. [S.l.]: IEEE, 2014. p. 290–294.

ZAKLOUTA, F.; STANCIULESCU, B. Real-Time Traffic-Sign Recognition Using Tree Classifiers. *IEEE Transactions on Intelligent Transportation Systems*, v. 13, n. 4, p. 1507–1514, dez. 2012. ISSN 1524-9050.

ZHANG, L. et al. Face detection based on multi-block lbp representation. In: *International Conference on Biometrics*. [S.l.]: Springer, 2007. p. 11–18.

ZHU, Q. et al. Fast human detection using a cascade of histograms of oriented gradients. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. [S.l.]: IEEE, 2006. v. 2, p. 1491–1498.

ZUIDERVELD, K. Contrast limited adaptive histogram equalization. In: *Graphics Gems IV*. [S.l.]: Academic Press Professional, Inc., 1994. p. 474–485.