UNIVERSIDADE FEDERAL DE SANTA CATARINA
CTC - CENTRO TECNOLÓGICO DA UFSC
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Luiz Eduardo Zis

Segmentação de objetos em imagens RGB-D: Um comparativo entre abordagens tradicionais e baseadas em aprendizado de máquina

Florianópolis
2020

Luiz Eduardo Zis

Segmentação de objetos em imagens RGB-D: Um comparativo entre abordagens tradicionais e baseadas em aprendizado de máquina

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Engenharia de Automação e Sistemas.
Orientador: Prof. Marcelo Ricardo Stemmer, Dr.

Florianópolis
2020

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Zis, Luiz Eduardo
Segmentação de objetos em imagens RGB-D : Um comparativo entre abordagens tradicionais e baseadas em aprendizado de máquina / Luiz Eduardo Zis ; orientador, Marcelo Ricardo Stemmer, 2020.
122 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2020.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Segmentação de Objetos. 3. RGB-D. 4. Aprendizado de Máquina. 5. Robótica Móvel. I. Stemmer, Marcelo Ricardo. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

Luiz Eduardo Zis

Segmentação de objetos em imagens RGB-D: Um comparativo entre abordagens tradicionais e baseadas em aprendizado de máquina

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Eric Aislan Antonelo, Dr.
Universidade Federal de Santa Catarina

Prof. Tiago Loureiro Figaro da Costa Pinto, Dr.
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Engenharia de Automação e Sistemas.

Documento assinado digitalmente
Werner Kraus Junior
Data: 16/07/2020 09:31:09-0300
CPF: 531.085.239-53

Prof. Werner Kraus Junior, Dr.
Coordenador do Programa

Documento assinado digitalmente
Marcelo Ricardo Stemmer
Data: 15/07/2020 21:52:40-0300
CPF: 376.093.279-72

Prof. Marcelo Ricardo Stemmer, Dr.
Orientador

Florianópolis, 2020.

Este trabalho é dedicado aos meus pais, à minha namorada Jaciara, ao meu orientador, familiares, colegas de profissão e amigos. Sem o apoio destes, este trabalho não teria razão e nem seria possível.

AGRADECIMENTOS

Aproveito essa oportunidade para agradecer as pessoas que me apoiaram durante a jornada. Ao meu orientador, Prof. Dr. Marcelo Ricardo Stemmer, pelas orientações, conselhos e, principalmente, pela paciência e incentivo. Ao coordenador, Prof. Dr. Werner Kraus Junior por todo o apoio e pelo trabalho prestado ao departamento. Aos professores e ao departamento PGEAS por proporcionarem um período de grande aprendizado e convivência, assim como, agradeço à UFSC pela oportunidade, a experiência acadêmica e pessoal nesse período foram marcantes, moldaram minha personalidade e visão sobre todas as coisas. Agradeço também à CAPES pelo auxílio financeiro.

Agradeço também aos meus pais e namorada, por todo o apoio e paciência nos momentos mais difíceis, me dando forças para seguir mesmo quando parecia ser impossível. Ao meu empregador, VH Soluções em Software, pelo apoio, incentivo e paciência. Aos meus amigos Thiago, Fernando, familiares, amigos e colegas pela ajuda e compreensão durante todo o curso e desenvolvimento deste trabalho.

À todos vocês, um sincero e profundo muito obrigado.

“Se eu vi mais longe,
foi por estar sobre ombros de gigantes.”
(Isaac Newton)

RESUMO

O crescente desenvolvimento de robôs e sistemas autônomos têm impulsionado as mais diversas áreas de pesquisa. Um dos grandes desafios para sistemas autônomos é a tarefa de percepção do ambiente, permitindo ao robô “enxergar” o ambiente em que se encontra, através de sensores, como as câmeras digitais. A recente câmera RGB-D é capaz de captar imagens digitais (RGB) e informações de distância (D), tornando atraente o seu uso em áreas como a robótica móvel. O ambiente pode ser percebido e compreendido pelo sistema computacional através da captura dessas imagens e na aplicação de técnicas computacionais. A segmentação de objetos em imagem é uma grande área de estudo, com crescentes trabalhos, inovações e algoritmos, que podem ser divididos entre abordagens tradicionais e aprendizado de máquina. Este trabalho possui o objetivo de comparar estas abordagens de segmentação de objetos, avaliando a viabilidade de uso para aplicação em cenas de diferentes ângulos de ambientes internos. Para isso, foi realizada uma revisão sistemática de trabalhos de segmentação de objetos, formação de um *dataset* de cenas de ambientes internos, seleção e aplicação dos trabalhos no *dataset* formado, e, por fim, a comparação dos resultados através de métricas bem estabelecidas. Os resultados indicam vantagens no uso de imagens RGB-D para a robótica móvel. Indicam também resultados superiores das abordagens aprendizado de máquina em relação às abordagens tradicionais, mas que soluções combinadas podem oferecer melhores resultados. Os recentes avanços e desenvolvimentos das abordagens aprendizado de máquina, das câmeras RGB-D e dos sistemas computacionais tornam promissores o desenvolvimento de pesquisas e aplicações de segmentação de objetos em imagens RGB-D.

Palavras-chave: Segmentação de Objetos. RGB-D. Aprendizado de Máquina. Robótica Móvel.

ABSTRACT

The growing development of robots and autonomous systems has boosted the most diverse research fields. A great challenge for autonomous systems is the task of perceiving the environment, enabling the robot to “see” the environment in which it’s in, through sensors, such as digital cameras. The recent RGB-D cameras are capable of capturing digital images (RGB) along with distance information (D), making its use attractive in mobile robotics. The environment can be perceived and understood by computational system through the capture of these images and the application of computational techniques. Object segmentation in images is a large research field that has been seeing an increase in works, innovations and algorithms, can be divided into traditional and machine learning approaches. This work aims to compare both approaches of object segmentation, evaluating the feasibility of use for applications in indoor scenes from different angles. The comparison comprises a systematic review of existing object segmentation works, a formation of a dataset with indoor scenes, a selection and application of works in the formed dataset, and, finally, a comparison between results through well-established metrics. These results indicate advantages in the use of RGB-D images for mobile robotics. They also indicate superior results from machine learning approaches when compared to traditional approaches, but both solutions combined can offer even better results. Recent improvements in machine learning approaches, RGB-D cameras and computer systems make the development of research and applications for object semantic segmentation in RGB-D images a promising subject.

Keywords: Object Segmentation. RGB-D. Machine Learning. Mobile Robotic.

LISTA DE FIGURAS

Figura 1 – Etapas de um software de processamento de imagens. 19
Figura 2 – Apresentação da imagem monocromática conforme o número de bits. 26
Figura 3 – Variação da imagem conforme diferença de resolução. 26
Figura 4 – Exemplo de representação RGB em uma imagem digital. 27
Figura 5 – Exemplo do funcionamento do método Structured Light. 28
Figura 6 – Localização das câmeras e projetores sem a capa de proteção da câmera Kinect v2 28
Figura 7 – Exemplos de imagens D com contraste ajustado para facilitar a visualização. 29
Figura 8 – Exemplos de aplicação de técnicas para a segmentação de regiões em imagens monocromáticas. 31
Figura 9 – Exemplos de histogramas com um limiar T e T_1, T_2 34
Figura 10 – Comportamento do histograma com o aumento de ruído em uma imagem. 35
Figura 11 – Cálculo de uma linha das derivadas de primeira e segunda ordem a uma imagem monocromática. 36
Figura 12 – Tipos de bordas de salto, rampa e telhado. 37
Figura 13 – Zoom em uma região de uma imagem apresentando a direção do gradiente de um píxel. 38
Figura 14 – Janela de vizinhos $z_1 ... z_9$ e diferentes filtros para cálculo do gradiente. 38
Figura 15 – Exemplo da aplicação da operação *Sobel* em uma imagem original (figura à esquerda) e imagem suavizada (figura à direita): (a) imagem original, (b) *Sobel* direção x, (c) *Sobel* direção y, (d) soma das direções. 39
Figura 16 – Exemplo de limiarização nas figuras (d) da Figura 15 40
Figura 17 – Exemplo de detecção de bordas, onde respectivamente: imagem original, limiarização, algoritmo *Marr-Hildreth* e algoritmo *Canny*. . . 41
Figura 18 – Aplicação do Canny em uma tomografia de crânio, onde respectivamente: imagem original, limiarização, algoritmo *Marr-Hildreth* e algoritmo *Canny* 42
Figura 19 – Respectivamente: imagem de raio-x, histograma, sementes iniciais, sementes finais, diferença da imagem original, histograma, imagem segmentada em dois limiares, imagem segmentada com o mínimo em dois limiares, segmentação obtida pelo crescimento em regiões. 43
Figura 20 – Exemplo de divisão de uma imagem em quadrantes. 44
Figura 21 – Exemplo de aplicação da segmentação de região *k-means*, com $k = 4$ 45

Figura 22 – Exemplo de segmentação com super píxeis. A primeira linha com a imagem original e uma segmentação usando 4.000 super píxeis. A segunda e terceira linha usando 1.000, 500, 250. As bordas em branco na linha 1, figura do meio, e linha 2 são apenas para facilitar a visualização 46
Figura 23 – Comparação entre diferentes segmentações: Imagem original, imagem segmentada por *k-means*, super píxel com e sem borda (k = 100), *k-means* aplicado à imagem 4 47
Figura 24 – Exemplo de uma imagem, respectivo grafo e o corte (segmentação). 48
Figura 25 – Exemplo de resultado de segmentação do método *GrabCut*, especificando duas regiões, aplicado à imagem suavizada (centro). 49
Figura 26 – Exemplo de aplicação do algoritmo, com progressivas taxas de adições de água nas regiões baixas (escuras). 50
Figura 27 – Relação entre Machine Learning, Deep Learning, CNN, Computer Vision (Visão Computacional) e Visão Humana. 51
Figura 28 – Arquitetura básica de uma *perceptron*. 52
Figura 29 – Detalhe de um nó, com a soma dos pesos, valor de pŕé-ativação, função de ativação e valor pós-ativação. 53
Figura 30 – Arquitetura básica de uma RNA multicamada *feed-forward*. 54
Figura 31 – Principais funções de ativação utilizadas em *Deep Learning*. 56
Figura 32 – Exemplo básico de uma CNN, com camadas iniciais com mapas de características e camadas finais para classificação. 57
Figura 33 – Arquitetura básica de uma RNA convolucional. 58
Figura 34 – Aplicação de filtro no processo de convolução. 58
Figura 35 – Aplicação de half-padding a uma matriz 7x7 e filtro 5x5. 59
Figura 36 – Passo a passo da aplicação de filtro 3x3 com *stride* igual a 2. 60
Figura 37 – Exemplo da aplicação de *max-pooling* com *stride* 1 e 2. 61
Figura 38 – Visão geral da arquitetura RedNet desenvolvida pelos autores. . . . 62
Figura 39 – Apresentação das camadas da rede RedNet. 63
Figura 40 – Exemplos visuais da sobreposição de diferentes valores IoU. 66
Figura 41 – Exemplo de imagens RGB, D e D com histograma equalizado de cada *dataset*. Da primeira à quarta linha, respectivamente, PUTKK, Active Vision, S3DS sem tratamento, S3DS com tratamento. 77
Figura 42 – *Screenshot* da ferramenta labelme com uma imagem do *dataset* e as *labels* definidas manualmente. 78
Figura 43 – Exemplos das imagens selecionadas. Respectivamente por linha: Active Vision, PUTKK e S3DS. 80

Figura 44 – Imagem gerada pela ferramenta labelme para demonstrar as anotações realizadas em uma imagem RGB. Constam na legenda todas as classes anotadas na respectiva imagem. 80
Figura 45 – Amostra de uma imagem RGB, D, D com histograma equalizado e GT dos *datasets* Active Vision, PUTKK, S3DS S/T e S3DS C/T. . . . 81
Figura 46 – Imagem D colorida apresentando áreas cegas, em cor roxa, no lado esquerdo e direito. 82
Figura 47 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T. 84
Figura 48 – *Screenshot* com a interface visual desenvolvida pelos autores com as imagens de demonstração disponibilizadas. 85
Figura 49 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T. 86
Figura 50 – Interface desenvolvida pelos autores, que possibilita testes e ajustes dos parâmetros de segmentação em tempo de execução. 87
Figura 51 – Predições de objetos de interesse obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T. 88
Figura 52 – Predições de segmentação de regiões obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T. 88
Figura 53 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T. 89
Figura 54 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T. 90
Figura 55 – Gráfico da função de perda do treinamento no *dataset* SUN-RGBD. 91
Figura 56 – Gráfico de acurácia durante o treinamento no *dataset* SUN-RGBD. . 92
Figura 57 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T. 93
Figura 58 – Imagem à esquerda gerada pelos algoritmos ML. À direita, imagem convertida em regiões. 95
Figura 59 – Exemplificação do processo de recorte de duas regiões. As regiões a e b são recortadas da imagem Pred e comparadas com o respectivo local na imagem GT. 96
Figura 60 – Exemplo de três instâncias de quadros a,b e c, cada uma com suas respectivas sub-instâncias segmentadas. 98
Figura 61 – Seleção de quatro imagens dos resultados no *dataset* Active Vision. Respectivamente por linha: Imagem RGB, D, RGBD Proposals, JSCA-RM, Graph Canny Obj, Graph Canny Reg, FuseNet, RedNet e FCN. 100
Figura 62 – Gráfico com as métricas IoU por algoritmo e conjunto. 104

Figura 63 – Gráfico com as métricas IoU por algoritmo e conjunto obtidas das regiões propostas pelos algoritmos. 107
Figura 64 – Comparação das imagens, respectivamente: RGB, predição do algoritmo RGBD Proposals e predição do algoritmo RedNet. 108

LISTA DE TABELAS

Tabela 1 – Matriz de confusão para segmentação de objetos. 64
Tabela 2 – Comparação de tempos e custos de execução dos algoritmos tradicionais e algoritmos ML com suporte a GPU 101
Tabela 3 – Comparação de tempos e custos de execução dos algoritmos ML sem suporte a GPU 102
Tabela 4 – Resultados dos algoritmos ML no *dataset* formado. 103
Tabela 5 – Resultados divulgados pelos autores no *dataset* SUN-RGBD 104
Tabela 6 – Resultados dos algoritmos baseados das predições de regiões, por conjunto. 106
Tabela 7 – Resultado médio dos algoritmos baseados das predições de regiões. 107
Tabela 8 – Resultado médio dos algoritmos baseados das predições de regiões com melhor IoU de cada instância das classes. 108
Tabela 9 – Resultados dos algoritmos baseados das predições de regiões com melhor IoU de cada instância das classes. 121

LISTA DE ABREVIATURAS E SIGLAS

2D	Duas dimensões
3D	Três dimensões
BB	Bounding Box
CPU	Central Processing Unit
CNN	Convolutional Neural Network
D	Depth
DL	Deep Learning
DNN	Deep Neural Network
F1	F-score ou F-measure
GPU	Graphics Processing Unit
GT	Ground Truth
HD	High Definition
IOU	Intersection Over Union
IR	Infrared
ML	Machine Learning
PCL	Point Cloud Library
PDI	Processamento Digital de Imagens
RGB	Red, Green and Blue
RGB-D	Red, Green, Blue and Depth
RNA	Rede Neural Artificial
RNR	Rede Neural Recorrente
SDK	Software Development Kit
SL	Structured Light
SVM	Support Vector Machine
TF	Time of Flight
VC	Visão Computacional

LISTA DE SÍMBOLOS

$\in$	Pertence
$\subseteq$	É sub conjunto
$\rightarrow$	Implica
$\forall$	Para todo
$\mathbb{N}$	Números naturais
$\cup$	União
$\cap$	Intersecção
$\varnothing$	Conjunto Vazio
∞	Infinito
α	Alpha
$\sum$	Sigma
Δ	Delta
δ	Delta
θ	Theta
ϑ	Theta
λ	Lambda
ω	Omega
ϕ	Phi
η	Eta
∂	Derivada parcial

SUMÁRIO

1 INTRODUÇÃO 18
1.1 OBJETIVOS 21
1.1.1 Objetivo Geral 21
1.1.2 Objetivos Específicos 22
1.1.3 Metodologia 22
1.1.4 Limitações 23
1.1.5 Contribuições 23
1.1.6 Estrutura do Documento 24
2 FUNDAMENTAÇÃO TEÓRICA 25
2.1 IMAGEM DIGITAL 25
2.1.1 Imagem D (depth - distância) 27
2.1.2 Histograma 30
2.2 SEGMENTAÇÃO DE OBJETOS 30
2.2.1 Conjunto verdade (GT - Ground Truth) das imagens 31
2.2.1.1 Tipos de GT 32
2.2.1.2 Datasets 32
2.2.2 Abordagens tradicionais 33
2.2.2.1 *Threshold* (Limiarização) 34
2.2.2.2 Detecção de bordas 35
2.2.2.3 *Region Growing Segmentation* 42
2.2.2.4 *Region Splitting and Merging* 44
2.2.2.5 *Region Segmentation using Clustering and Superpixels* 44
2.2.2.6 *Region GrabCut Segmentation* 47
2.2.2.7 *Watershed Based Segmentation* 49
2.2.2.8 Aplicações em imagens coloridas 49
2.2.3 Abordagens Machine Learning 50
2.2.3.1 Redes Neurais Artificiais 52
2.2.3.1.1 Redes Neurais Convolucionais (CNN) 56
2.2.3.2 Estudo de Caso - RedNet 61
2.2.4 Métricas de segmentação 64
2.2.4.1 Acurácia 65
2.2.4.2 Precisão 65
2.2.4.3 Recall 65
2.2.4.4 F1 66
2.2.4.5 IoU 66
3 TRABALHOS RELACIONADOS 67
3.1 TRABALHOS COM PROPOSTAS SEMELHANTES 67

3.2 SELEÇÃO DOS TRABALHOS PARA OS EXPERIMENTOS 68
3.3 ABORDAGENS TRADICIONAIS 70
3.4 ABORDAGENS APRENDIZADO DE MÁQUINA 72
4 DESENVOLVIMENTO E IMPLEMENTAÇÃO 75
4.1 FORMAÇÃO DO *DATASET* DE CENAS DE AMBIENTES INTERNOS 75
4.1.1 Seleção 75
4.1.2 Criação do conjunto verdade (GT) 78
4.1.3 Recortes de áreas cegas 81
4.2 EXECUÇÃO DOS ALGORITMOS SELECIONADOS 82
4.2.1 Abordagens tradicionais 83
4.2.2 Abordagens Machine Learning 88
4.3 DESENVOLVIMENTO DAS MÉTRICAS NOS RESULTADOS OBTIDOS 93
4.3.1 Métricas para segmentação semântica por classe 94
4.3.2 Métricas para segmentação por região 94
5 RESULTADOS 99
5.1 RESULTADOS DO CONSUMO DE RECURSOS 99
5.2 RESULTADOS DAS ABORDAGENS ML 103
5.3 RESULTADOS DAS ABORDAGENS TRADICIONAIS E ML 105
5.4 CONSIDERAÇÕES FINAIS 109
6 CONCLUSÃO 110
6.1 TRABALHOS FUTUROS 112
REFERÊNCIAS 114
APÊNDICE A – PARÂMETROS UTILIZADOS PARA O GRAPH CANNY SEGMENTATION 119
APÊNDICE B – INFORMAÇÕES COMPLEMENTARES DOS RESULTADOS 121

1 INTRODUÇÃO

O crescente desenvolvimento da robótica móvel, aliado ao aumento do poder computacional embarcado, tem proporcionado muitos desafios para os pesquisadores. Cada vez mais tarefas manuais, e até tarefas antes inexistentes, estão sendo automatizadas através da robótica. Tarefas como locomoção e exploração em ambientes hostis, transporte de materiais, automação de produções industriais, controle aéreo, vigilância, e mais recentemente, os tão previstos veículos autônomos, vistos como algo futurista no passado, e que hoje o seu desenvolvimento é um assunto popular, sendo motivo de notícia e discussões éticas em diversas mídias.

Para a execução de tarefas como essas, desafios de percepção, localização, cinemática e navegação são enfrentados pela robótica, questões essas multidisciplinares e que envolvem os mais diversos conhecimentos, soluções e aplicações, demandando novas pesquisas, aprimoramentos e validações (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Tipicamente um robô é constituído por componentes de *hardware* (sensores e atuadores), e *software* (programas para a realização das ações do robô) (BARBOSA, 2017). Para a robótica móvel, um sensor comum para percepção do ambiente externo são as câmeras de captação de imagens digitais que, aliadas à algoritmos computacionais, permitem ao robô “enxergar” o ambiente. A área de VC (Visão Computacional - *Computer Vision*) da computação é responsável pelo estudo e desenvolvimento de algoritmos sobre imagens digitais (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Deseja-se em alguns casos que o robô tenha certo grau de autonomia, como em sua movimentação por um ambiente, por exemplo. Segundo (RUSSELL; NORVIG, 2009), um agente, como um robô, pode ser considerado autônomo quando além de agir, o agente é capaz de perceber o ambiente em que está inserido, adaptar-se a mudanças, criar e perseguir objetivos, por um período longo de tempo.

Para auxiliar o robô na percepção do ambiente, um tipo de sensor que vem sendo utilizado na robótica móvel é a câmera *RGB-D*. Esse sensor, acoplado ao robô, além de captar imagens com informações de cor (*RGB - Red, Green, Blue*), capta também informações de distância (D -*Depth* - profundidade), através de raios infravermelhos emitidos pela câmera. A vantagem dessa câmera, em relação à câmeras comuns que captam apenas imagens com informações *RGB*, é que a informação de profundidade independe de fonte de luz no ambiente, e permite o cálculo de distância dos objetos, servindo como uma informação adicional para as técnicas de visão computacional. Dentre as câmeras *RGB-D* disponíveis, a mais popular é o Kinect V2, com imagem *RGB FULL-HD*, a um baixo custo (MORAIS ALONSO *et al.*, 2015).

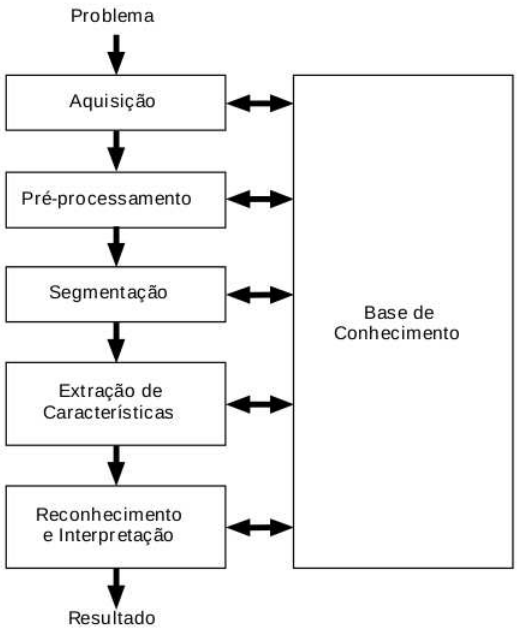
A VC é uma área da computação que trata da interpretação de informações da imagem, ou seja, trata de interpretar o conhecimento obtido de uma imagem. É atra-

vés de técnicas de PDI (Processamento Digital de Imagens) que as informações são extraídas da imagem. Estas técnicas realizam transformações na imagem, de forma a possibilitar a extração de informações semânticas através de algoritmos computacionais (PEDRINI; SCHWARTZ, 2007).

Nesse contexto de VC e PDI, um software de processamento de imagens comumente possui as seguintes etapas: Aquisição, Pré-processamento, Segmentação, Representação e Descrição, Reconhecimento e Interpretação (MARQUES FILHO; VIEIRA NETO, 1999).

Essas etapas, ilustradas na Figura 1, possuem as seguintes finalidades:

Figura 1 – Etapas de um software de processamento de imagens.



Fonte: (MARQUES FILHO; VIEIRA NETO, 1999)

1. Etapa de Aquisição: Obtenção/captação da imagem;
2. Pré-processamento: Aplicação de algoritmos de PDI para diminuição de ruídos, correção de contrastes, dentre outras manipulações na imagem com o objetivo de melhorar a qualidade da imagem e prepará-la para as etapas seguintes;
3. Segmentação: Etapa em que são aplicados algoritmos com o objetivo de definir/extrair as formas dos objetos contidos na imagem e encontrar regiões de interesse na imagem;
4. Representação e Descrição: cujo objetivo é colocar os pontos, linhas e regiões encontradas na etapa anterior em uma estrutura de dados, o objetivo é organizar e facilitar a próxima etapa de reconhecimento; e

5. Reconhecimento e Interpretação: É a etapa que atribui um significado a um conjunto de objetos reconhecidos na imagem, também chamada de classificação.

Em cada etapa desse processamento, informações são geradas, manipuladas e obtidas da imagem e utilizadas pelas etapas posteriores. O conhecimento é acumulado e utilizado para a geração de um resultado desejado. Essas etapas permitem ao robô ou ao sistema computacional (programa de computador) extrair informações semânticas das imagens e possibilitando o conhecimento de objetos, obstáculos e outras características, dependendo do contexto da aplicação.

Este trabalho está focado em trabalhos e algoritmos para a etapa de segmentação. Existem muitos algoritmos e métodos para segmentar os píxeis de uma imagem, formando conjuntos, ou regiões, indicando a existência de objetos a serem classificados. Para essas técnicas, existem a abordagem tradicional e a abordagem utilizando ML. Na abordagem tradicional, técnicas de PDI são aplicadas à imagem, como erosão e dilatação, detecção de bordas, crescimento de região, *watershed*, algoritmos de clusterização de píxeis, dentre outros. Ou seja, algoritmos são estudados e aplicados caso a caso, em uma espécie de mineração para a seleção desses conjuntos de píxeis (SOLOMON; BRECKON, 2011).

Já na abordagem ML, técnicas como RNA (redes neurais artificiais) são utilizadas. A RNA é treinada com *datasets* de exemplos de imagens e seus *ground truths* (conjuntos verdade), que indicam o formato e classe dos objetos na imagem. Deste modo, a rede neural ajusta os seus pesos, aprendendo de forma indireta os padrões dos objetos existentes na imagem, sendo capaz de identificar esses padrões em novas imagens, ainda não conhecidas pela rede neural.

Em ML, segundo (MITCHELL, 1997), muitos algoritmos e abordagens são desenvolvidos com o objetivo de fazer que a máquina tenha a capacidade de aprender e executar tarefas, como nós, humanos. Embora ainda limitado, comparado à nossa capacidade, muitas aplicações práticas foram desenvolvidas nesse campo. As RNAs são técnicas utilizadas em ML que permitem à máquina aprender e reconhecer padrões.

As Redes Neurais Artificiais (RNA) são algoritmos e estruturas computacionais inspiradas pelo processo de decisão das conexões dos neurônios do cérebro biológico humano, embora a rede neural biológica seja ainda muito mais complexa e eficiente do que a artificial, até o momento (GRAUPE, 2007). A RNA passou a ser adotada pela sua capacidade de resolver diferentes problemas, de se adaptar às condições da imagem (diminuindo a necessidade de eliminação de ruídos e outros ajustes na imagem).

Existem na literatura uma ampla gama de pesquisas e aplicações elaboradas sobre o tema de segmentação de objetos em imagens. Entretanto, grande parte das pesquisas estado-da-arte são aplicadas a *datasets* públicos, que em geral contém imagens de diferentes cenas, com diferentes objetos pertencentes a um conjunto definido de categorias de objetos, visando medir o desempenho do algoritmo desenvolvido com

outros algoritmos aplicados ao mesmo *dataset*.

Em situações gerais, onde deseja-se encontrar a maior taxa de acerto possível a um problema de segmentação de objetos, é preferível que algoritmos de ML (*Machine Learning* - Aprendizado de Máquina) sejam treinados no mesmo ambiente que serão aplicados. Por exemplo, treinados e aplicados no ambiente em que um robô autônomo deve movimentar-se e localizar objetos. Quando aplicados a conjuntos de imagens diferentes dos conjuntos de treino e validação, é esperado que a taxa de acerto dos algoritmos ML sejam menores.

Diante disso, algumas perguntas surgem para a aplicação dessas técnicas computacionais estado-da-arte treinadas em *datasets* com cenas de diferentes ambientes, mas que deseja-se aplicar a um único ambiente real, onde as cenas e objetos se apresentam em diferentes ângulos. Quais resultados esses algoritmos teriam nesse caso? As métricas de desempenho dos resultados permanecem semelhantes? A diferença de captação das imagens através das câmeras RGB-D influenciam os resultados? Quais abordagens apresentaram maior diferença entre os resultados? Através dos experimentos e métricas aplicadas, pode-se concluir se há uma abordagem melhor, aplicada a determinadas situações? Qual dificuldade teria um pesquisador fazer o uso desses algoritmos e aplicá-los a um determinado problema?

Para contribuir com informações e resultados que auxiliem a responder essas perguntas, esta pesquisa apresenta uma comparação entre experimentações de abordagens tradicionais e ML, com a formação de um *dataset* de cenas de ambientes internos e executados em uma máquina com processamento similar à capacidade de processamento de robôs autônomos.

1.1 OBJETIVOS

Nas seções abaixo estão descritos o objetivo geral e os objetivos específicos para esta pesquisa.

1.1.1 Objetivo Geral

O objetivo geral desta pesquisa é apresentar uma comparação entre o uso de abordagens clássicas e abordagens ML na segmentação de objetos em imagens RGB-D, aplicados a cenas em ângulos diferentes de ambientes internos. Como objetivos complementares pretende-se investigar e estudar pesquisas estado-da-arte dessas duas abordagens de segmentação de objetos, descrever todo o processo de seleção, implementação e comparação dos resultados obtidos dessas técnicas computacionais, além de contribuir com informações e estudos para a área.

1.1.2 Objetivos Específicos

Com base no objetivo geral, foram definidos os seguintes objetivos específicos:

- Pesquisar trabalhos estado-da-arte com o tema de segmentação de objetos em imagens RGB-D, usando o método de revisão sistemática;
- Selecionar trabalhos aplicáveis ao contexto;
- Reproduzir os trabalhos em um *dataset* de cenas em diferentes ângulos de ambientes internos;
- Validar os algoritmos selecionados no *dataset* encontrado;
- Apresentar e explanar a bibliografia encontrada, de modo a contribuir com a área de conhecimento;
- Apresentar a comparação entre as abordagens aplicadas ao problema proposto; e
- Responder às questões propostas e concluir sobre os resultados obtidos.

1.1.3 Metodologia

Esta pesquisa é de natureza pesquisa aplicada, com objetivo prático de gerar conhecimento através da comparação proposta entre as abordagens, aplicadas ao problema. A revisão sistemática foi escolhida como metodologia para esta pesquisa, definindo como foi realizada a pesquisa dos artigos científicos e seleção daqueles que são de interesse e aplicáveis ao problema proposto.

A revisão sistemática tem como objetivo pesquisar e realizar uma síntese dos trabalhos existentes na literatura para um determinado tema. Um plano de pesquisa com etapas deve ser criado e seguido de forma metódica, de modo a alcançar de maneira justa o maior número de trabalhos possíveis em diferentes bases de trabalhos publicados (KITCHENHAM, 2004). Com os trabalhos encontrados, as etapas finais da revisão sistemática aplicada definem filtros, através de métricas, para diminuir o conjunto de trabalhos de interesse e estabelecer um *ranking* de artigos. O detalhamento dos passos utilizados na revisão sistemática aplicada estão descritos no Seção 3.2.

Os artigos ML selecionados e aplicados na comparação foram padronizados para serem treinados com o mesmo *dataset*, e comparados com outro conjunto de imagens, formados pelo autor, a partir da seleção de imagens de outros *datasets* públicos pesquisados. Para a formação do *dataset* dos experimentos, foram pesquisados os mais utilizados, e também verificado entre os artigos selecionados. O *dataset* SUN-RGBD (SONG; LICHTENBERG; XIAO, 2015) foi utilizado no treinamento dos algoritmos ML.

Para os experimentos dos algoritmos em cenas em diferentes ângulos de ambientes internos, foram selecionadas imagens de três *datasets* diferentes, contendo imagens em diferentes ângulos de ambientes internos. O Capítulo 4 descreve em detalhe o processo de seleção.

Os resultados foram comparados de forma quantitativa, com as mesmas métricas e aplicados ao *dataset* formado, apenas diferindo a implementação de cada algoritmo, disponibilizada pelos autores. Devido a diversidade das pesquisas selecionadas, autores, *frameworks*, linguagens e códigos-fonte, as implementações dos algoritmos são diferentes, o que adiciona uma variável de erro não conhecida entre as comparações dos resultados obtidos. Entretanto, houve o esforço de manter e validar com igualdade as implementações.

1.1.4 Limitações

O *dataset* utilizado nas experimentações foi formado a partir de imagens RGB-D, selecionadas de *datasets* públicos. Inicialmente, houve a tentativa de captação própria das imagens, porém, inviabilizada por problemas de compatibilidade entre a câmera Kinect v2 e os recursos computacionais disponíveis. Deste modo, um *dataset* foi formado, a partir de imagens de *datasets* públicos encontrados, com imagens selecionadas seguindo o critério de cenas em diferentes ângulos de ambientes internos.

O *dataset* formado foi utilizado nos experimentos apenas para a avaliação das abordagens tradicionais e ML, e não para o treinamento, no caso das abordagens ML. Ou seja, o conjunto de validação formado é diferente do conjunto de treino e validação do *dataset* SUN-RGBD, utilizado pelos algoritmos ML selecionados para os experimentos deste trabalho.

Em relação aos trabalhos pesquisados, foram selecionados trabalhos com o código-fonte disponível. Nos trabalhos de interesse, porém sem o código-fonte disponível, foi realizado o contato com os autores.

Trabalhos baseados exclusivamente em PCL - Point Cloud Library (RUSU; COUSINS, 2011) foram descartados dos experimentos, pois exigiriam a criação, ou a geração, do *dataset* criado em formato PCL. Trabalhos com alta exigência de GPU (no quesito memória) também foram descartados, limitados ao *hardware* disponível (GPU com 4GB de memória).

1.1.5 Contribuições

As seguintes contribuições foram obtidas após o desenvolvimento deste trabalho:

- Formação de um *dataset* de 75 imagens, de cenas em diferentes ângulos, de ambientes internos, com GT criados pelo autor;

- Disponibilização de 6 códigos-fonte adaptados para os experimentos dos trabalhos selecionados de segmentação de objetos;
- Comparação dos resultados das abordagens tradicionais e aprendizado de máquina na segmentação de objetos; e
- Resposta e avaliação dos problemas propostos.

1.1.6 Estrutura do Documento

Esta dissertação, além do Capítulo 1 de Introdução, é composta por mais cinco capítulos, os quais abordam diferentes etapas e descrições da pesquisa e experimentação desenvolvida.

O Capítulo 2 de fundamentação teórica tem como objetivo explicar e revisar a bibliografia dos conceitos e abordagens utilizadas nos demais capítulos e que envolvem o tema desta dissertação.

No Capítulo 3, intitulado trabalhos relacionados, são descritos o que foi encontrado de trabalhos relacionados a esta pesquisa, como também os trabalhos selecionados, que foram estudados e utilizados nos experimentos.

No Capítulo 4 são apresentadas as implementações dos experimentos, condições de execuções, métricas aplicadas, método de execução e coleta de dados.

No Capítulo 5 os resultados são sumarizados e analisados, comparando os resultados de cada algoritmo nos respectivos cenários.

Por fim, no Capítulo 6 são descritas as conclusões e relatos obtidos após o desenvolvimento desta pesquisa. Também são descritos as sugestões para trabalhos futuros, visando a melhoria e expansão da pesquisa e dos experimentos.

2 FUNDAMENTAÇÃO TEÓRICA

Visando a compreensão dos assuntos envolvidos na segmentação de objetos em imagens RGB-D, uma pesquisa foi realizada para fundamentar os conceitos e soluções utilizadas, de modo a complementar o trabalho e também contribuir com o compartilhamento do conhecimento.

Conforme exposto na Introdução, as câmeras são sensores que permitem aos robôs e sistemas computacionais “enxergarem” o ambiente. Para permitir a compreensão da cena, a primeira etapa é o processo de conversão da luz capturada pelas câmeras em imagens digitais. A segunda etapa é o processamento digital da imagem, de modo a obter informações desejadas, como a segmentação e identificação dos objetos em cena (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011).

Somente para realizar a segmentação de objetos na imagem, são envolvidas uma série de conceitos e propostas que fornecem os meios para possibilitar a tarefa de segmentação. Nas seções seguintes, esses conceitos são apresentados através dos assuntos: imagens digitais e suas propriedades; segmentação de objetos, tipos e métricas de avaliação; abordagens clássicas e algoritmos; abordagens ML e os principais conceitos de redes neurais.

2.1 IMAGEM DIGITAL

Quando uma imagem é captada a partir de um sensor, como uma câmera fotográfica, ela passa por um processo de digitalização. A representação de uma imagem digital é estabelecida pela função Equação (1), onde x e y representam coordenadas bidimensionais de uma matriz (linha e coluna), e o resultado de f é chamado de píxel, a menor unidade de uma imagem digital, representando uma cor.

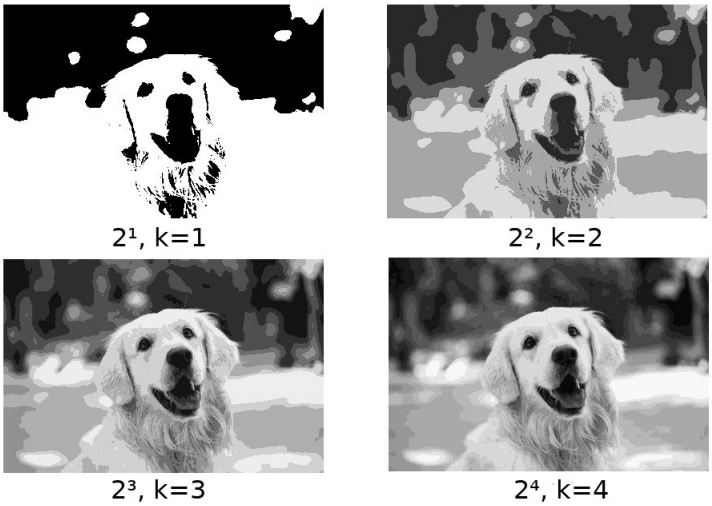
$$f(x,y) \mid \{x,y\} \in \mathbb{N}. \tag{1}$$

Em uma imagem digital monocromática, onde são representadas as intensidades de luz contidas na imagem (escala de cinza), os valores assumidos pela função Equação (1) dependem do número de bits que serão usados para o armazenamento da imagem digital, podendo assumir os valores conforme a Equação (2), onde k representa o número de bits. A Figura 2 mostra a diferença visual conforme aumenta-se o valor de k, permitindo uma maior representação de valores e tons de cinza.

$$2^k \mid \{k\} \in \mathbb{N}\,,\ \{k\} > 0. \tag{2}$$

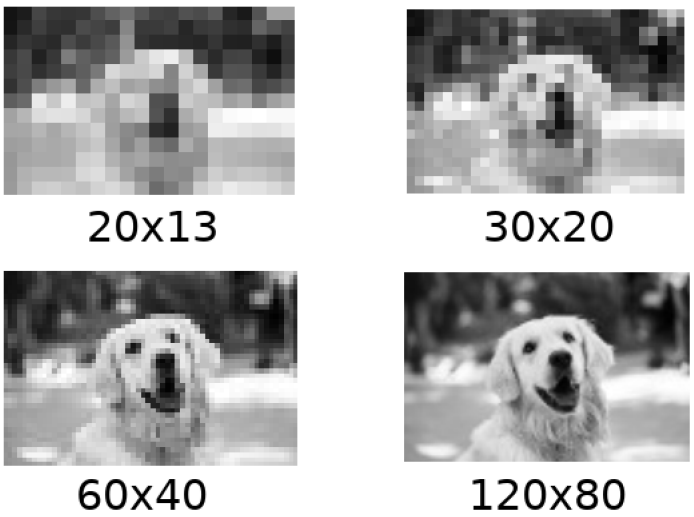
Uma propriedade das imagens digitais é a sua resolução, que refere-se ao número de linhas e colunas que a imagem será representada. A Figura 3 apresenta uma mesma imagem em diferentes dimensões. Quando uma imagem digital é impressa,

Figura 2 – Apresentação da imagem monocromática conforme o número de bits.



seja em um monitor, ou em papel, ainda há a informação de PPI (Píxel Per Inch - píxel por polegada) e DPI (Dots Per Inch - pontos por polegada), ou seja, quantos píxeis são enquadrados em uma polegada, no caso do monitor, ou quantos pontos serão enquadrados por polegada, no caso de impressão física (como o papel). Essa informação é necessária para trazer essa informação de número de píxeis e dimensões para o mundo físico. A Figura 3 ilustra as figuras em diferentes PPI, por isso apresentam o mesmo tamanho das quatro figuras, embora as figuras tenham resoluções diferentes, apresentando os píxeis em "tamanhos diferentes".

Figura 3 – Variação da imagem conforme diferença de resolução.



Para imagens digitais coloridas, a representação da cor é dada através de um sistema de modelo de cor. O mais comum é o uso do sistema RGB. A cor é

representada através dos três componentes de cores principais: vermelho, verde e azul. Nesse formato, um píxel é formado pelos três componentes. O armazenamento a imagem digital também é dada pela Figura 2, para cada componente de cor. O padrão atual de 8 bits para cada componente permite a representação de 16.777.216 milhões de cores (PEDRINI; SCHWARTZ, 2007). Um exemplo de imagem RGB com a amostra de 9 píxeis e seus respectivos valores é apresentado na Figura 4.

Figura 4 – Exemplo de representação RGB em uma imagem digital.

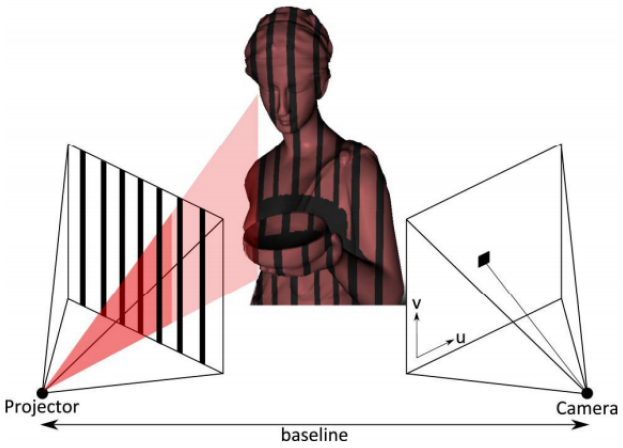
Uma característica de câmeras de vídeo é a informação de quantos *frames* (quadros) por segundo (fps) ela é capaz de captar. Isso indica quantas imagens por segundo a câmera gera. Um vídeo nada mais é que uma sequência de imagens (GONZALEZ; WOODS, 2011). Para a câmera Kinect v2, sua capacidade é a de captar imagens RGB-D a 30 fps. Em caso de aplicações em tempo real, elas devem aplicar uma estratégia para lidar com essa grande quantidade de informação por segundo, uma vez que o *hardware* pode não ser capaz de aplicar técnicas como a segmentação de objetos, a todos esses 30 fps fornecidos pela câmera.

2.1.1 Imagem D (depth - distância)

As câmeras RGB-D usam um emissor de IR (raios infravermelhos) e um receptor para calcular a distância dos objetos dentro do seu campo de visão. Dentre as câmeras RGB-D citadas, para o cálculo de distância dos objetos em relação à câmera, são utilizados dos métodos de cálculo: Structured Light (SL - Luz Estruturada) e Time of Flight (TF - Tempo de vôo).

O método SL, utilizado pelo Kinect v1, consiste em emitir raios infravermelhos em um padrão conhecido nos objetos, que então sofrem uma distorção pelo formato dos objetos. Os objetos são observados por câmera, em uma diferente posição, que através da distorção do padrão emitido com o padrão observado, a informação de distância pode ser calculada (SARBOLANDI; LEFLOCH; KOLB, 2015). A Figura 5 apresenta um exemplo de padrão emitido pelo projetor de IR e observado por uma câmera adjacente.

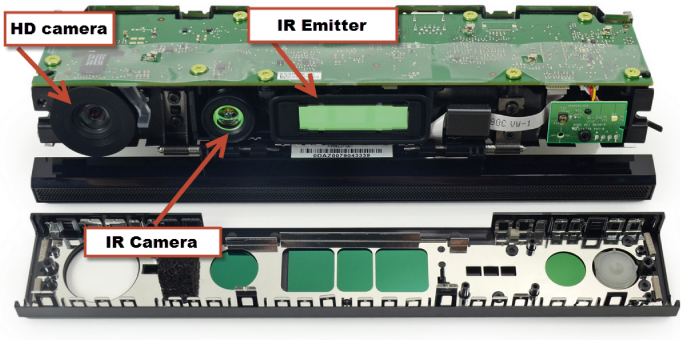
Figura 5 – Exemplo do funcionamento do método Structured Light.



Fonte: Sarbolandi, Lefloch e Kolb (2015).

O método TF é baseado na medição da diferença do tempo em que a luz IR emitida por um projetor é refletida de volta a um receptor. Conhecida a velocidade da luz, pode-se deduzir a distância em que os objetos se encontram. A Figura 6 mostra como estão localizadas as câmeras RGB e o receptor IR, como também o projetor IR.

Figura 6 – Localização das câmeras e projetores sem a capa de proteção da câmera Kinect v2



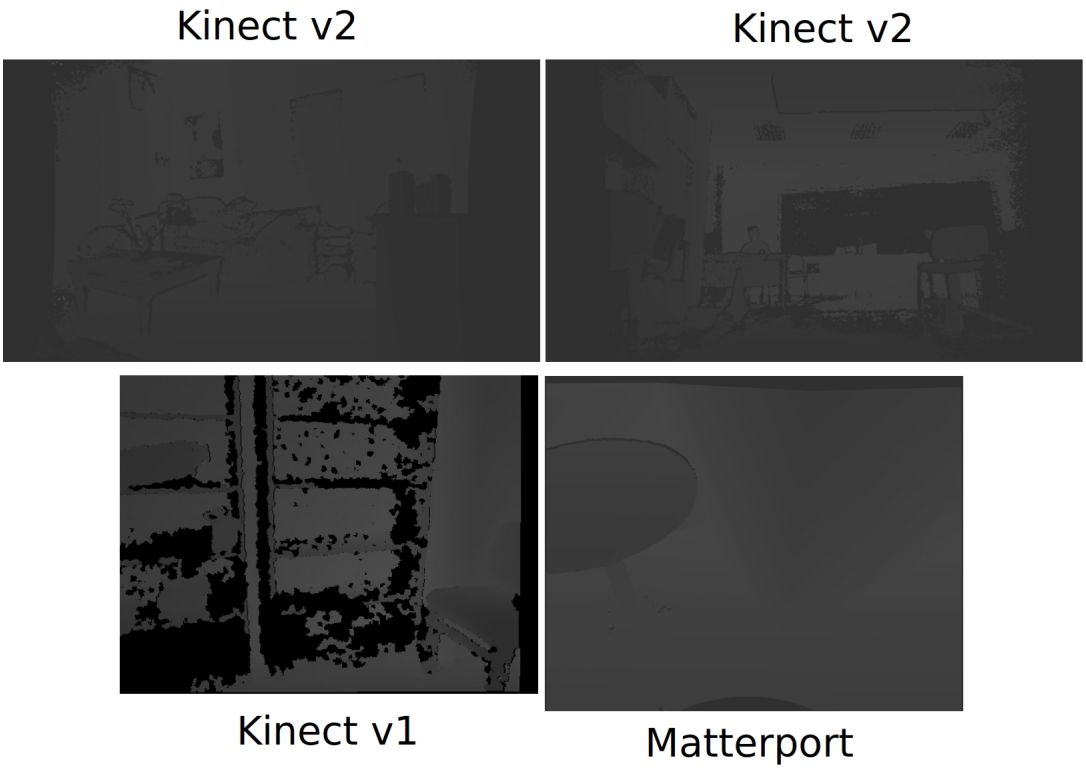
Fonte: Internet.

Diante dos cálculos sensíveis à luz, as câmeras RGB-D são suscetíveis a alguns problemas de interferências e erros de cálculo, como por exemplo: luz do ambiente, interferência por outros dispositivos, aquecimento do dispositivo (devido a constante emissão de IR), erros do cálculo de distância (devido a falta de calibração da câmera), oclusões de objetos, sobreposição de luzes refletidas, objetos mais reflexivos e menos reflexivos, objetos transparentes (atraso na reflexão dos sinais IR), objetos distantes e objetos em movimento. Esses erros são intrínsecos à essa tecnologia, e muitos trabalhos foram elaborados de forma a diminuir o seu impacto, conforme a necessidade

de cada aplicação. Os autores (SARBOLANDI; LEFLOCH; KOLB, 2015) exploram, em detalhe, a diferença dos métodos para cálculo da imagem D entre as câmeras Kinect v1 e Kinect v2.

Com isso, as câmeras RGB-D proveem a informação de distância em forma de uma imagem em escala de cinza. Devido ao *driver* de captura utilizado, ou da fonte em que a imagem D foi obtida, o padrão e qualidade da imagem pode sofrer alteração, tanto para a sincronização com a imagem RGB, quanto para a suavização de ruídos. A Figura 7 apresenta exemplos de imagens D sem tratamento de diferentes câmeras, apenas o contraste foi ajustado para melhor visualização. O método de captação e cálculo também influenciam na qualidade da imagem, ocasionando também diferenças de ruídos e pontos cegos.

Figura 7 – Exemplos de imagens D com contraste ajustado para facilitar a visualização.



Podem ser encontrados na literatura muitos trabalhos que visam corrigir as imperfeições da imagem D, desde eliminação/suavização de ruídos a até reconstrução da imagem D, de forma a reproduzir com maior fidelidade possível a cena em questão.

Em (JING *et al.*, 2017), os autores fazem uma descrição completa das características das câmeras Kinect v1, Kinect v2 e PrimeSense, suas compatibilidades de *hardware* e *software*, além de uma análise comparativa da qualidade das imagens D de cada câmera em diferentes distâncias, como também uma avaliação do cálculo de distância entre os objetos em diferentes distâncias para cada câmera.

2.1.2 Histograma

Histograma é um termo frequentemente encontrado em trabalhos de soluções em PDI. O histograma consiste na quantificação de píxeis para cada canal de cor. Em imagens em escala de cinza, o histograma informa quantos píxeis, ou o percentual destes, existem para cada valor encontrado na imagem. Em imagens coloridas, normalmente os canais R, G e B são calculados separadamente. Essa técnica permite, por exemplo, identificar o contraste, nível médio de brilho e predominância de píxeis.

Diversas técnicas de PDI foram elaboradas com o objetivo de modificar o histograma de uma imagem, ou seja, aplicar transformações nos píxeis, modificando o seu valor, e consequentemente o histograma da imagem. As técnicas podem ser para modificar a intensidade, equalização ou expansão do histograma (MARQUES FILHO; VIEIRA NETO, 1999).

2.2 SEGMENTAÇÃO DE OBJETOS

A segmentação em imagem é um processo que trata de dividir a imagem em um número definido de regiões individuais que identificam suas unidades significativas. No caso de imagens de ambientes internos, trata-se de dividir a imagem em objetos (MARQUES FILHO; VIEIRA NETO, 1999).

Em (GONZALEZ; WOODS, 2011), os autores estabelecem que a segmentação é o processo de dividir uma imagem R em n sub-regiões, processo este definido pelas equações:

$$\bigcup_{i=1}^{n} R_i = \mathrm{R}. \tag{3}$$

$$R_i \text{ é um conjunto conectado, para } i = 0,1,2,...,n. \tag{4}$$

$$R_i \cap R_j = \varnothing \text{ para todos os i e j, } i \neq j. \tag{5}$$

$$Q(R_i) = \textit{VERDADEIRO} \text{ para todos os } i = 0,1,2,...,n. \tag{6}$$

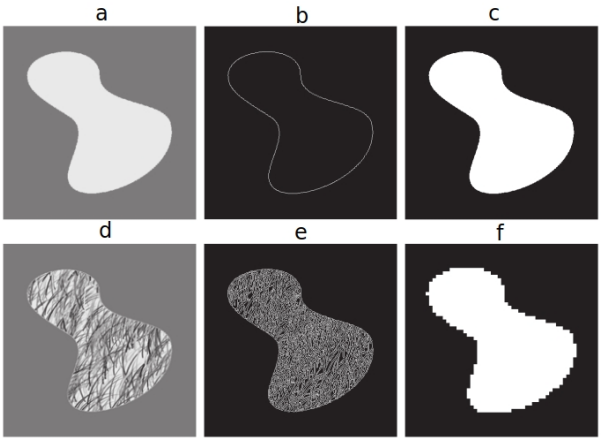
$$Q(R_i \cup R_j) = \textit{FALSO} \text{ para qualquer região adjacente } R_i \text{ e } R_j. \tag{7}$$

A Equação (3) estabelece que todo píxel deve pertencer em uma região. A Equação (4) estabelece que os pontos dentro de uma sub-região devem estar conectados. A Equação (5) descreve que as sub-regiões devem ser desconectadas uma das outras (um píxel pode pertencer a apenas uma região). A Equação (6) diz que os píxeis presentes na sub-região devem satisfazerem uma condição para pertencerem à

respectiva sub-região. Por fim, a Equação (7) indica que os píxeis de duas sub-regiões adjacentes não devem satisfazerem uma mesma condição Q.

O conceito de segmentação pode ser compreendido através da Figura 8. Na primeira linha a figura (a) é segmentada por uma técnica de segmentação de contornos, através da diferença de intensidade, resultando na figura (b), a segmentação então é obtida atribuindo a classe 0 aos píxeis de fora do contorno, e 1 aos píxeis de dentro, resultando na figura (c). Desta forma, entende-se como segmentado uma sub-região de interesse, identificada com o valor 1, e descartado os demais píxeis para a região de fora, resultando em uma segmentação binária (com duas classes).

Figura 8 – Exemplos de aplicação de técnicas para a segmentação de regiões em imagens monocromáticas.



Fonte: Adaptado de (GONZALEZ; WOODS, 2011)

Já na segunda linha, se a mesma técnica é aplicada a figura (d), que possui uma textura, o resultado não é o esperado, conforme a figura (e) demonstra, não sendo um método aplicável para esse caso. Entretanto, pode-se observar que a região não desejada na imagem (d) é constante, e a região de interesse é uma textura. Pode-se então propor uma solução onde, divide-se a imagem em conjuntos de 8x8 píxeis, e para cada conjunto, aplica-se a classe 1 se o cálculo de desvio padrão dos píxeis é positivo, e 0 para o contrário, resultando na figura (f) com a segmentação binária da sub-região de interesse, que apresenta um aspecto serrilhado devido à janela 8x8 aplicada à imagem para o cálculo (GONZALEZ; WOODS, 2011).

2.2.1 Conjunto verdade (GT - Ground Truth) das imagens

Os GT são anotações, geralmente criadas manualmente pelos autores, que descrevem as informações verdadeiras contidas nas imagens. Essas descrições podem ser armazenadas em texto (através de uma estrutura bem definida, como XML ou Json), em imagem, ou em uma outra estrutura desejada.

2.2.1.1 Tipos de GT

Existem diferentes tipos de GT que variam conforme o que deseja-se descrever. Dos tipos existentes mais utilizados na segmentação de objetos, pode-se citar:

- Classificação da imagem: Rótulos descrevendo os objetos contidos nas imagens;
- Cena: Descrição da cena representada pela imagem, podem ser descrições simples do ambiente da cena, como por exemplo sala, cozinha, quarto, ou ainda, descrições mais completas, como pessoa assistindo tv na sala, pessoa trabalhando no computador;
- Localização dos objetos: Normalmente descritos através de BB dos objetos, onde cada objeto de interesse é anotado através de um retângulo, enquadrando a forma do respectivo objeto. A anotação descreve a posição (x, y) do retângulo na imagem e o tamanho do retângulo, comumente também é anotada a classe à qual o objeto pertence;
- Semântica: A imagem é descrita através da anotação de classes para cada píxel contido na imagem, sendo estas anotações representadas por uma imagem em escala de cinza, onde cada valor representa uma classe. Deste modo, sabe-se a forma e a classe dos objetos presentes na imagem;
- Instância: Semelhante à anotação semântica, porém os objetos recebem identificação única, permitindo diferenciar objetos de uma mesma classe; e
- Pose: Anotação da orientação dos objetos contidos na classe.

2.2.1.2 Datasets

Os *datasets* em visão computacional são conjuntos de imagens públicas captadas seguindo uma metodologia definida pelos autores, aplicados para uma determinada abordagem. Existem uma ampla gama de categorias de *datasets* para os diferentes objetivos, por exemplo *datasets* para objetos isolados, faces humanas, ruas urbanas, ambientes internos, gestos e ações humanas, dentre outros.

Os *datasets* são fundamentais para que trabalhos sejam desenvolvidos sem a necessidade (ou impossibilidade) de realizar a captação própria das imagens, como no caso deste trabalho. Também são importantes para obtenção de imagens de ambientes de diferentes localizações e diferentes câmeras. Além de permitir que diferentes abordagens, de diferentes autores, sejam comparadas e analisadas sob o mesmo conjunto de imagens.

Existem *datasets* que fornecem apenas a metodologia aplicada e as imagens captadas, outros fornecem também o GT semântico, BB, instância, pontos de nuvem,

normais de superfície, dentre outras informações, dependendo dos recursos disponibilizados aos autores, uma vez que a criação de um *dataset* é uma tarefa custosa.

Neste trabalho foi selecionado o *dataset* SUN-RGBD para o treinamento e validação dos algoritmos ML. O *dataset* possui 10.335 imagens RGB-D, com 146.617 polígonos 2D anotados (objetos semânticos) e 64.595 BB (SONG; LICHTENBERG; XIAO, 2015).

Uma relação de *datasets* RGB-D públicos foi elaborada em (FIRMAN, 2016), o autor descreve os *datasets* existentes para diferentes categorias, como *datasets* de objetos isolados, rastreamento de objetos e reconstrução de cenas, estimação de pose de objetos, classes semânticas, ações, gestos e faces humanas, reconhecimentos de indivíduos (através de padrões como marcha humana) e sintéticos (imagens geradas por computador).

2.2.2 Abordagens tradicionais

As abordagens tradicionais são técnicas e algoritmos de processamento digital de imagens, que possuem o objetivo de extrair representações e descrições das imagens, através da aplicação de técnicas morfológicas na imagem, onde dada uma entrada, uma imagem resultante é obtida, quanto a técnicas descritivas, onde atributos são obtidos.

Diferentemente das abordagens ML, onde a rede neural tem como objetivo aprender a como extrair as características das imagens através de um treinamento supervisionado, as abordagens tradicionais tratam da aplicação de sucessivos algoritmos de PDI que auxiliem a obter o resultado desejado, para cada problema proposto, de modo a oferecer uma solução específica para a qual foi desenvolvida.

A maioria das técnicas são baseadas em duas propriedades sobre os valores da imagem: descontinuidade e similaridade. A primeira abordagem trata de segmentar a imagem em regiões baseadas na mudança abrupta de intensidade entre os píxeis, como no caso de bordas de objetos, segmentando as regiões conforme estão incluídas dentro das bordas. Nesta abordagem estão incluídas as técnicas de detecção de bordas como *Sobel*, *Prewitt*, *Roberts* e *Canny*. A segunda abordagem trata de segmentar os objetos em regiões baseados na similaridade dos píxeis, através de critérios. São baseadas em regiões as técnicas *thresholding* (limiarização), *clustering* (agrupamento), *region splitting and merging* (divisão e união de regiões), *region growing* (crescimento de regiões) (LAPLANTE, 2018).

Para compreensão dos conceitos, algoritmos e abordagens para segmentação de imagens através de técnicas tradicionais, as seções abaixo descrevem os conceitos fundamentais e principais técnicas existentes.

2.2.2.1 *Threshold* (Limiarização)

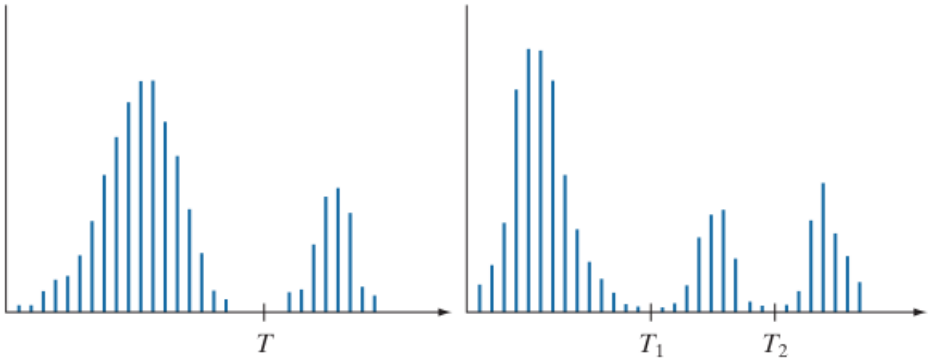
Limiarização é um método básico para segmentação por ser intuitivo e computacionalmente rápido. A base de uma limiarização consiste em segmentar a imagem através de uma seleção baseada em um limiar. A Equação (8) define um limiar T global, resultando em uma binarização da imagem, que pode ser compreendida como segmentar um objeto do seu fundo, em uma imagem.

$$g_{(x,y)} = \begin{cases} 1 & \text{se } f(x,y) > T \\ 0 & \text{se } f(x,y) \leq T \end{cases} \tag{8}$$

Uma limiarização pode ainda utilizar múltiplos limiares, em casos onde existem mais de um ponto dominante de intensidade em uma imagem, conforme definido na Equação (9). A Figura 9 apresenta dois histogramas com o comportamento de apresentarem um ponto visível para ser limiarizado (ponto T), e outro com dois pontos possíveis (pontos $T_1 e T_2$).

$$g_{(x,y)} = \begin{cases} a & \text{se } f(x,y) > T_2 \\ b & \text{se } T_1 < f(x,y) \leq T_2 \\ c & \text{se } f(x,y) \leq T_1 \end{cases} \tag{9}$$

Figura 9 – Exemplos de histogramas com um limiar T e T_1, T_2

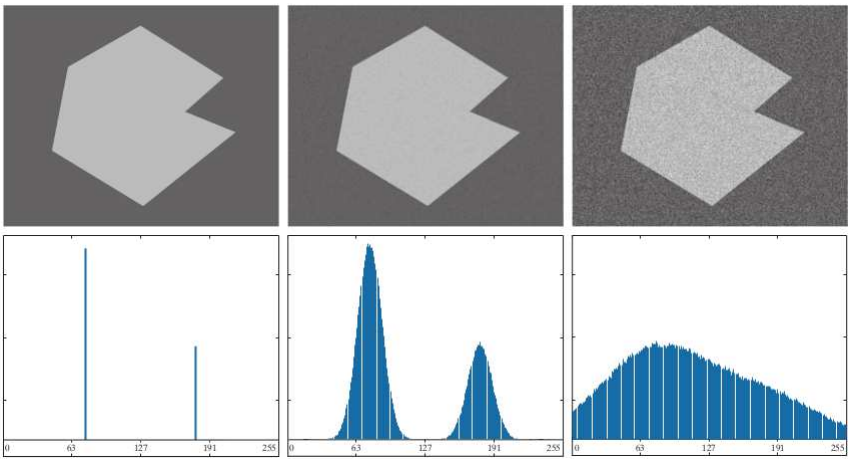


Fonte: (GONZALEZ; WOODS, 2011)

Entretanto, a escolha de um ou mais limiares T pode ser uma tarefa difícil, à medida que há um aumento de ruído na imagem, ou mesmo uma texturização, conforme exibe a Figura 10. A primeira e a segunda figura apresentam em seu histograma um vale bem visível, sendo um ponto para a escolha de um limiar, já na última, com uma distribuição bem maior de intensidades, a escolha de um limiar pode não apresentar um resultado satisfatório.

Em situações onde não é possível definir um único limiar global, pode-se pensar em um algoritmo para calcular o limiar com base na imagem desejada:

Figura 10 – Comportamento do histograma com o aumento de ruído em uma imagem.



Fonte: (GONZALEZ; WOODS, 2011)

1. Inicie com um valor estimado global T;
2. Segmente a imagem através de T, produzindo dois grupos de píxeis (G_1 e G_2);
3. Calcule a média de intensidade em cada grupo (m_1 e m_2);
4. Calcule a média de intensidade em cada grupo (m_1 e m_2)
5. Calcule um novo limiar entre m_1 e m_2: $T = 1/2(m_1 + m_2)$; e
6. Repita dos passos 2 ao 4, até que as diferenças em um ΔT seja menor que um valor predefinido.

Segundo (GONZALEZ; WOODS, 2011), a limiarização pode ser vista como um problema da teoria de decisão e estatística, que tem como objetivo minimizar o erro da segmentação dos píxeis em grupos. Existem também outros algoritmos de limiarização mais complexos, como o método de limiarização *Otsu*, que realiza uma limiarização binária automática, através da minimização da variância entre as classes 0 e 1 (SOLOMON; BRECKON, 2011).

Normalmente, na segmentação de imagens, usa-se combinações de técnicas de PDI, como técnicas de operações morfológicas (correções de ruídos na imagem), detecção de bordas e limiarização.

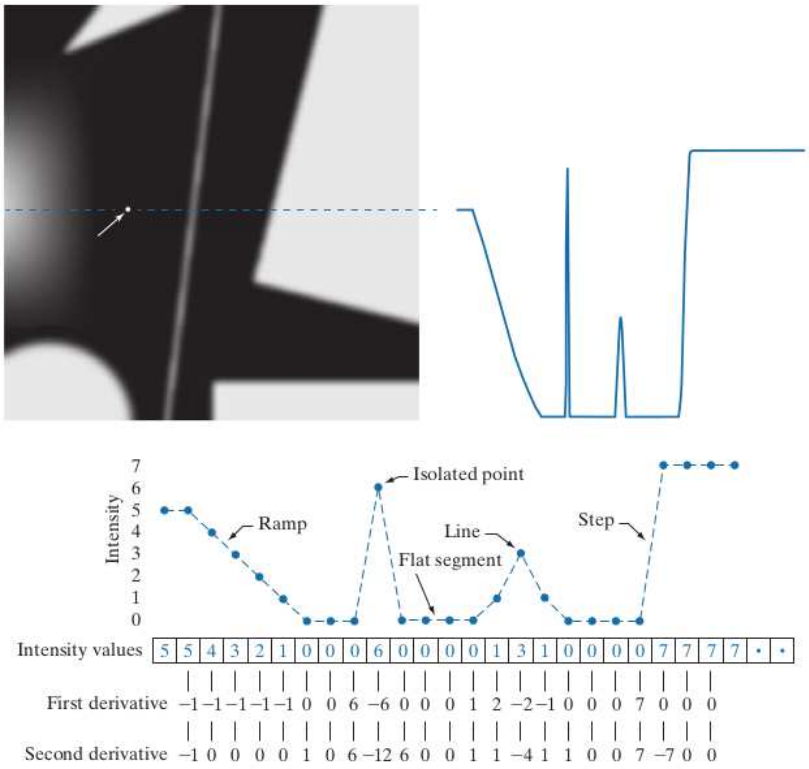
2.2.2.2 Detecção de bordas

A detecção de bordas tem como objetivo isolar pontos, linhas e bordas nas imagens, através da mudança abrupta de intensidade entre os píxeis. Detectores de bordas aplicam processamentos locais na imagem de forma a destacar esses píxeis,

enquanto que existem outros algoritmos para conectar esses segmentos de bordas, os quais podem possuir pontos com interrupção (falhas) nas bordas.

As mudanças de intensidade em uma imagem podem ser detectadas usando derivadas de primeira e segunda ordem, definidas através de finitas diferenças. Para entender como isso é aplicado, a Figura 11 apresenta o comportamento das derivadas em uma linha azul traçada à uma imagem com diferentes intensidades. Os valores de intensidade da linha estão apresentados no vetor *Intensity values*, seguido dos vetores contendo os valores das derivadas de primeira e segunda ordem, obtidos do vetor de intensidade. Pode-se observar a linha, da esquerda à direita, iniciando em uma área que perde intensidade a medida que avança para a direita, para até a ausência de intensidade (valor 0), para então encontrar um pequeno ponto de alta intensidade (*Isolated point* de valor 6), indicado pela seta, seguindo com baixa intensidade até atravessar uma linha vertical, e continuando até chegar a uma borda, onde a intensidade baixa aumenta imediatamente, permanecendo até o fim.

Figura 11 – Cálculo de uma linha das derivadas de primeira e segunda ordem a uma imagem monocromática.



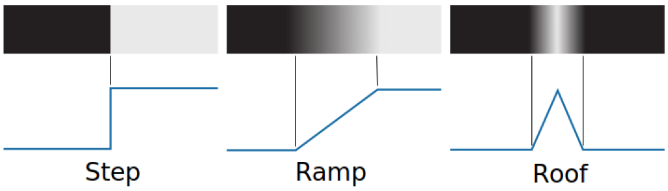
Fonte: (GONZALEZ; WOODS, 2011)

A derivada de primeira ordem inicia com valor diferente de 0 durante todo o avanço progressivo da rampa (*Ramp*), de alta intensidade até a baixa, enquanto que a de segunda ordem é diferente de 0 apenas no início e ao fim da rampa. Segundo

(GONZALEZ; WOODS, 2011), devido às bordas das imagens serem semelhantes a esse tipo de transição entre intensidades (rampa), pode-se concluir que a derivada de primeira ordem identifica bordas mais grossas, enquanto que a de segunda ordem bordas mais finas, sendo também mais sensível a ruídos, como visto no traço que passa pela linha vertical, apresentando um valor de maior magnitude, em relação à primeira ordem. Uma outra propriedade da derivada de segunda ordem é a "borda dupla", conforme observado ao final da rampa e na última borda, que determina se uma das abordas está transitando da alta intensidade para a baixa (valor negativo) ou da baixa para a alta intensidade (valor positivo).

As derivadas de primeira e segunda ordem são utilizadas pelos algoritmos de detecção de bordas. Para detecção de pontos isolados, a derivada de segunda ordem é utilizada, podendo ser obtido através de um *kernel* laplaciano e um limiar para considerar quais diferenças são consideradas como pontos isolados. Assim como linhas podem ser detectadas utilizando um *kernel* laplaciano, porém, considerando as bordas duplas. A Figura 12 define os possíveis de padrões de bordas: salto (bordas com grande diferença de intensidade); rampa (com as mudanças gradativas de intensidade); e telhado (breve mudança brusca de intensidade) (GONZALEZ; WOODS, 2011).

Figura 12 – Tipos de bordas de salto, rampa e telhado.



Fonte: Adaptado de (GONZALEZ; WOODS, 2011)

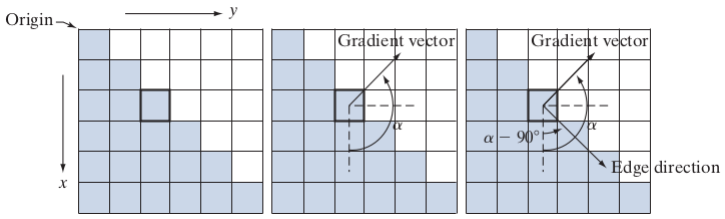
Dentre as propriedades da imagem envolvidas no processo de detecção de bordas e cálculo de derivadas, o ruído e texturas granulares causam grande influência nas operações de detecção de bordas existentes. Nesses casos, é de interesse a aplicação de operações morfológicas sobre a imagem, antes da detecção de bordas, como por exemplo, operações de suavização e eliminação de ruídos (GONZALEZ; WOODS, 2011).

Gradiente

O cálculo do gradiente de um determinado local da imagem permite encontrar a força e direção da borda, através do cálculo da derivada na direção x e direção y de um ponto, conforme a Figura 13.

Uma abordagem para determinar as derivadas de x e y é uma operação usando uma janela de 3x3 vizinhos, com o píxel ao centro, subtraindo a linha de cima com

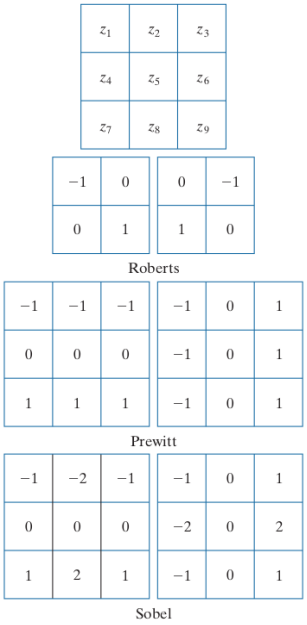
Figura 13 – Zoom em uma região de uma imagem apresentando a direção do gradiente de um píxel.



Fonte: (GONZALEZ; WOODS, 2011)

a linha de baixo para obter uma derivada parcial na direção x, e subtraindo a coluna da esquerda com a direita, obtendo uma derivada parcial na direção y, conforme apresentado um trecho de imagem 3x3 na Figura 14, com z vizinhos, aplicados conforme Equação (10) e Equação (11) (GONZALEZ; WOODS, 2011).

Figura 14 – Janela de vizinhos $z_1 ... z_9$ e diferentes filtros para cálculo do gradiente.



Fonte: (GONZALEZ; WOODS, 2011)

$$g_x = \frac{\partial f}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3). \tag{10}$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7). \tag{11}$$

Existem diferentes operações para o cálculo de gradiente de uma imagem. Em PDI, são utilizados *kernels* (filtros) que possibilitam a detecção de borda através do

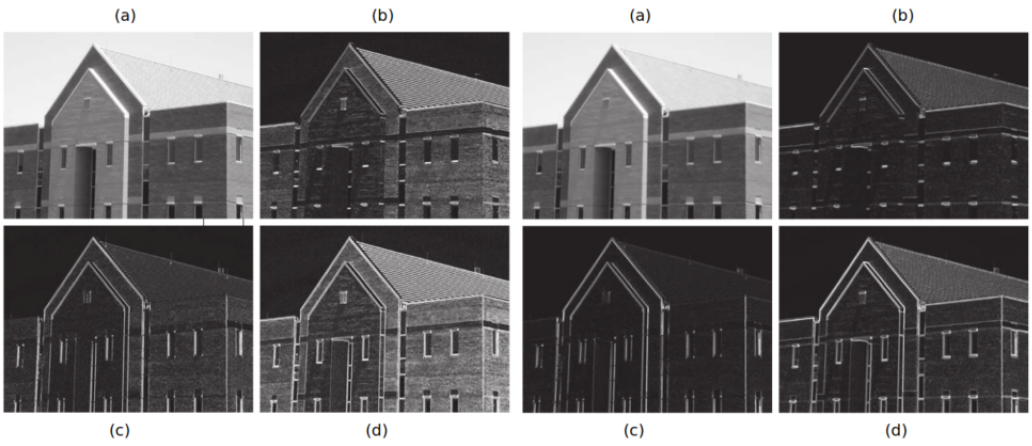
cálculo de gradientes aproximados, onde, através de dois filtros, são obtidos as regiões derivadas na direção x e direção y, como os *kernels Roberts*, *Prewitt* e *Sobel* da Figura 14. A detecção é obtida realizando o processo de filtro (deslize) desses *kernels* sobre a imagem, somando os resultados das derivadas na direção x e y. Esses filtros são utilizados para obtenção da predominância das bordas nas direções horizontais e verticais (GONZALEZ; WOODS, 2011).

O *kernel* é aplicado à uma imagem *I* de resolução HxW (height x width - altura x largura) através da Equação (12). Considerando um *kernel* 3x3, onde o centro do *kernel* refere-se ao píxel em que está sendo aplicado o filtro na imagem. Caso o píxel, dada uma imagem *I*, seja $I_{(1,1)}$ (primeiro píxel), é aplicado o filtro apenas em z_5, z_6, z_8, z_9, que representam os píxeis $I_{(1,1)}, I_{(1,2)}, I_{(2,1)}, I_{(2,2)}$ (GONZALEZ; WOODS, 2011).

$$Z = \sum_{k=1}^{9} \omega_k z_k = \omega_1 z_1 + \omega_2 z_2 + ... + \omega_9 z_9. \tag{12}$$

A Figura 15 apresenta a aplicação da detecção de bordas utilizando o filtro *Sobel*, demonstrando a diferença dos resultados nas direções x e y. Pode-se notar, nas imagens da figura (b), as bordas horizontais bem destacadas do telhado, e na figura (c), as bordas verticais podem ser observadas.

Figura 15 – Exemplo da aplicação da operação *Sobel* em uma imagem original (figura à esquerda) e imagem suavizada (figura à direita): (a) imagem original, (b) *Sobel* direção x, (c) *Sobel* direção y, (d) soma das direções.



Fonte: (GONZALEZ; WOODS, 2011)

Em relação ao problema de ruídos e texturas, uma operação de suavização permite que as bordas sejam identificadas com maior distinção. Além da suavização, uma limiarização pode ser aplicada para tornar seletiva a detecção de bordas, por exemplo, aplicando um limiar onde são mantidos os píxeis cujos valores sejam 33%

maiores que o valor máximo obtido. Este comportamento pode ser observado na Figura 16, observa-se, através da seta, que as bordas do telhado na imagem sem suavização apresentam descontinuações em relação ao mesmo local na imagem em que houve suavização.

Figura 16 – Exemplo de limiarização nas figuras (d) da Figura 15

.

Fonte: (GONZALEZ; WOODS, 2011)

Existem outros *kernels*, como o *Kirsch*, que são capazes de detectar as bordas nas 8 direções (ângulos) de um píxel. O resultado é obtido através do cálculo de 8 convoluções sobre a imagem, considerando como resultado os pontos de maior magnitude de cada posição dentre as 8 possibilidades (GONZALEZ; WOODS, 2011).

Detector de bordas *Canny*

Canny é um algoritmo detector de bordas mais complexo, onde leva em conta os ruídos presentes na imagem, além da própria natureza das bordas presentes na imagem. Baseia-se em três princípios:

1. Baixa taxa de erro: Todas as bordas devem ser encontradas, sem respostas falsas;
2. Bordas devem ser bem localizadas: As bordas identificadas devem ser as mais próximas possíveis das bordas reais; e
3. Único ponto de borda: O detector deve retornar apenas um ponto de borda para cada ponto de borda verdadeiro.

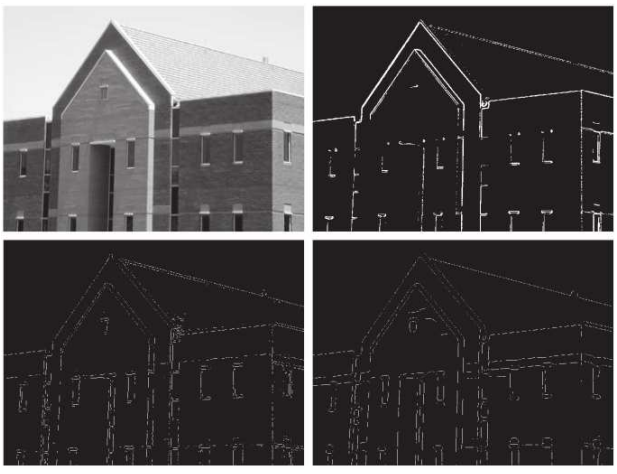
Embora os critérios sejam rígidos, o detector *Canny* aplica matematicamente os princípios, de forma a obter uma boa aproximação da solução ótima. O algoritmo, segundo (SOLOMON; BRECKON, 2011), aplica os seguintes passos para realizar a detecção de bordas:

1. Aplica um operador gaussiano para suavizar os ruídos da imagem. Quanto maior o comprimento do *kernel*, maior a suavização (possibilitando também maior erro na detecção);

2. Encontra a força das bordas através do cálculo de gradiente, utilizando o filtro de *Sobel* nas direções horizontais e verticais, utilizando como medida a magnitude dos componentes: $E(x,y) = |G_X(x,y)| + |G_Y(x,y)|$;
3. Calcula a direção da borda através da equação: $\theta = \tan \frac{G_x(x,y)}{G_y(x,y)}$;
4. A direção da borda é aproximada para uma direção que seja possível de ser traçada na imagem, dentre as quatro possibilidades: 0° (vizinhos do leste e oeste), 90° (vizinhos do norte e sul), 4° (vizinhos do nordeste e sudoeste) e 135° (vizinhos do noroeste e sudeste);
5. Após conhecidas as direções dos píxeis, traça-se uma linha seguindo a direção dos píxeis, onde os píxeis fora dessa linha são suprimidos; e
6. Por fim, os limiares mínimos e máximos são aplicados.

A Figura 17 demonstra a aplicação do algoritmo *Canny*, em comparação com o algoritmo *Marr-Hildreth* (algoritmo ao qual Canny foi baseado) e uma limiarização aplicada à uma suavização da imagem original. O algoritmo *Canny* tende a apresentar resultados com as bordas mais significantes, resultando em uma imagem com menos bordas a serem processadas por algoritmos posteriores (GONZALEZ; WOODS, 2011).

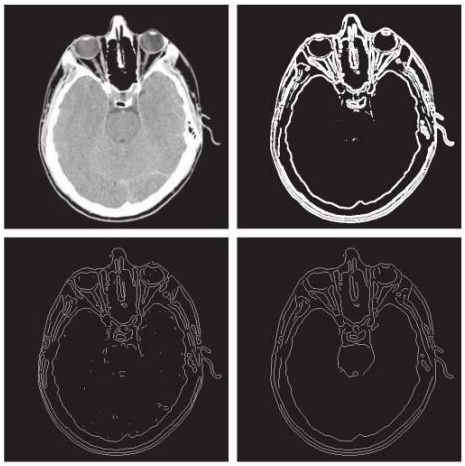
Figura 17 – Exemplo de detecção de bordas, onde respectivamente: imagem original, limiarização, algoritmo *Marr-Hildreth* e algoritmo *Canny*.



Fonte: (GONZALEZ; WOODS, 2011)

Outro exemplo de aplicação é em tomografia computadorizada, utilizado para extrair o contorno de fora do cérebro, região nasal e contorno da cabeça. A Figura 18 apresenta o resultado, comparando a imagem original com uma limiarização da imagem original suavizada, algoritmo *Marr-Hildreth* e algoritmo *Canny*, observa-se as diferentes definições de bordas obtidas pelos algoritmos.

Figura 18 – Aplicação do Canny em uma tomografia de crânio, onde respectivamente: imagem original, limiarização, algoritmo *Marr-Hildreth* e algoritmo *Canny*



Fonte: (GONZALEZ; WOODS, 2011)

2.2.2.3 *Region Growing Segmentation*

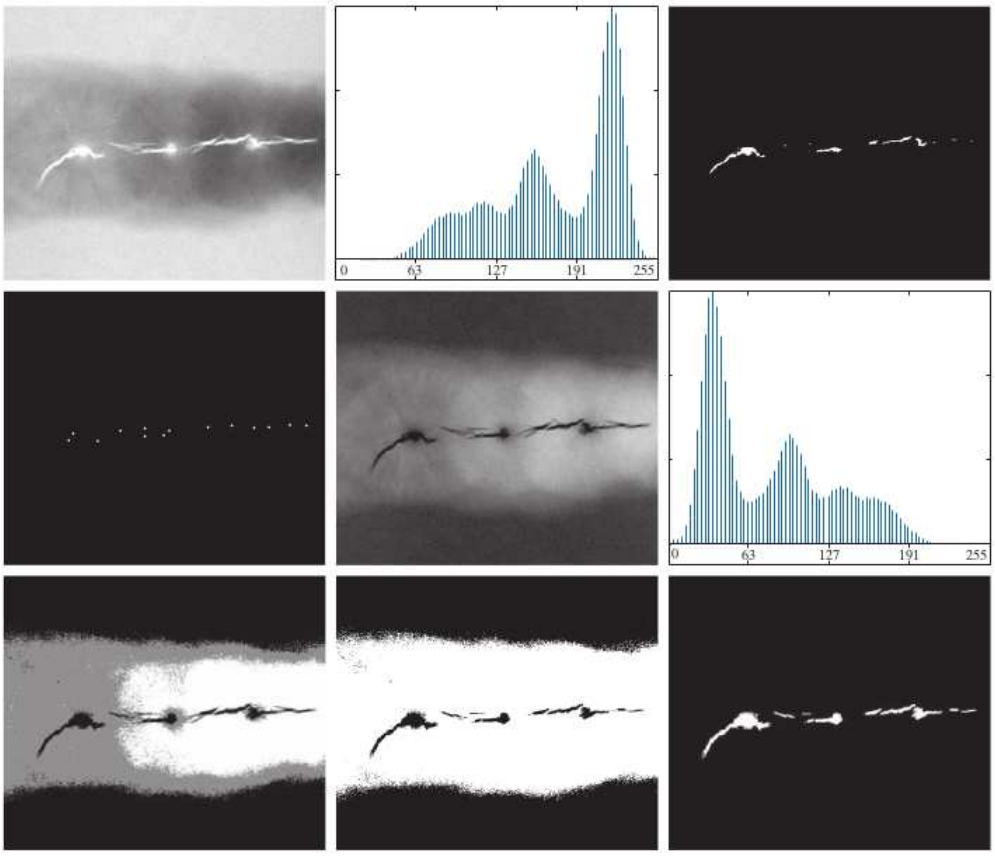
Region growing (crescimento de regiões) é um método de segmentação que tem como objetivo agrupar píxeis em regiões, através de um critério, iniciando com um conjunto de *seeds* (sementes), espalhados pela imagem e agrupando os píxeis similares vizinhos. Seja $f(x,y)$ a imagem; $S(x,y)$ uma matriz de sementes, onde 1 é o ponto com a semente, 0 os demais; Q o critério de semelhança a ser aplicada; e seja também condicionada a conectividade de 8-vizinhos, pode ser estabelecida pelo algoritmo:

1. Encontre todos os componentes em $S(x,y)$ e reduza todos os píxeis conectados até um píxel;
2. Forme uma matriz f_Q, onde para cada ponto (x,y), $f_Q(x,y) = 1$ quando o ponto satisfaz o critério Q, e 0 quando não;
3. Forme uma matriz g através de S, e adicione todos os valores 1 de f_Q em g que sejam vizinhos 8-conectados ao ponto de semente; e
4. Rotule cada componente conectado em g com um rótulo de região diferente (números sequenciais ou letras).

Um exemplo dessa abordagem é apresentado na Figura 19, onde é aplicado o algoritmo de crescimento de regiões para encontrar rachaduras e poros em uma solda. Na primeira linha, a partir da imagem original e histograma, são encontradas as áreas de interesse para estabelecer sementes, pois espera-se que as rachaduras e poros resultem no raio-x em áreas com maior intensidade, sendo estas áreas

identificadas através de uma limiarização 254 simples (sobrando os píxeis 255). As áreas de sementes devem então virar uma unidade menor, através da modificação morfológica erosão (quarta figura). As figuras seguintes, até a penúltima, demonstram a tentativa de aplicação de uma limiarização simples para tentar segmentar a imagem, mas que não apresentam o resultado desejado. A quinta figura apresenta a diferença da primeira figura (que é utilizada para aplicação do critério), e o respectivo histograma. Nota-se a possibilidade de usar os limiares $T_1 = 68$ e $T_2 = 126$, por estarem próximos aos vales no histograma. A sétima figura apresenta o resultado da limiarização dupla, já a oitava figura com apenas T_1, porém, ambas apresentam uma segmentação não desejada (considerando que os pontos 0 são os desejados). Por fim, na nona figura, após as sementes agregarem os píxeis vizinhos 8-conectados que satisfazem o critério de diferença de intensidade através de um limiar (GONZALEZ; WOODS, 2011).

Figura 19 – Respectivamente: imagem de raio-x, histograma, sementes iniciais, sementes finais, diferença da imagem original, histograma, imagem segmentada em dois limiares, imagem segmentada com o mínimo em dois limiares, segmentação obtida pelo crescimento em regiões.



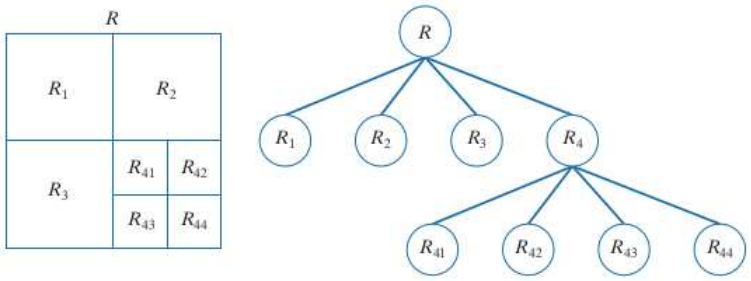
Fonte: (GONZALEZ; WOODS, 2011)

2.2.2.4 *Region Splitting and Merging*

A *Region Splitting and Merging* (divisão e união de região) é uma alternativa à segmentação de regiões com sementes, pois trata de sub-dividir uma imagem em quadrantes, para então unificá-los, usando um determinado critério. A Figura 20 apresenta visualmente a divisão, cujo algoritmo possui como estrutura as seguintes etapas:

1. Divida progressivamente a região em quadrantes, para quando a região R_i não satisfaz o critério $Q(R_i) = FALSO$;
2. Quando não ocorrer mais divisão, unifique regiões adjacentes, das quais dada região R_j e região adjacente R_k, $Q(R_j \cup R_k) = VERDADEIRO$, até que mais nenhuma união seja possível.

Figura 20 – Exemplo de divisão de uma imagem em quadrantes.



Fonte: (GONZALEZ; WOODS, 2011)

2.2.2.5 *Region Segmentation using Clustering and Superpixels*

Uma forma de segmentar uma imagem em regiões é o agrupamento por *k-means*, onde a imagem é particionada em um número especificado, chamado de *cluster*. Segundo (GONZALEZ; WOODS, 2011), um algoritmo comum, dado vetor de entrada $z_1, z_2, ..., z_Q$, um valor k e o vetor de interesse $C = C_1, C_2, ..., C_k$, é definido por:

1. Inicialize um vector de médias, $m_i(1), i = 1, 2, ..., k$;
2. Atribua cada valor do vetor de entrada ao *cluster* onde a média (através da distância euclidiana) é a mais próxima (adicionado a apenas um *cluster*): $z_q \rightarrow C_i$ se $||z_q - m_i||^2 < ||z_q - m_j||^2$; $j = 1, 2, ..., k (j \neq i); q = 1, 2, ..., Q$;
3. Atualize o vetor de médias: $m_i = \frac{1}{|C_i|} \sum_{z \in C_i} z$; $i = 1, 2, ..., k$, onde $|C_i|$ é o número de exemplos dentro do conjunto C_i; e

4. Calcule as normas euclidianas das diferenças entre os vetores de médias, da etapa atual e da etapa anterior, obtendo o erro residual E. Para quando $E \leq T$, onde T é um limiar especificado.

A Figura 21 apresenta o resultado de uma aplicação do algoritmo, com $k = 4$, onde a imagem é segmentada em diferentes regiões, mas limitadas a 4 classes.

Figura 21 – Exemplo de aplicação da segmentação de região *k-means*, com $k = 4$

.

Fonte: (GONZALEZ; WOODS, 2011)

Uma abordagem semelhante é a utilização de super píxeis, como são denominadas pela técnica. Essa abordagem busca mudar a forma como uma imagem é organizada, de uma matriz de píxeis para uma forma em regiões primitivas, onde essas regiões menores possuem maior significado do que individualmente em píxeis. Um algoritmo SLIC *(Simple Linear Iterative Clustering) superpixel* consiste nos seguintes passos (GONZALEZ; WOODS, 2011).

1. Seja $z = [rgbxy]$, onde (r,g,b) são os três componentes de cores, e (x,y) as coordenadas, n_{sp} o número desejado de super píxeis, n_{tp} o número total de píxeis na imagem, e o espaçamento de grade $s = [n_{tp}/n_{sp}]^{1/2}$;

2. Calcule o super píxel inicial dos centros de *clusters*, onde $m_i = [r_i g_i b_i x_i y_i]^T, i = 1,2,...,n_{sp}$, através da divisão da imagem em exemplos em grade regulares de passo s. Mova o centro dos *clusters* para a posição de menor gradiente em uma janela de vizinhança 3x3, a partir do centro original. Ainda, para cada posição de píxel p, atribua um vetor de *labels* L(p) = -1, e outro de distância $d(p) = \infty$;

3. Para cada centro de *cluster* m_i, $i = 1,2,...,n_{sp}$, calcule a a distância $D_i(p)$ entre m_i e cada píxel em $2sx2s$ vizinhos de m_i. Para cada p e i = 1,2,3...n_{sp}, se $D_i <$ d(p), então d(p) = D_i e L(p) = i;

4. Atualize os centros de cada *cluster*. Seja C_i o conjunto de píxeis na imagem com L(p) = i, atualize m_i como:

$m_i = \frac{1}{C_i} \sum_{z \in C_i} z,\ i = 1,2,...,n_{sp},$

onde $|C_i|$ é o número de píxeis no conjunto C_i, e vetor z (valores r,g,b,x,y);

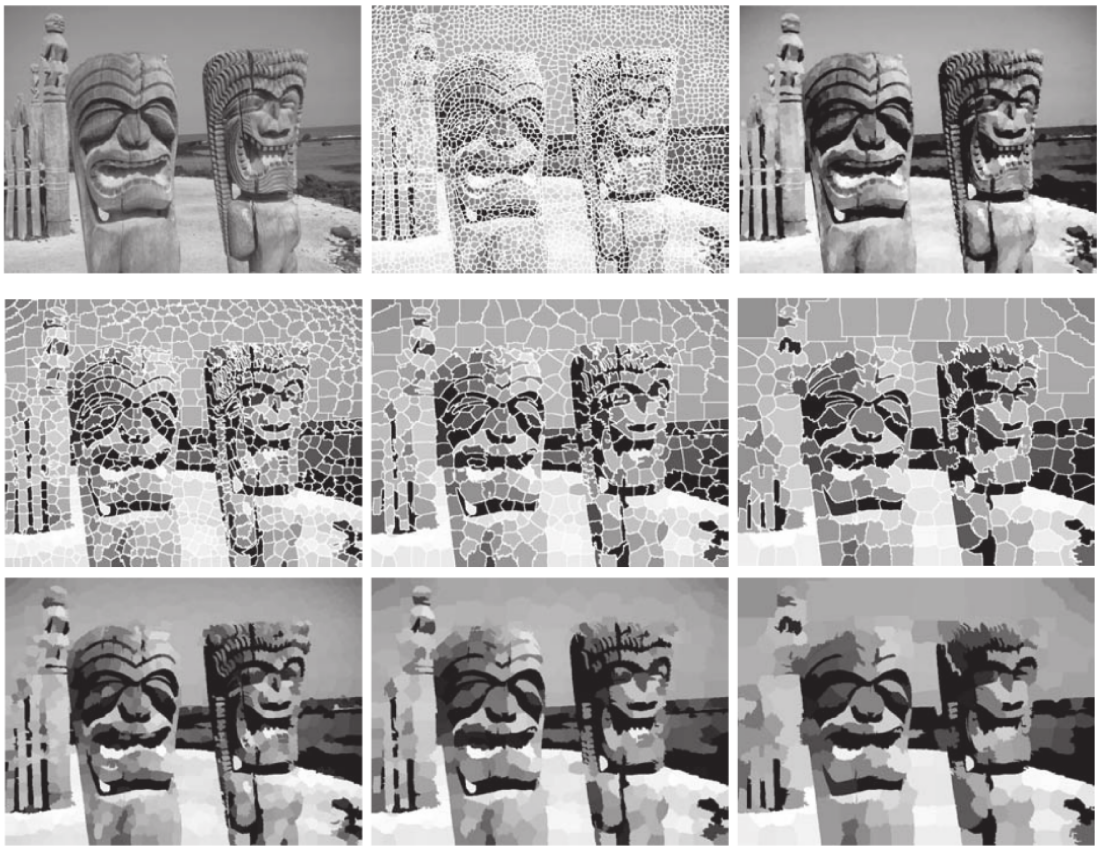
5. Calcule a norma euclidiana das diferenças entre a média de vetores da iteração atual com a anterior. Calcule o erro residual E, através da soma de norma de n_{sp}. Se $E < T$, onde T é um limiar definido, vá para o último passo, se não, volte ao passo 3; e

6. Pós-processamento, substitua todos os super píxeis em cada região C_i pela média em m_i.

A Figura 22 apresenta diferentes resultados obtidos pelo algoritmo, utilizando diferentes parâmetros de super píxel.

Figura 22 – Exemplo de segmentação com super píxeis. A primeira linha com a imagem original e uma segmentação usando 4.000 super píxeis. A segunda e terceira linha usando 1.000, 500, 250. As bordas em branco na linha 1, figura do meio, e linha 2 são apenas para facilitar a visualização

.



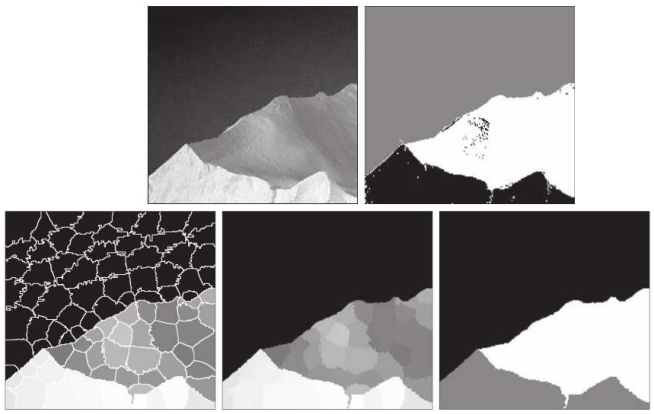
Fonte: Adaptado de (GONZALEZ; WOODS, 2011)

Uma outra forma de mostrar a importância da combinação de algoritmos, além das infinitas possibilidades de desenvolvimento de soluções, é apresentada na Fi-

gura 23, onde são combinadas as segmentações de super píxel e *k-means* para realizar uma segmentação de um *iceberg* em 3 regiões.

Figura 23 – Comparação entre diferentes segmentações: Imagem original, imagem segmentada por *k-means*, super píxel com e sem borda (k = 100), *k-means* aplicado à imagem 4

.



Fonte: (GONZALEZ; WOODS, 2011)

2.2.2.6 *Region GrabCut Segmentation*

Nesta abordagem de segmentação de regiões em corte de grafos, os píxeis de uma imagem são expressos em nós de um grafo, e então é encontrado um corte ótimo (*cut*) do grafo, gerando grupos de nós. Esse tipo de algoritmo tem grande potencial, perante as implementações anteriores, mas tende a ser mais custoso computacionalmente.

A estrutura de um grafo G é constituído por um conjunto de vértices (nós) V e um conjunto de arestas E conectando os vértices, conforme Equação (13).

$$G = (V, E). \tag{13}$$

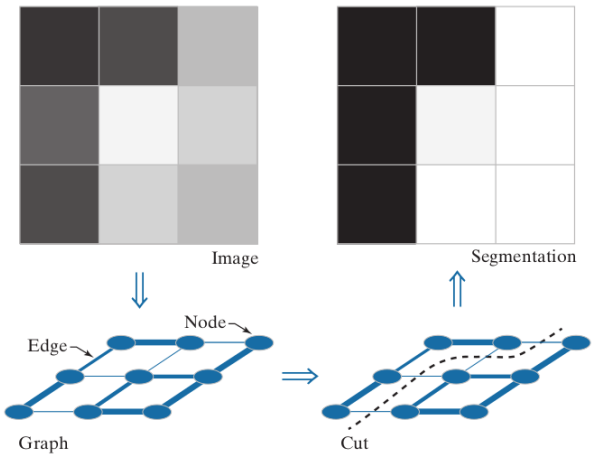
Onde E é um conjunto de pares ordenados de elementos de V. Se $(u, v) \in E$ implica que $(v, u) \in E$, indicando que o grafo é não-direcionado, em caso contrário, o grafo é direcionado (Equação (14)).

$$E \subseteq VxV. \tag{14}$$

Um exemplo simples da relação entre uma imagem, o grafo correspondente, o corte e a segmentação final é apresentada na Figura 24. Cada aresta, que faz a conexão a dois nós, recebe um peso ω, o qual pode variar conforme a solução a ser proposta. Os pesos geralmente são formados pelas relações espaciais (distância) e

intensidade (cor, similaridade). Os pesos podem ser estabelecidos sendo, dado dois nós n_i e n_j, possuem em sua aresta um peso $\omega(i,j) = \frac{1}{|I(n_i)-I(n_j)|+c}$, onde $I(n_i)$ e $I(n_j)$ são funções da similaridade escolhida, e c uma constante para prevenir a divisão por 0.

Figura 24 – Exemplo de uma imagem, respectivo grafo e o corte (segmentação).



Fonte: (GONZALEZ; WOODS, 2011)

Em relação ao algoritmo, os autores (GONZALEZ; WOODS, 2011) descrevem os seguintes passos básicos:

1. Considerando um grafo ponderado, obtido de uma imagem, $G = (V, E)$, calcule o peso das arestas, de forma a construir as matrizes W (pesos) e D (matriz com elementos diagonais). Dado o número desejado de K regiões;

2. Resolva o autovetor $(D-W)_y = \lambda D_y$ para encontrar o segundo menor autovalor;

3. Use o autorvetor do passo anterior para repartir o grafo encontrando o ponto de divisão em que $Ncut(A,B)$ é minimizado;

4. Se o número de cortes não alcança K, verifique se o corte corrente deve ser subdividido checando a estabilidade do corte; e

5. Recursivamente reparta as partes se necessário.

A função *Ncut* é baseada no teorema *Min-Cut*. Interpretando o grafo como uma rede de fluxo de tubos, o teorema estabelece que, o caminho de maior fluxo possível da origem até o fundo é igual ao caminho mínimo. O corte mínimo é definido pelo menor peso total das arestas que, se removidas, desconectariam a fonte do fundo.

A Figura 25 apresenta a aplicação do algoritmo. Segundo (GONZALEZ; WOODS, 2011), o algoritmo é ideal para obter as principais regiões aproximadas de uma imagem.

Figura 25 – Exemplo de resultado de segmentação do método *GrabCut*, especificando duas regiões, aplicado à imagem suavizada (centro).

Fonte: (GONZALEZ; WOODS, 2011)

2.2.2.7 *Watershed Based Segmentation*

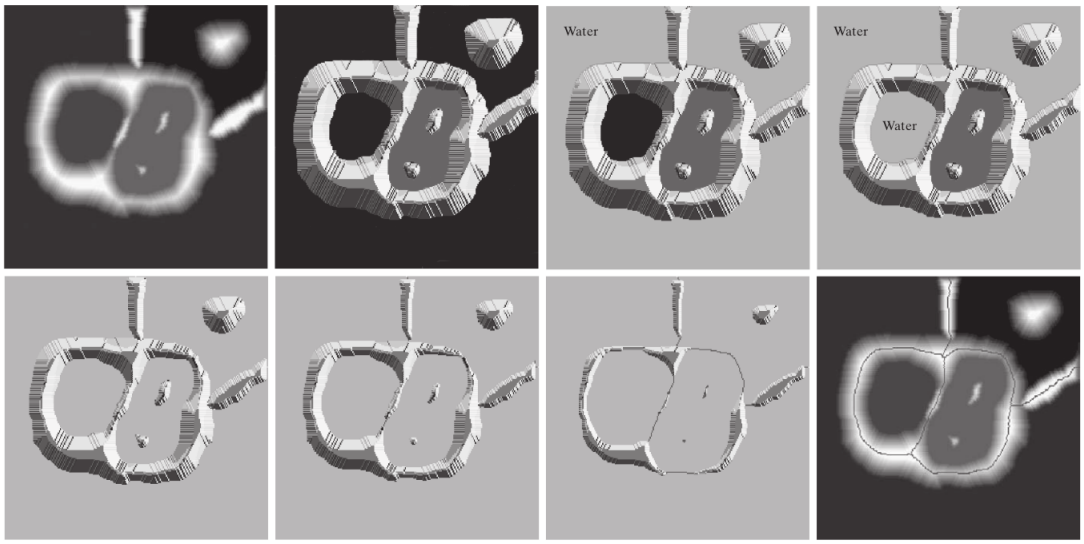
A concepção desse algoritmo consiste em visualizar a imagem em 3D (considerando uma imagem em escala de cinza), duas dimensões espaciais e uma de intensidade, de forma a calcular uma profundidade da imagem através da intensidade, obtendo uma visão topográfica da imagem. A ideia é baseada também no processo de inundação e enchimento de água. Considerando essa interpretação, podem existir 3 tipos de pontos: pontos pertencendo a uma região mínima (baixa); pontos de montanha em que, caso colocada uma bota de água, cairia com certeza em uma região mínima; e pontos onde a água poderia cair em qualquer uma das regiões mínimas.

O algoritmo parte do princípio que, uma vez encontrado os tipos de pontos na imagem, a segmentação da imagem pode ser obtida através de consecutivas taxas de adição de “água” nos pontos de região mínima, até que os mesmos atinjam o limite de suas bordas, processo este apresentado na Figura 26. O principal objetivo do algoritmo é encontrar as linhas de divisão de águas. Devido a isso, uma grande vantagem do algoritmo é obter a segmentação bem próxima às bordas, mantendo também as bordas contínuas entre as regiões (GONZALEZ; WOODS, 2011).

2.2.2.8 Aplicações em imagens coloridas

A maioria dos conceitos e algoritmos apresentados, como também em outros da literatura, utilizam uma imagem monocromática de exemplo, considerando apenas imagens 2D com um canal e tratando os valores como intensidade dos píxeis. Em contrapartida, as imagens coloridas possuem 3 canais de informação em cada píxel. Entretanto, os algoritmos podem ser adaptados para serem aplicados às imagens RGB. As medidas de intensidade, utilizadas nas imagens de um canal, podem ser substituídas pela aplicação de uma função aos valores RGB, como por exemplo funções de distância, como distância euclidiana, aplicadas aos valores RGB. Em alguns casos, pode-se considerar aplicar o algoritmo em cada canal separadamente, para

Figura 26 – Exemplo de aplicação do algoritmo, com progressivas taxas de adições de água nas regiões baixas (escuras).



Fonte: Adaptado de (GONZALEZ; WOODS, 2011)

então somar os resultados, ou ainda, converter o modelo de cor RGB para outros modelos, como HSI (hue, saturation, intensity - matiz, saturação, intensidade) e CMYK, aplicando em cada canal separado, ou em apenas um canal (GONZALEZ; WOODS, 2011).

O modo HSI é interessante, pois apenas o canal matiz já representa satisfatoriamente bem a cor. O canal de saturação é usado com menos frequência, em geral utilizado para isolar regiões de interesse do canal da matiz (GONZALEZ; WOODS, 2011). Pode-se ainda, como opção, aplicar uma transformação da imagem RGB em imagem em escala de cinza, de modo a extrair características que podem ser somadas à outras características obtidas através dos 3 componentes de cor.

Existem ainda na literatura inúmeras outras propostas e algoritmos, tanto baseadas em adaptações dos métodos tradicionais, quanto em novas propostas. Portanto, no desenvolvimento de uma solução utilizando as técnicas de PDI, é necessária uma análise do problema e validação das abordagens e algoritmos, de modo a encontrar combinação que melhor se adapte ao problema.

2.2.3 Abordagens Machine Learning

As abordagens *Machine Learning* (ML) se tornaram muito populares nas últimas décadas, tornando-se parte de soluções em diversas áreas. As técnicas de ML foram desenvolvidas para resolver problemas que não podem ser resolvidos facilmente através de uma sequencia de instruções ou equações, problemas onde existe uma dificuldade em estabelecer uma solução computacional satisfatória.

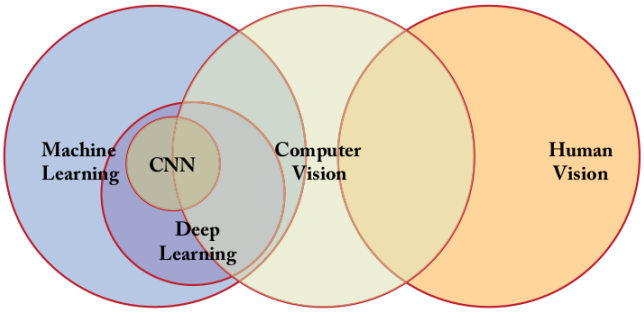
Uma das características dessa abordagem é o aprendizado por experiência, através de dados de exemplos ou experiências passadas, de forma a oferecer como resultado uma predição, um valor aproximado, sem que se tenha definido explicitamente uma lógica para resolução do problema. Além disso, pode-se modelar a abordagem para aprender com as suas próprias predições, possibilitando evolução e melhoria do modelo com o tempo (KHAN *et al.*, 2018).

Existem três formas de aprendizado dos algoritmos ML: supervisionados, semi-supervisionados e não-supervisionados. Na supervisionada, o treinamento da rede é realizado através de um *dataset* contendo pares de (dado, rótulo), permitindo que o algoritmo processe o dado, realize uma predição *y'* e compare com o rótulo *y* (também chamado de GT, classe ou *label*), permitindo medir e aprender com o erro. No caso da não-supervisionada, não há a existência de rótulo no *dataset*, o próprio algoritmo, através dos dados, procura estabelecer e aprender os padrões existentes no *dataset*. Já na semi-supervisionada, nem toda informação do *dataset* possui o rótulo, sendo um meio termo entre as formas de treinamento (KHAN *et al.*, 2018).

Existem diversos modelos estatísticos de ML, como árvores de decisão, SVM (*Support Vector Machine*), classificador bayesiano, regras de associação, algoritmos genéticos e redes neurais (ALPAYDIN, 2010). O modelo mais utilizado, dentre os trabalhos encontrados de segmentação de imagens, é a rede neural.

Uma termo muito empregado atualmente é o *deep learning* (aprendizagem profunda). Esse termo engloba técnicas de ML robustas, com alta computação e larga escala de dados, normalmente empregados em tarefas que visam simular a forma humana de resolver um determinado problema.

Figura 27 – Relação entre Machine Learning, Deep Learning, CNN, Computer Vision (Visão Computacional) e Visão Humana.



Fonte: (KHAN *et al.*, 2018)

A Figura 27 apresenta uma relação dos conceitos presentes na segmentação de objetos, onde a visão computacional engloba uma parte da capacidade de visão humana e engloba as técnicas de ML como sendo as ferramentas, ou meios, para o

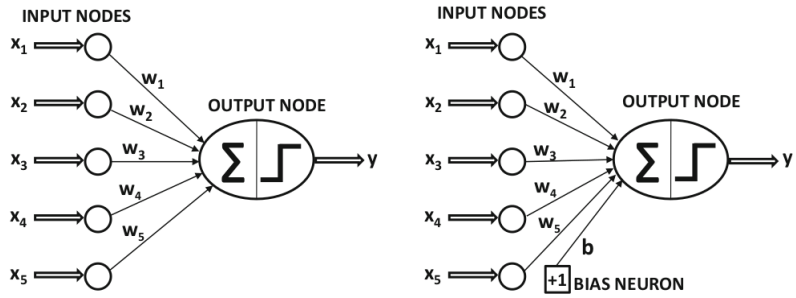
desenvolvimento das soluções (KHAN *et al.*, 2018). Dentro da visão computacional ainda constam todas as técnicas de PDI, incluindo as abordagens tradicionais.

As seções abaixo descrevem em maior detalhe os principais conceitos das redes neurais e as diferentes arquiteturas.

2.2.3.1 Redes Neurais Artificiais

As redes neurais artificiais (RNA) são técnicas computacionais que visam simular, de forma mais simples, o aprendizado de organismos biológicos. Em uma rede neural simples, de uma camada, um conjunto de entradas são diretamente mapeados a uma saída, através de uma função de ativação.

Figura 28 – Arquitetura básica de uma *perceptron*.



(a) Perceptron without bias (b) Perceptron with bias

Fonte: (AGGARWAL, 2018)

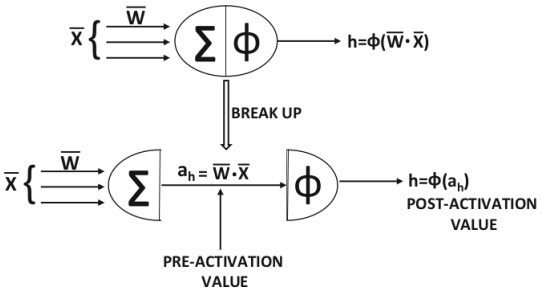
Essa configuração de rede neural mais simples, de uma camada, também é chamada de *perceptron*, ou neurônio. A Figura 28 apresenta os elementos básicos de uma *perceptron*, com um conjunto de entradas (x_i), conectados a uma saída y, com cada aresta possuindo um peso (w_i). A saída recebe como entrada o produto das entradas e aplica uma função sinal (função de ativação) para converter o produto das entradas para um valor de faixa restrita.

Em alguns casos, pode-se desejar um valor de entrada *bias* (viés) com valor constante, para ser sempre adicionado junto às demais entradas, de modo a serem utilizados para influenciar o resultado da saída.

A Figura 29 apresenta em resumo uma *perceptron* (neurônio), composto por uma soma de valores, multiplicados pelos seus pesos (valores pré-ativação), resultando em um valor pós-ativação, após a aplicação da função de ativação (ϕ).

Considerando, para a rede neural *perceptron*, um treinamento onde cada instância de treino é dada por $\overline{X} = [x_1, ..., x_d]$, contendo um número d de características, e $y \in \{-1, +1\}$ como saída binária esperada. A Equação (15) estabelece a relação entre a aplicação de uma instância do conjunto de treino, em uma *perceptron*, onde a predição

Figura 29 – Detalhe de um nó, com a soma dos pesos, valor de pŕé-ativação, função de ativação e valor pós-ativação.



Fonte: (AGGARWAL, 2018)

y' será dada pela função sinal (função de ativação), mapeando as entradas e pesos, resultando em uma predição binária (AGGARWAL, 2018).

$$\text{y'} = sinal\{\overline{W}.\overline{X} + b\} = sinal\{\sum_{i=1}^{n} w_i x_i + b\}. \tag{15}$$

O erro, dado o conjunto de entradas $\overline{X}$, a saída desejada y e uma predição y', pode ser estabelecido pela Equação (16). Se o erro for diferente de 0, os pesos da *perceptron* podem ser atualizados na direção negativa ao gradiente de erro. Esse processo é utilizado no treinamento de redes neurais para o aprendizado.

$$\text{E}(\overline{X}) = \text{y - y'}. \tag{16}$$

Para treinamento, dado um *dataset* D, com pares $\overline{X}$ e y (valor real), pode-se obter uma função de minimização de erro (*loss function*), utilizado para a medição de erro e ajuste dos pesos. A função de perda é importante no processo de ajuste de pesos das RNAs. A Equação (17) estabelece a relação entre a função de erro para um *dataset* D aplicado à *perceptron* (AGGARWAL, 2018).

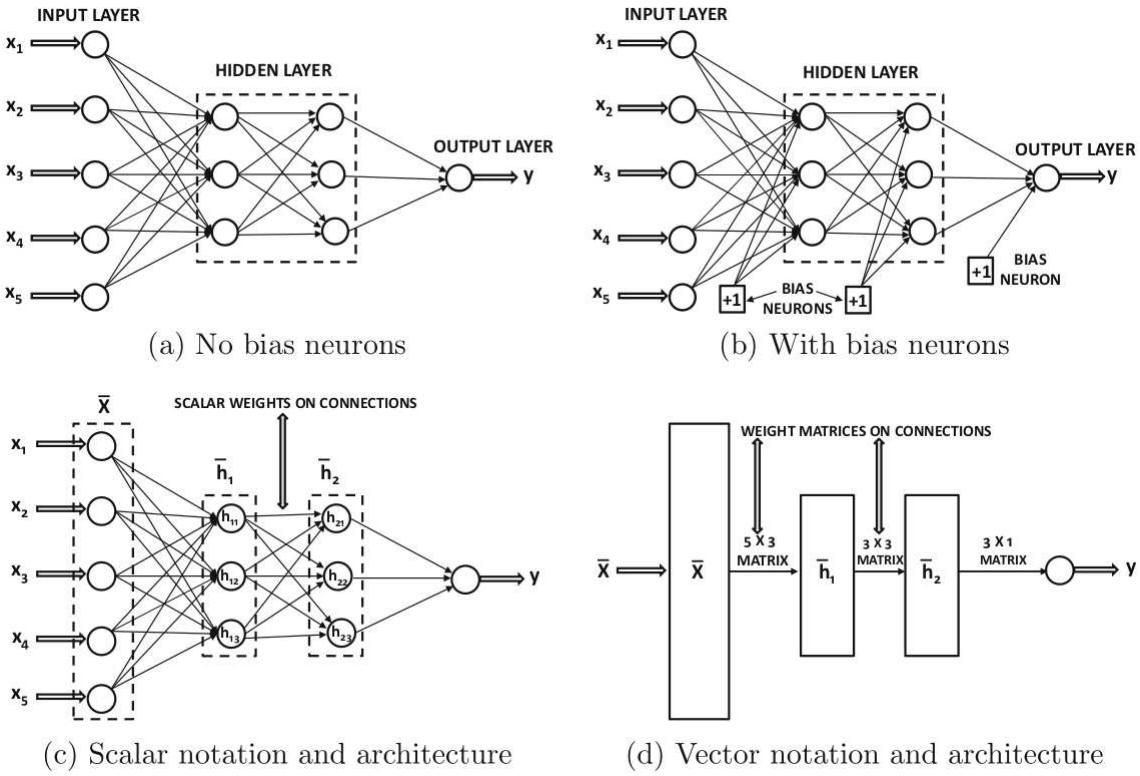
$$\text{Minimização}_{\overline{X}} L = \sum_{(\overline{X},y)\in D} (y - y')^2. \tag{17}$$

Em redes neurais multicamadas (MLP - *Multi-layer Perceptron*), os neurônios são organizados em camadas, onde entre a camada de entrada e saída, existem camadas intermediárias, também chamadas de camadas ocultas. O modelo de MLP também é denominado de *feed-forward network*, devido ao fluxo das informações serem em apenas uma direção.

A Figura 30 apresenta um modelo básico de arquitetura multicamada. Podem existir n camadas intermediárias (camadas ocultas). Nesse modelo de arquitetura,

todas as saídas de uma camada são diretamente conectadas a todas as entradas da camada seguinte (AGGARWAL, 2018).

Figura 30 – Arquitetura básica de uma RNA multicamada *feed-forward*.



(a) No bias neurons

(b) With bias neurons

(c) Scalar notation and architecture

(d) Vector notation and architecture

Fonte: (AGGARWAL, 2018)

Retropropagação do Erro (*back-propagation*)

Em uma MLP (*Multiple Layer Perceptron*), o número de camadas intermediárias na rede define o número de pesos existentes e que precisam de valores a serem atribuídos durante o treinamento. Devido a grande quantidade de pesos, esse procedimento de atualização de valores dos pesos deve ser feito de uma forma automática, procedimento conhecido como *learning* (aprendizado).

Segundo (KHAN *et al.*, 2018), uma forma básica para aprendizado é a *Delta Rule* (regra delta). A ideia consiste em a rede aprender com os erros, durante o treinamento, através da diferença entre a saída desejada e a saída obtida. O erro é obtido através de uma função linear LMS (*Least Mean Square*). A Equação (18) define o erro, onde *n* é o número de categorias do *dataset* (ou classes, número de neurônios de saída), *y* o valor esperado e *y'* a predição (valor obtido).

$$E = 1/2 \sum_{n} (y_n - y'_n)^2. \tag{18}$$

A regra delta calcula o gradiente da função de erro, através dos pesos da rede: $\partial \mathrm{E}/\partial\theta_{i,j}$. Os pesos (representados por θ) são atualizados iterativamente através da seguinte regra de aprendizado:

$$\theta_{i,j}^{t+1} = \theta_{i,j}^{t} + \eta \frac{\partial \mathrm{E}}{\partial \theta_{i,j}}. \tag{19}$$

$$\theta_{i,j}^{t+1} = \theta_{i,j}^{t} + \eta (y_i - y_i')x_j. \tag{20}$$

Onde t é a iteração anterior, η é o *hyper-parameter* (hiper-parâmetro definido antes do treinamento) referente ao tamanho do passo para atualização do aprendizado e x as saídas da função linear. Os parâmetros são atualizados até que a predição se aproxime do desejado. Após as iterações, a rede é dita convergida quando o erro permanece constante.

Uma forma básica para cálculo do gradiente da função de erro é uma generalização da regra delta. A regra delta calcula apenas uma combinação linear entre uma entrada e uma saída. Já a generalização da regra delta usa uma função de ativação não linear, e diferente da original, o erro é recursivamente retropropagado pela rede através das camadas, permitindo atualização dos pesos até que o erro seja reduzido progressivamente ao valor mínimo (KHAN *et al.*, 2018).

Dada a Equação (20), o gradiente pode ser calculado para a camada de saída L, em cada nó, através da Equação (21) e Equação (22):

$$\frac{\partial \mathrm{E}}{\partial \theta_{i,j}^{\mathrm{L}}} = \delta_i^{\mathrm{L}} x_j. \tag{21}$$

$$\delta_i^L = (y_i - y_i')f_i'(a_i). \tag{22}$$

Onde $a_i = \sum_j \partial_{i,j} x_j + b_i$ é a ativação de entrada ao neurônio (antes da função de ativação), x_j são as saídas da camada anterior, $y_i' = f(a_i)$ é a saída do neurônio (ou predição se for a camada de saída) e $f(.)$ define uma função não linear de ativação, enquanto que $f'(.)$ a sua derivada.

Para o cálculo do gradiente de erro neste tipo de rede, são utilizadas funções de ativação que resultam em maiores variações. O comportamento das principais funções utilizadas para converter as entradas lineares em não-lineares pode ser visto na Figura 31.

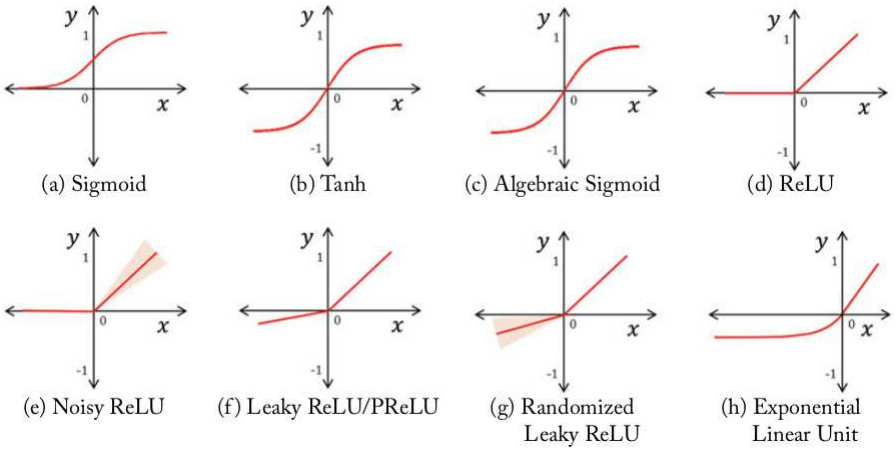
Uma função de ativação popular é a sigmoid, definida pela Equação (23):

$$y_i' = f(a_i) = \frac{1}{1 + exp(-a_i)}. \tag{23}$$

A derivada da sigmoid é dada pela Equação (24):

$$f_i'(a_i) = y_i'(1 - y_i'). \tag{24}$$

Figura 31 – Principais funções de ativação utilizadas em *Deep Learning*.



Fonte: (KHAN *et al.*, 2018)

O cálculo do gradiente das saídas das camadas é definido pela Equação (25):

$$\frac{\partial \mathrm{E}}{\partial \theta_{i,j}^{\mathrm{L}}} = (y_i - y_i')(1 - y_i')x_j y_i'. \tag{25}$$

Com isso, o sinal de erro a ser retropropagado para as camadas intermediárias através da Equação (26):

$$\delta_i^l = f'(a_i^l) \sum_j \theta_{i,j}^{l+1} \delta_j^{l+1}. \tag{26}$$

Onde $l \in 1...L-1$, L o total de camadas. Por fim, para a atualização progressiva dos pesos, a equação Equação (22) é aplicada à todas as camadas conforme a Equação (27):

$$\theta_{i,j}^{t+1} = \theta_{i,j}^{t} + \eta \delta_i^l x_j^{l-1}. \tag{27}$$

Onde x_j^{l-1} como a saída da camada anterior e t o número da iteração do treinamento. Os parâmetros são atualizados continuamente até que os pesos sejam otimizados (até um número limite de iterações ou até que $\theta_{i,j}^{t+1}$ seja constante) (KHAN *et al.*, 2018).

2.2.3.1.1 Redes Neurais Convolucionais (CNN)

As RNA convolucionais (CNN - *Convolutional Neural Network*) foram projetadas para trabalhar em forma de grade, possuindo grande uso na visão computacional. Na CNN, cada camada possui 3 dimensões, com duas para a informação espacial e uma de profundidade, correspondendo ao número de características da camada (*HxWxD*

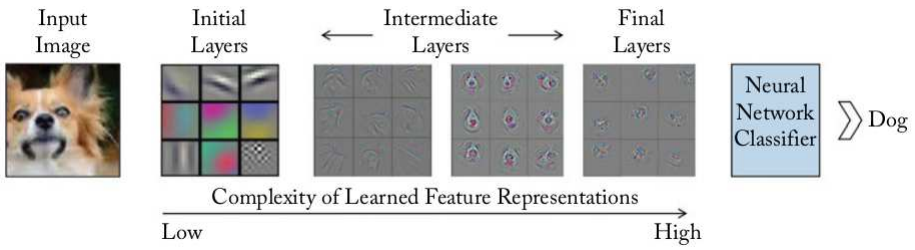
- altura x largura x profundidade). Na camada de entrada, para imagens RGB, a profundidade corresponde ao número de canais da imagem (vermelho, verde, azul). Caso seja uma imagem em escala de cinza, a camada de entrada terá apenas uma profundidade, embora, nas camadas ocultas, a profundidade será o quanto de características forem desejadas. A arquitetura geralmente possui dois tipos de camadas: convolução e subamostragem (*pooling*).

Nas camadas de convolução, um operador é aplicado para extrair características de uma camada para a próxima camada oculta. O operador é um filtro (ou *kernel*) de pesos 3D com a mesma profundidade da camada em questão, mas em geral com dimensões menores (como *3x3xD, 5x5xD, 11x11xD*). O produto dessa operação, após aplicada uma função de ativação, é usado como entrada para a próxima camada.

Devido à convolução e à extração de características locais (através dos operadores), rede é considerada *translation invariance* (invariante de translação), isso significa que um mesmo objeto pode estar em diferentes posições, em diferentes imagens (AGGARWAL, 2018).

A Figura 32 apresenta um esquema básico de CNN, onde as camadas intermediárias capturam diversas características como bordas, direção, formas e, por fim, classificam a imagem. O número de camadas e características desejadas, assim como o tamanho dos filtros, varia de acordo com o problema e com a arquitetura da rede desenvolvida.

Figura 32 – Exemplo básico de uma CNN, com camadas iniciais com mapas de características e camadas finais para classificação.



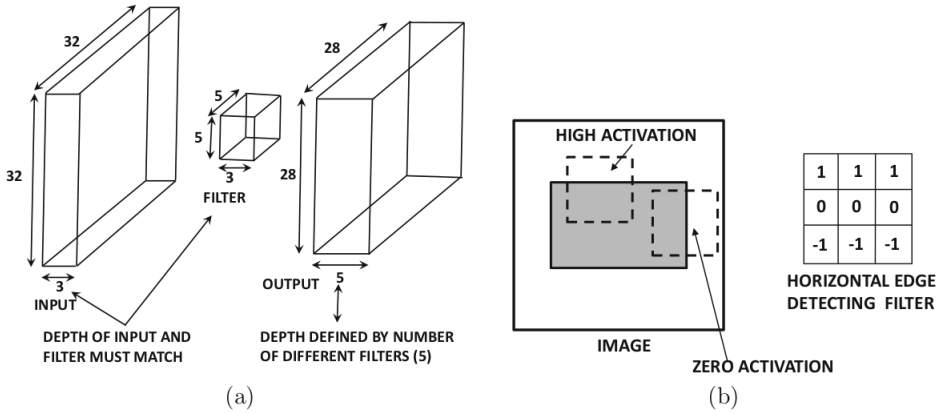
Fonte: (AGGARWAL, 2018)

Convolução

Conforme apresentado, a convolução é uma etapa onde um operador é aplicado para extrair características de uma camada. A Figura 33 (a) apresenta um esquema básico de convolução e filtro. Para a entrada 32x32x3, 5 características são extraídas através de um filtro 5x5x3, resultando em uma camada 28x28x5. Nos filtros, o número de profundidade (*D*) define o número de características resultantes. A figura (b) demonstra um exemplo de filtro para detectar bordas horizontais, que permite, dada uma

imagem, criar uma camada que extrai apenas as características de bordas horizontais desta. Em CNN, os pesos dos filtros normalmente são aprendidos pela rede durante o treinamento supervisionado (contendo GT), através da técnica de retropropagação do erro. Portanto, os pesos podem ser inicializados com um valor fixo, com uma função, ou ainda, recebendo os pesos pré-existentes de outra rede (MICHELUCCI, 2019).

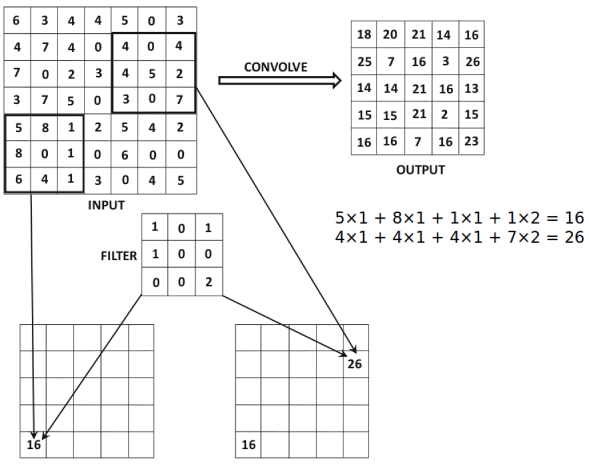
Figura 33 – Arquitetura básica de uma RNA convolucional.



Fonte: (AGGARWAL, 2018)

A aplicação do filtro pode ser compreendida através da Figura 34, com a matriz de entrada simplificada a 2D. O filtro 3x3 deve ser “deslizado” sobre a matriz de entrada 7x7, de modo a multiplicar e somar o filtro com a área em convolução, através das matrizes obtidas. Após a aplicação por toda a matriz de entrada, o resultado é uma matriz 5x5, havendo uma perda de informação devido ao tamanho do filtro aplicado (AGGARWAL, 2018).

Figura 34 – Aplicação de filtro no processo de convolução.

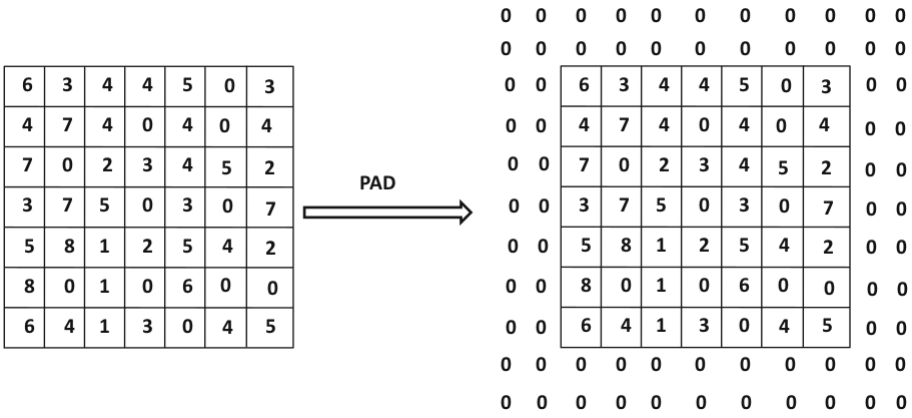


Fonte: Adaptado de (AGGARWAL, 2018)

Preenchimento (*padding*)

A subsequente redução de tamanho causada pela operação de convolução pode não ser desejada, pois tende a perder informações das bordas da imagem. Para tratar isso, a técnica de *padding* pode ser aplicada à imagem, de modo a adicionar um valor constante em torno da imagem. Por exemplo, conforme a Figura 35, em uma imagem 7x7 com filtro 5x5, um *padding* de valor 0 pode ser aplicado, tornando a matriz de entrada 9x9, que após a convolução, resultará novamente em uma matriz 7x7. Quando o *padding* não é utilizado, o resultado da convolução pode ser referida como *valid padding* (como na Figura 34), quando é utilizado de forma a resultar em uma matriz de mesmo tamanho, refere-se como *half-padding*. Pode-se ainda utilizar o *padding* para gerar um resultado com tamanho maior que a matriz de entrada, adicionando mais zeros em torno da matriz do que o filtro necessita, de modo que o filtro inicie a janela de deslize pegando apenas a primeira posição da matriz, sendo este referido como *full-padding* (AGGARWAL, 2018).

Figura 35 – Aplicação de half-padding a uma matriz 7x7 e filtro 5x5.

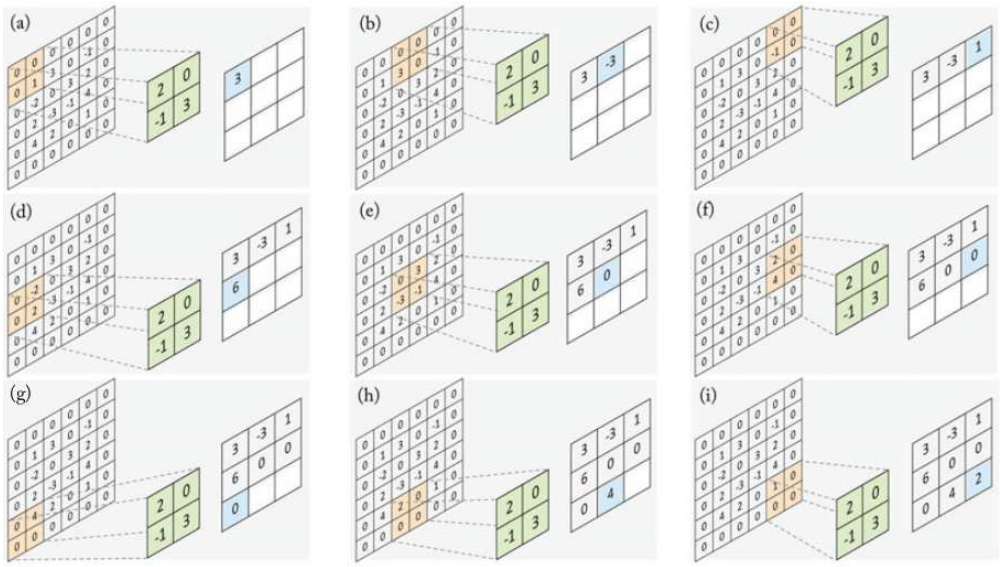


Fonte: Adaptado de (AGGARWAL, 2018)

Passo (*stride*)

A técnica de *stride* é uma outra forma da convolução gerar o mapa de características reduzido. O processo de deslize do filtro, exemplificado na Figura 34, ocorre em toda a matriz de entrada em passo 1, ou seja, a janela é deslizada elemento a elemento até que a janela de deslize atinja a borda da matriz. Esse passo completo indica o uso de *stride* igual a 1. Pode ser desejado aplicar o filtro com uma *stride* maior que um, indicando que o deslize do filtro deve ser feito em maior passo, resultando em uma matriz menor que a de entrada. Esse processo pode ser compreendido na Figura 36, onde dada uma matriz de entrada 6x6 e *stride* igual a 2, a aplicação do filtro resulta em um mapa de característica de tamanho 3x3 (KHAN *et al.*, 2018).

Figura 36 – Passo a passo da aplicação de filtro 3x3 com *stride* igual a 2.



Fonte: (KHAN *et al.*, 2018)

Função de ativação

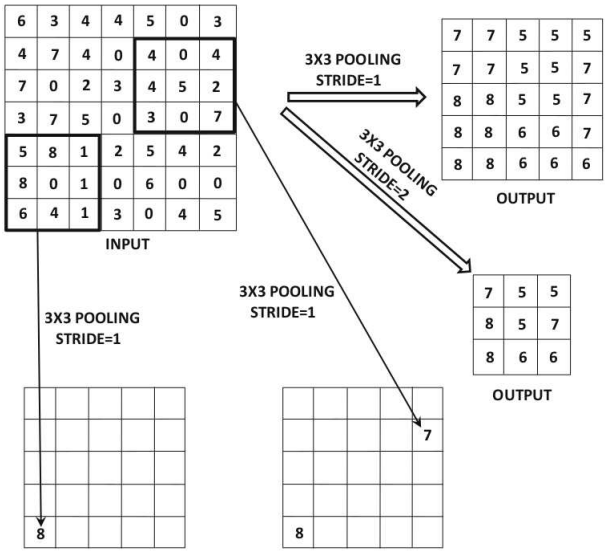
A função ode ativação em CNN semelhante à função de ativação de uma RNA, porém, tem como objetivo gerar um mapa de *thresholds* (limiares) de mesmo tamanho e dimensão da camada em que foi aplicado, normalmente utilizado após a convolução, sendo estes limiares passados para a próxima camada. A função de ativação ReLU é um exemplo de uma função de ativação utilizada (AGGARWAL, 2018).

Amostragem (*pooling*)

A operação de *pooling* tem por objetivo aplicar uma operação na qual produz como resultado, diferente da convolução, uma nova matriz, de mesma profundidade, mas com dimensões diferentes. A operação de *pooling* é realizada através da definição de uma janela (por exemplo 3x3), *stride* e uma função para o cálculo da amostra, resultando um valor a cada aplicação do *pooling*. A Figura 37 demonstra o processo de *pooling* em uma matriz 7x7, com *pooling* 3x3 e com *strides* 1 e 2, utilizando a função *max-pooling*, comumente utilizada, que define como resultado o maior valor de cada deslize da janela de *pooling* (AGGARWAL, 2018).

O *pooling* é útil para gerar mapas de características mais compactos e invariantes a mudanças moderadas de escala de objetos, pose e translação. Existem ainda camadas *unpooling*, ou *upsample*, onde a operação é inversa, ao invés de diminuir a dimensão da entrada, ela é aumentada (KHAN *et al.*, 2018).

Figura 37 – Exemplo da aplicação de *max-pooling* com *stride* 1 e 2.



Fonte: (AGGARWAL, 2018)

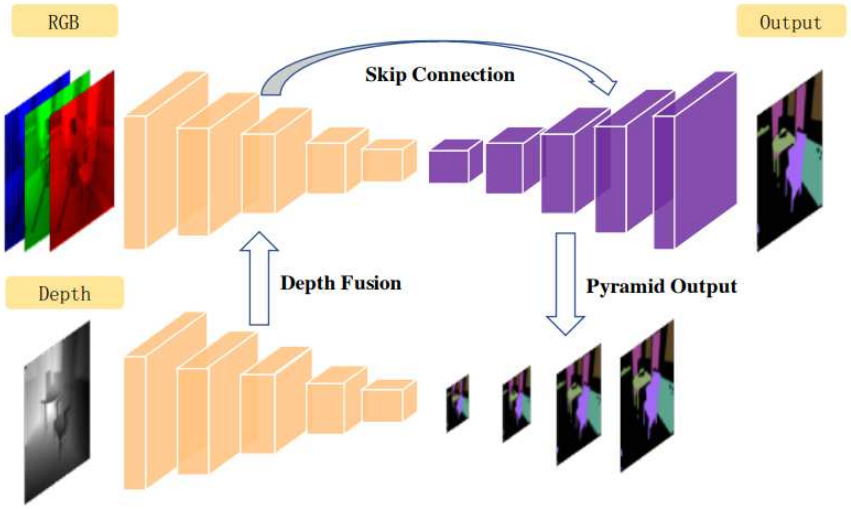
Camadas completamente conectadas (*Fully Connected Layers*)

Na conexão entre camadas, pode-se optar por uma configuração onde todas as saídas da camada são diretamente ligadas a todas as entradas da camada seguinte, de forma semelhante às tradicionais RNAs *feed-foward*, resultando em densas conexões. Essa configuração é utilizada para aumentar o poder computacional da rede. Porém, essa configuração incrementa muito a quantidade de arestas e pesos, o que pode ser um problema, principalmente na aplicação em visão computacional, onde as imagens exigem, por si só, uma grande quantidade de nós e pesos. Por exemplo, se duas camadas possuem 4096 nós (neurônios), serão criados mais de 16 milhões de pesos através de 16 milhões de arestas (AGGARWAL, 2018).

2.2.3.2 Estudo de Caso - RedNet

Dentre os trabalhos selecionados, o RedNet apresenta diferentes camadas e soluções em sua arquitetura, sendo este trabalho interessante para uma análise dos componentes utilizados para compreensão de como os conceitos apresentados são aplicados nas arquiteturas mais recentes de CNN. Conforme explicado pelos autores (JIANG *et al.*, 2018), a Figura 38 apresenta uma visão geral da rede, apresentando em laranja uma primeira parte da arquitetura que realiza a codificação das imagens RGB e D, extraindo o mapa de características, onde a imagem D é codificada isoladamente, em uma cópia da arquitetura do fluxo principal, e constantemente mesclada às camadas da imagem RGB, transformando as predições não lineares em predições píxel a píxel, conforme o limite de classes estabelecido.

Figura 38 – Visão geral da arquitetura RedNet desenvolvida pelos autores.



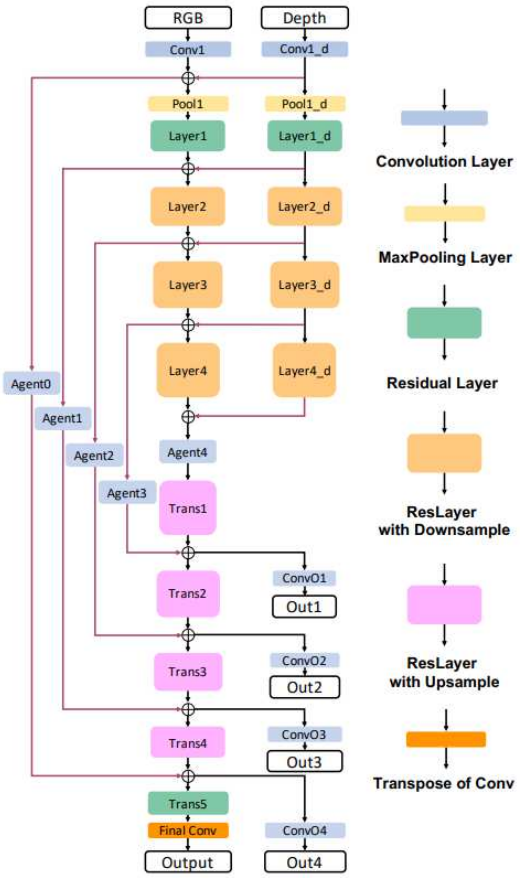
Fonte: (JIANG *et al.*, 2018)

Na Figura 39 a arquitetura é expandida, sendo apresentados os elementos que compõem a rede. A primeira parte de codificação da rede engloba da camada *Conv1* até a camada *Layer4*, onde os mapas de características são obtidos. A camada de entrada RGB inicia com uma camada de profundidade para cada canal de cor, enquanto que o ramo da imagem D apenas uma. Os autores informam que todas as operações de convolução e camadas residuais são seguidas por uma camada de *batch normalization* e pela camada da função de ativação ReLU. A normalização é utilizada para normalizar os dados de entrada dos diferentes mapas de características obtidos, permitindo um processo mais eficiente de treinamento da rede.

A primeira camada *Conv1* e *Conv1_d* realizam uma convolução 7x7 com *stride* 2, seguidas de uma camada *max-pooling* 3x3 com *stride* 2. A camada seguinte é uma camada *Layer1* chamada *residual layer*. Uma camada residual indica um salto de conexão entre camadas (*Skip Layer Connection*), onde os valores de saídas de uma camada são diretamente adicionados às saídas da camada destino. Das camadas *Layer2* a *Layer4* também são *residual layers*, porém aplicam também subamostragem e aumentam o número de mapas de características (profundidade da camada) em fator 2. O salto de conexão das camadas é realizado entre as camadas da etapa de codificação com as camadas da etapa de decodificação. Essa abordagem foi desenvolvida para minimizar o problema de perda de gradiente do erro, observado em arquiteturas profundas (com mais camadas intermediárias com operações de *pooling*), pois são reutilizados os valores previamente calculados da camada origem para a camada destino.

Na arquitetura da imagem D, os seus mapas de características são unificadas ao ramo RGB em cinco camadas, através do método *element-wise summation*, onde

Figura 39 – Apresentação das camadas da rede RedNet.



Fonte: (JIANG *et al.*, 2018)

os elementos das matrizes RGB e D são somados.

A segunda parte de decodificação da rede inicia na camada *Trans1*, onde todas as camadas são residuais com *upsampling* (aumento da resolução) fator 2, menos a *Final Conv*, que realiza a operação de convolução com transposição. Uma convolução de transposição indica uma convolução com aumento da dimensão do mapa de característica, através de um processo de *padding* entre os dados (e não apenas nas bordas, como no *padding* comum) da camada em que a convolução será aplicada.

A rede proposta realiza a etapa de aprendizado através da uma técnica de aprendizado supervisionado em pirâmide. Conforme ilustrado na arquitetura, a rede calcula 4 saídas intermediárias (*Out1...Out4*), além da predição final (*Output*). Todas as saídas possuem resoluções diferentes, conforme a camada em que foi aplicada, sendo a saída *Out1* com 1/16 da resolução da predição (conforme ilustra a Figura 38). Todas as saídas são introduzidas à uma camada *softmax*, utilizando a função *cross-entropy* (entropia cruzada) para o cálculo da função de perda. Para o cálculo do erro destes *Outputs*, os *Outputs* de baixa resolução possuem maior peso sobre o cálculo de erro, com o GT redimensionado conforme a resolução de cada *Output*. Segundo os

autores, essa configuração apresentou melhor eficiência durante o treinamento.

2.2.4 Métricas de segmentação

As métricas fazem parte de qualquer publicação na área de segmentação de objetos, pois facilitam a análise dos resultados, além de possibilitar comparações entre diferentes trabalhos e autores que utilizam as mesmas métricas. Também fornecem dados para os pesquisadores mensurarem a performance, eficácia e fraqueza dos algoritmos. Também podem ser usados como avaliação do desempenho de uma RNA em treinamento.

Entre os algoritmos aplicados à *datasets* que possuem GT, as métricas mais utilizadas pertencem a categoria de avaliações supervisionadas, ou seja, realizam a comparação entre a imagem segmentada e uma imagem GT, fornecendo métricas quantitativas. As métricas ainda podem ser divididas em métricas baseadas em área e métricas baseadas em bordas (LAPLANTE, 2018).

As mais comuns, utilizadas nos trabalhos selecionados neste trabalho, e também utilizadas nos demais trabalhos estado-da-arte em segmentação semântica de objetos, são as métricas baseadas em regiões. Para saber mais sobre outras métricas, há uma excelente relação de métricas em (LAPLANTE, 2018).

Para compreensão das métricas selecionadas, um conceito importante a ser introduzido é a matriz de confusão, que para segmentação de objetos, pode ser estabelecida da seguinte forma, apresentada pela Tabela 1:

Tabela 1 – Matriz de confusão para segmentação de objetos.

		Predição	
		Objeto	Fundo
GT	Objeto	TP	FN
	Fundo	FP	TN

Onde, conforme descrito em (LAPLANTE, 2018):

1. TP (True Positive), definido pela Equação (28), identifica a soma dos píxeis cuja predição foram identificados com a *label* objeto (Positivo) e que de fato são objetos na GT.

2. FP (False Positive), definido pela Equação (29), identifica a soma dos píxeis cuja predição foram identificados com a *label* objeto (Positivo), mas que na verdade percentem a outra *label* (fundo).

3. TN (True Negative), definido pela Equação (30), identifica a soma dos píxeis cuja predição foram identificados com a *label* fundo (Negative) e que de fato pertencem a essa *label* na GT.

4. FN (False Negative), definido pela Equação (31), identifica a soma dos píxeis cuja predição foram identificados com a *label* fundo (Negative), mas que não pertencem a essa *label* na GT.

$$\mathrm{TP} = \sum_{i=1}^{k} n_{(i,i)}. \tag{28}$$

$$\mathrm{FP} = \sum_{j=1}^{k} \sum_{i \neq j}^{k} n_{(i,j)}. \tag{29}$$

$$\mathrm{TN} = \sum_{j=1}^{k} \sum_{i \neq j}^{k} \sum_{j \neq i}^{k} n_{(i,j)}. \tag{30}$$

$$\mathrm{FN} = \sum_{i=1}^{k} \sum_{j \neq i}^{k} n_{(i,j)}. \tag{31}$$

Com base na matriz de confusão, as cinco métricas baseadas em regiões e que foram utilizadas nos experimentos podem ser estabelecidas

2.2.4.1 Acurácia

A acurácia é a relação entre os acertos positivos e negativos sobre o número total de píxeis.

$$\text{Acurácia} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}. \tag{32}$$

2.2.4.2 Precisão

A precisão é a medida entre os acertos positivos da predição sobre o número total de positivos indicados pela predição, ou seja, o número de acertos sobre o número total de píxeis indicados como positivos, indicando o quão precisa é o acerto daquilo que foi indicado como positivo.

$$\text{Precisão} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{33}$$

2.2.4.3 Recall

A revocação/*recall* é a proporção entre os acertos positivos da predição sobre o total de positivos do GT.

$$\text{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{34}$$

2.2.4.4 F1

Também chamado de *F-measure / Dice Coefficient*, oferece um balanço entre a precisão e o *recall*. É uma métrica interessante para obter uma precisão ponderada da predição em relação ao conjunto verdade dos positivos.

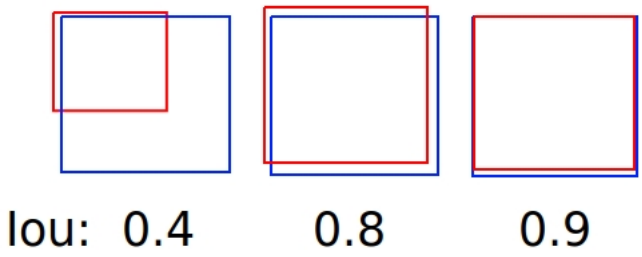
$$\text{F1} = \frac{2 * \text{TP}}{(\text{TP} + \text{FP}) + (\text{TP} + \text{FN})} \text{ ou } \frac{2 * (|S_{pred} \bigcap S_{gt}|)}{|S_{pred}| + |S_{gt}|}. \tag{35}$$

2.2.4.5 IoU

Intersection over Union, também chamado de *Jaccard Index*, é uma relação entre o quanto as predições se sobrepõem ao GT. A Figura 40 apresenta graficamente a sobreposição de uma predição, em vermelho, com o GT em azul. Quanto mais semelhantes são os conjuntos, mais sobrepostos estarão, próximo a 1.

$$\text{IoU} = \frac{|S_{pred} \bigcap S_{gt}|}{|S_{pred} \bigcup S_{gt}|}. \tag{36}$$

Figura 40 – Exemplos visuais da sobreposição de diferentes valores IoU.



3 TRABALHOS RELACIONADOS

Nesta seção são apresentados os trabalhos encontrados na literatura com propostas semelhantes à este trabalho, que visam estabelecer uma comparação entre diferentes abordagens. Também é apresentado em detalhe a metodologia aplicada para a seleção dos trabalhos utilizados nos experimentos, os quais foram desenvolvidos no Capítulo 4 e comparados no Capítulo 5, assim como uma breve descrição destes.

3.1 TRABALHOS COM PROPOSTAS SEMELHANTES

Foi realizada uma pesquisa para encontrar trabalhos semelhantes, que possuem a proposta de comparar abordagens e algoritmos de segmentação em imagens RGB-D. Foram encontrados 6 trabalhos com essa proposta, onde realizam uma seleção de trabalhos, comparados nos *datasets* e métricas publicadas pelos autores de cada trabalho, de modo a realizar uma avaliação das abordagens e métricas disponíveis. Diferente destes, nesta dissertação os resultados dos trabalhos selecionados foram reproduzidos, aplicados a um *dataset* formado, avaliados com as mesmas métricas e discutidos suas aplicabilidades.

Em relação aos trabalhos encontrados, em (GHOSH *et al.*, 2019), os autores fazem uma análise bem completa do impacto das RNAs na tarefa de segmentação de imagens RGB, das áreas de aplicações, dos tipos de arquiteturas de redes neurais, dos *datasets* existentes para a segmentação e dos tipos de segmentação. Em (MINAEE *et al.*, 2020) e (GARCIA-GARCIA *et al.*, 2018), os autores também realizam uma revisão dos tipos de redes neurais para segmentação de imagens, e avaliam mais de 100 trabalhos de RNAs, comparando-os através dos *datasets* RGB, RGB-D e métricas disponibilizadas.

Em (WARD; LAGA; BENNAMOUN, 2019) e (FOOLADGAR; KASAEI, 2019), os autores realizam uma comparação entre abordagens tradicionais e abordagens ML em imagens RGB-D, realizando uma revisão das abordagens, além de comparações entre as métricas disponibilizadas pelos autores. Diante as informações apresentadas, nota-se a dificuldade na realização de comparações entre abordagens diferentes, pois não há um padrão de métricas divulgadas pelos autores, dificultando a análise.

Para trabalhos de detecção de objetos através de BB (*bounding box*) em imagens RGB-D e 3D (pontos de nuvem), os autores (RAHMAN *et al.*, 2019) estabelecem um comparativo de trabalhos recentes em ML em diferentes *datasets*, através dos dados disponibilizados pelos autores dos trabalhos.

3.2 SELEÇÃO DOS TRABALHOS PARA OS EXPERIMENTOS

Para a seleção dos trabalhos utilizados nos experimentos, com base em (KITCHENHAM, 2004) e (NEIVA; SILVA, 2016), os seguintes passos foram aplicados nesta pesquisa:

1. O primeiro passo da revisão sistemática foi definir as questões principais que a pesquisa deseja responder e a sua abrangência. Foi então definida a frase “Quais as melhores técnicas existentes para segmentação de objetos em imagens RGB-D, entre técnicas de aprendizagem supervisionadas e não-supervisionadas, com data de publicação a partir de 2015, aplicadas à robótica móvel?”;

2. O segundo passo foi identificar as palavras-chaves. Foram definidas as seguintes palavras: *rgbd*, *object*, *segmentation, supervised*, *unsupervised*, *video*, *scenes*, *real-time*, *frame*, *neural*, *network*, *robotic*, *mobile*, *autonomous*;

3. O terceiro passo foi criar strings com combinações das palavras-chave definidas no passo anterior. As strings definidas foram essas: "*(rgb-d OR rgbd) AND (object OR foreground OR semantic) AND (segmentation OR detection OR recognition OR extraction) AND (supervised OR unsupervised OR neural OR network OR cnn) AND (image OR images OR video OR scene OR scenes OR frame OR frames OR real-time OR real time) AND (mobile OR autonomous) AND (robotic OR robotics OR robot)*". Essas strings serão informadas aos sites de buscas para encontrar os artigos desejados;

4. O quarto passo foi definir as bases de buscas. Foram utilizadas as seguintes bases:

    a) Scopus (https://www.scopus.com/);
    b) IEEE Explore (https://www.ieeexplore.com.br/);
    c) ScienceDirect (https://www.sciencedirect.com/);
    d) Springer (https://link.springer.com/);
    e) ACM (https://dl.acm.org/); e
    f) Google Scholar (https://scholar.google.com.br/).

5. O quinto passo foi realizar um refinamento na *string* de busca conforme a sua aplicação e visualização dos resultados nas bases de buscas. Para a base Scopus, foi utilizada a seguinte *string*: "*TITLE-ABS-KEY((rgb-d OR rgbd) AND (object OR foreground OR semantic) AND (segmentation OR detection OR recognition OR extraction) AND (supervised OR unsupervised OR neural OR network OR cnn) AND (image OR video OR scene OR frame OR real-time)) AND PUBYEAR > 2014*". Para as demais, foi utilizada a seguinte *string* no mecanismo de busca:

"*(rgb-d OR rgbd) AND (object OR foreground OR semantic) AND (segmentation OR detection OR recognition OR extraction) AND (supervised OR unsupervised OR neural OR network OR cnn) AND (image OR video OR scene OR frame OR real-time OR real time)*".

6. No sexto passo as strings foram aplicadas nos mecanismos de buscas das bases de pesquisa listadas no passo 4. Os resultados de cada base foram baixados em formato CSV e organizados usando a ferramenta (JABREF, 2018). No total, foram obtidos as seguintes quantidades de resultados em cada base:
    a) Scopus: Executado dia 02/07/2018 - 205 resultados obtidos;
    b) Ieeexplore: Executado dia 02/07/2018 - 181 resultados obtidos;
    c) sciencedirect: Executado dia 02/07/2018 - 64 resultados obtidos;
    d) Springer: Executado dia 02/07/2018 - 1084 resultados obtidos;
    e) ACM: Executado dia 02/07/2018 - 72 resultados obtidos; e
    f) Google Scholar: Executado dia 02/07/2018 - 29 resultados obtidos.

    Total importados: 1632 resultados. Removendo artigos duplicados através da ferramenta (JABREF, 2018), 1462 artigos para serem filtrados e analisados.

7. O sétimo passo foi definir os critérios de inclusão e exclusão dos artigos, a fim de reduzir o número de artigos selecionados e filtrar por aqueles de maior interesse. Os critérios de inclusão foram os seguintes:

    - Artigos em inglês; e
    - Relação com o tema.

    Os critérios de exclusão foram os seguintes:

    - Artigos sobre datasets RGB-D; e
    - Artigos fora do assunto.

8. O oitavo passo foi aplicar os critérios de seleção e exclusão nos arquivos. Neste passo, os critérios são aplicados lendo apenas o título e o *abstract* dos artigos. Foi utilizado a ferramenta LibreOffice Calc (THE DOCUMENT FOUNDATION, 2018) para a leitura e anotação dos artigos selecionados. Nesta etapa, sobraram 355 artigos.

9. O nono passo foi aplicar os mesmos critérios de seleção e exclusão nos arquivos, mas incluindo a leitura da introdução e conclusão (opcional), de forma a auxiliar no entendimento do artigo para confirmar se o artigo é de interesse para a pesquisa. Nesta etapa, sobraram 254 artigos.

10. O décimo passo foi a leitura completa dos artigos e realização de um *checklist* de qualidade, ou seja, novos critérios de seleção foram estabelecidos, de forma a criar um *ranking* daqueles de maior interesse para serem aplicados à pesquisa. As métricas definidas são as seguintes: Qualidade (Baixa, Média, Alta, Muito Alta), Performance (Presente, Ausente), Facilidade para implementar (Baixa, Média, Alta), Usa Caffe (Sim, Não) (BERKELEY AI RESEARCH, 2018), Rede Neural (Sim, Não), *Open Source* (Sim, Não), (THE MATHWORKS, 2019) (Sim, Não), Necessita complemento de programação (Sim, Não), Benchmark vs outras técnicas (Superior, Ausente, Nulo). O intuito foi ranquear os trabalhos passíveis de serem utilizados, disponíveis on-line e implementados em linguagem de programação como C++ e Python, que aproximam mais a performance dos resultados a aplicações reais. No caso do Matlab, por exemplo, apesar de ser uma ferramenta mais pesada para a execução dos algoritmos, a maioria dos trabalhos de abordagens tradicionais foram implementados nele, sendo neste caso, listados para avaliação.
11. O décimo primeiro passo foi, para cada artigo bem ranqueado, encontrar o código-fonte da implementação e executar conforme as instruções dos autores, de modo a confirmar a possibilidade de seu uso nesta pesquisa.

Após todas essas etapas, resultaram 115 artigos, dos quais 6 foram obtidos os código-fonte, testados e utilizados nos experimentos.

3.3 ABORDAGENS TRADICIONAIS

A seguir uma breve descrição dos três trabalhos de abordagens tradicionais selecionados, ou seja, que utilizam técnicas tradicionais para a segmentação dos objetos.

RGBD Proposals - Unsupervised object region proposals for RGB-D indoor scenes

Neste trabalho os autores (DENG; TODOROVIC; JAN LATECKI, 2017) estabelecem uma série de características geradas através de cálculos nas informações de imagens RGB, D e 3D através de PCL (Point Cloud Library - Biblioteca de nuvem de pontos), para segmentação não-supervisionada de objetos. A proposta de segmentação é derivada através do cálculo de cinco aspectos: regiões planas, Regiões não planas, planos detectados, planos detectados unificados e agrupamento hierárquico de regiões. O *dataset* Nyuv2 (SILBERMAN *et al.*, 2012), utilizado pelos autores, disponibiliza esse conjunto de informações RGB-D e 3D.

Os autores implementaram um sistema onde o algoritmo propõe tanto a segmentação das imagens, quanto o BB (*bounding box* - caixa delimitadora) dos objetos. A arquitetura é baseada em primeiramente realizar uma estimativa de planos através da nuvem de pontos e identificar as regiões planas das regiões com obstruções (objetos). Após isso, são propostas algumas BB de objetos, utilizando métodos diferentes de GBS (*Graph Based Segmentation*) e WBS (*Watershed Based Segmentation* para detectar regiões planares e não planares. Para as regiões planas sem obstruções, regiões próximas são unificadas com base na proximidade na nuvem de pontos. Para distinguir regiões ambíguas na imagem RGB, a clusterização hierárquica euclidiana é utilizada na nuvem de pontos, uma vez que podem estar ambíguas na imagem RGB, mas bem separadas na nuvem de pontos.

Por fim, uma adaptação do método *GrabCut* é aplicado para gerar a segmentação final. As BB propostas são verificadas, enquanto que os planos iniciais identificados são estimados no restante da imagem. O *GrabCut* foi customizado para trabalhar em 6 dimensões. A imagem RGB foi concatenada à nuvem de pontos (RGB+XYZ). Deste modo, o *GrabCut* é mais sensível a objetos semelhantes nas dimensões RGB, mas distintos na nuvem de pontos.

JCSA-RM - Unsupervised RGB-D image segmentation using joint clustering and region merging

Neste trabalho (HASNAT; ALATA; TREMEAU, 2016) os autores propõem um método não-supervisionado de segmentação baseado em geração de potenciais regiões e agrupamento dessas regiões, através da extração de características das imagens RGB, D, nuvem de pontos e normais de superfície.

O processo do método é dividido em duas etapas: divisão em regiões e unificação de regiões. Com base no modelo RGB, XYZ e N (normais de superfície, de 3 dimensões), estatísticas são extraídas e combinadas, para então serem divididas em regiões através do método de Bregman. A partir das regiões segmentadas, um grafo é estabelecido, de forma a unificar as regiões usando a estratégia de unificação é realizada através da RAG (Region Adjacency Graph), onde são estabelecidas métricas enquanto ocorrem as unificações entre regiões adjacentes, resultando em uma segmentação final da imagem.

Graph Canny Segmentation - Fast Graph-Based Object Segmentation for RGB-D Images

Neste trabalho (TOSCANA; ROSA, 2016) os autores propõem um método não-supervisionado de segmentação de objetos para tarefas de manuseio desses objetos por robôs, tarefas onde robôs precisam identificar o objeto e pegá-lo. Para isto, os autores propõem um extrator de bordas Canny modificado, agregando as informações

RGB e D como uma função de custo. Dessa forma, grafos são obtidos através dessas bordas, gerando regiões de interesse.

Para realizar a segmentação dos objetos, os autores aplicaram uma suavização na imagem D, de forma a tratar os ruídos, também calculam a diferença de cores e mapa de saliências na imagem RGB. Após é aplicado um detector de bordas através do algoritmo Canny modificado que integra a imagem D à imagem RGB. Bordas "falsas" devido a mudança de texturas são removidas conforme as informações da imagem D. Também são filtradas bordas consideradas internas de objetos, visto que só bordas externas dos objetos são de interesse. Duas funções de grafos ponderados são propostas pelos autores, baseadas nos cálculos das imagens RGB, D, diferença de cores e mapa de saliências, aplicadas às bordas de forma a formar grafos, unificando-os por interações, e, então, gerar uma proposta de regiões. Por fim, regiões não entendidas como objetos de interesse são descartadas.

Esse trabalho é diferente dos demais trabalhos incluídos para esses experimentos, pois o algoritmo proposto não tem o objetivo de segmentar toda a imagem, somente possíveis objetos de interesse. Apesar disso, foi adicionado pois foram encontrados poucos trabalhos não-supervisionados para segmentação de objetos. A maioria dos trabalhos recentes em segmentação não-supervisionada, observados durante a pesquisa, tem foco na identificação de objetos para manuseio robótico, ou em áreas especializadas da medicina, astronomia, geologia, agronomia, dentre outras, para identificação de padrões, mas poucos para segmentação.

Porém, em análise ao funcionamento do algoritmo, foi visto que os autores realizam a segmentação completa da imagem, definindo possíveis objetos e regiões planares (por exemplo paredes). Após essa segmentação que são realizadas a pós-segmentação dos autores, onde o que não é objeto é descartado, e objetos de interesse são selecionados. Por esse motivo, esse algoritmo será avaliado pelos resultados da segmentação completa das imagens e também pela segmentação dos objetos de interesse.

3.4 ABORDAGENS APRENDIZADO DE MÁQUINA

A seguir uma breve descrição dos três trabalhos de abordagens aprendizado de máquina selecionados, os quais utilizam redes neurais para a segmentação dos objetos.

FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture

Neste trabalho (HAZIRBAS *et al.*, 2016) os autores propõem um método supervisionado de segmentação de objetos através de redes neurais convolucionais. A

rede proposta extrai separadamente o mapa de características da imagem RGB e da imagem D, em uma etapa de codificação, realiza a fusão dos mapas de características, para então realizar a decodificação dos valores não lineares para chegar na predição final, píxel a píxel. Os autores apresentam duas propostas de fusão, com constante fusão do mapa de características da imagem D à imagem RGB em cada bloco de convolução, e uma a fusão apenas antes dos blocos de *pooling*, mas que não observaram melhoria significativa entre essas duas abordagens. Devido à técnica de fusão, os autores denominaram essa arquitetura desenvolvida como FuseNet.

A rede implementa camadas de convolução (convolução + *batch normalization* + ReLU), *pooling*, *dropout* e *unpooling* sucessivos para então classificar os mapas de características na predição final normalizada. O *dropout* é apenas utilizado durante o treinamento, é uma técnica para aleatoriamente desligar uma porcentagem dos nós existentes em uma camada, de modo a tentar reduzir o *overfitting* (a perda de generalização) da rede.

RedNet: Residual Encoder-Decoder Network for indoor RGB-D Semantic Segmentation

Neste trabalho (JIANG *et al.*, 2018) os autores também propõem um método supervisionado de segmentação de objetos através de redes neurais convolucionais. Semelhante à FuseNet, realiza posterior fusão dos mapas RGB e D. Porém, a rede proposta possui algumas diferenças estruturais, com ligação direta entre unidades residuais da etapa de codificação com a decodificação, além de possuir uma estrutura que aplica um aprendizado supervisionado em pirâmide durante o treinamento, melhorando a performance durante o treinamento da rede. A arquitetura é apresentada em maior detalhe na Seção 2.2.3.2.

FCN - Fully Convolutional Networks for Semantic Segmentation

Uma questão no uso de imagens RGB-D em relação à imagens RGB é avaliar o quão benéfico é agregar a imagem D para a segmentação de objetos. Neste trabalho (LONG; SHELHAMER; DARRELL, 2015), encontrado durante a pesquisa, aborda a segmentação supervisionada através de redes neurais convolucionais, porém utilizando apenas a informação RGB das imagens. Diante dessa questão, foi de interesse adicionar este trabalho para avaliação no *dataset* formado, e por também ser um trabalho de grande referência na literatura.

Porém, devido às diferenças entre a arquitetura FCN e as arquiteturas RedNet e FuseNet, não é possível concluir se a camada D contribui ou não para melhorar o resultado da segmentação. Para que seja possível extrair essas conclusões, experimentos devem ser realizados com algoritmos com a mesma arquitetura, considerando e não considerando a informação adicional D das imagens RGB-D.

O trabalho proposto pelos autores serviu de referência a muitos trabalhos seguintes de redes neurais convolucionais, foi a primeira solução publicada para aplicar uma rede com camadas de convolução em todo o processo de segmentação de objetos. A rede proposta é uma rede neural convolucional, onde a partir da imagem RGB de resolução arbitrária HxWxD, passa por camadas de codificação de mapa de características e decodificação para a predição das classes na resolução de entrada da imagem.

A rede implementa as típicas camadas de convolução e *pooling* para extrair mapas de características da imagem em resoluções menores, para então converter esse mapa de características em uma camada de dimensão 1x1x4096, para então adicionar uma camada 1x1x21 (21 classes do *dataset PASCAL*), e então realizar um *upsampling* desses mapas de características classificados para a resolução original. A proposta ainda adiciona três modelos de rede: FCN-32s, FCN-16s e FCN-8s. A proposta FCN-32s é o fluxo normal da rede, realizando um *upsample* com *stride* 32 ao final, já a FCN-16s aplica uma *stride* 16 combinando a última camada com uma camada de *pooling* anterior, e FCN-8s, combinando a última camada com duas camadas *pooling* anteriores, realizando uma *stride* com 8, permitindo um maior detalhe das predições, com um maior custo computacional.

4 DESENVOLVIMENTO E IMPLEMENTAÇÃO

Foram desenvolvidas as seguintes etapas para permitir a comparação entre as abordagens tradicionais e as abordagens ML na segmentação de objetos em imagens RGB-D:

1. Formação de um *dataset* com cenas em diferentes ângulos de ambientes internos, contendo imagens RGB-D e GT (*ground truth* - conjunto verdade) destas imagens;
2. Adaptação e execução do código-fonte dos algoritmos selecionados para aplicação ao *dataset* formado; e
3. Desenvolvimento das métricas nos resultados obtidos.

A seguir são descritas em detalhes o desenvolvimento de cada etapa.

4.1 FORMAÇÃO DO *DATASET* DE CENAS DE AMBIENTES INTERNOS

4.1.1 Seleção

A criação de um *dataset* com cenas em diferentes ângulos de um ambiente não é uma tarefa trivial. A primeira tentativa de estabelecer esse conjunto de imagens foi captar imagens próprias usando um equipamento Kinect v2 (MORAIS ALONSO *et al.*, 2015). Entretanto, após o manuseio e realização de diversas tentativas, foi observado que o equipamento não foi construído para ser usado com um hardware além do videogame Xbox One, pois apresentou muitos problemas de compatibilidade. Nos dois computadores testados, a conexão USB 3.0 desses equipamentos não foi capaz de realizar o tráfego requerido dos dados enviados do Kinect ao computador, não ocorrendo a captação dos *frames* e a gravação das imagens RGB-D.

Com a impossibilidade da captação própria, uma pesquisa foi realizada com o objetivo de encontrar *datasets* que possuem as características desejadas, de cenas em diferentes ângulos, de ambientes internos, captadas por câmeras como o Kinect v2, contendo GT semântico dos objetos. Alguns *datasets* públicos, com imagens captadas por câmeras mais recentes, como o Kinect v2, foram encontrados, mas poucos passíveis de uso. O grande problema de todos os encontrados foi a falta de GT para segmentação de objetos, muitos *datasets* encontrados possuíam apenas os *bouding box* dos objetos. Dessa forma, não tornando possível o treino dos algoritmos ML nesses *datasets*, e também não permitindo a validação dos resultados dos algoritmos com o GT, de forma a calcular as métricas de acerto e erro das predições.

Dentre os *dataset* encontrados, três apresentaram boas imagens, captadas de ambientes internos, com cenas de alguns ambientes em diferentes ângulos e posições:

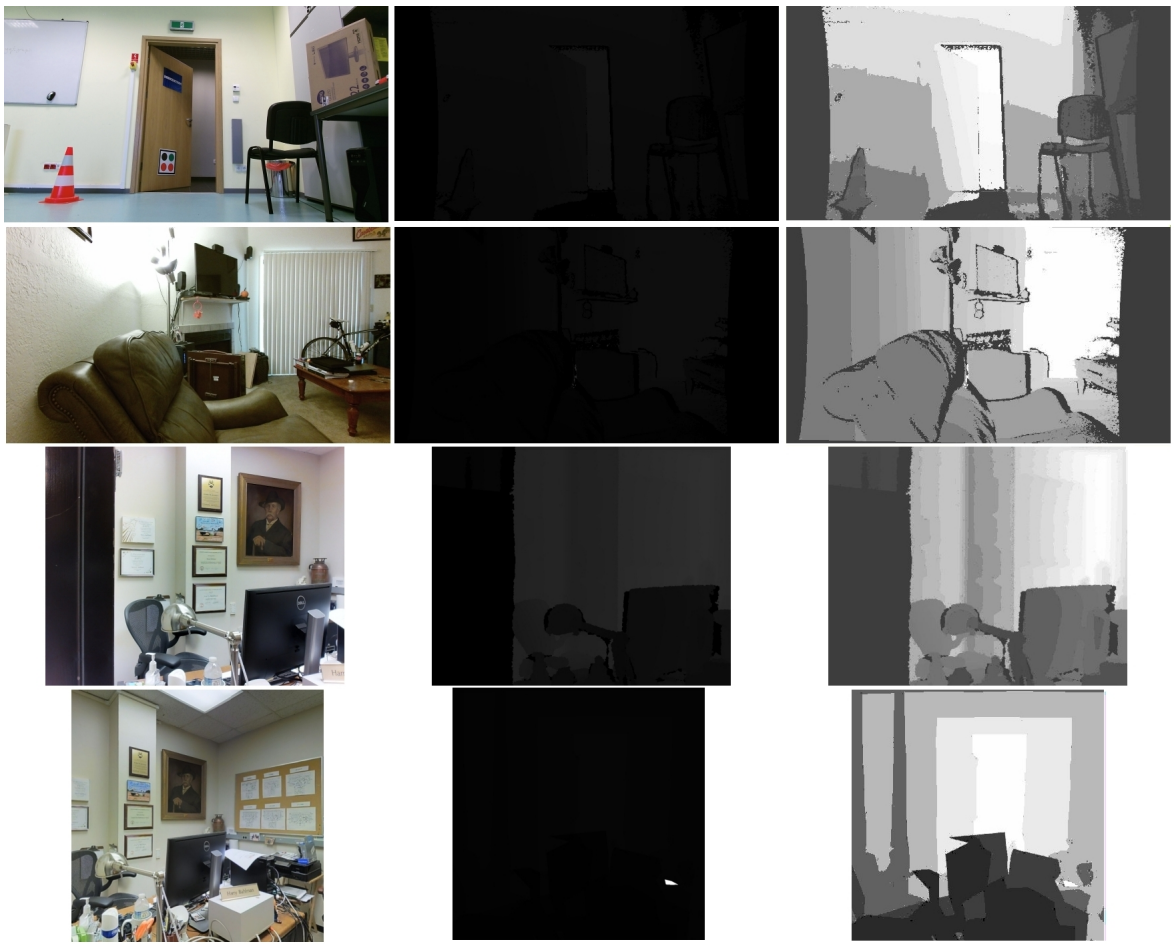
1. PUTKK (KRAFT *et al.*, 2017) - Apresenta imagens RGB-D de uma sala de pesquisa, captadas pelo sensor Kinect v2, acoplada a um robô móvel, movimentando-se pela sala. As imagens RGB e D foram tratadas para ficarem sincronizadas, possuindo a mesma resolução (1920x1080), e aparentemente sem mais nenhum outro tratamento aplicado à elas;

2. Active Vision (AMMIRATO *et al.*, 2017) - Apresenta cenas de diversos ambientes, salas, escritórios, banheiros, cozinhas. A captação das imagens também foi automatizada, utilizando o Kinect v2, porém, com intervalo maior entre os *frames*, ou seja, não são imagens contínuas, mas imagens em diferentes ângulos de diversos ambientes, captadas da mesma forma. Uma desvantagem são as imagens RGB em formato JPG comprimidas. Os autores disponibilizam as imagens originais RGB e D (em resolução 512x424) mediante contato, porém, não foi obtido acesso;

3. Stanford 2D-3D-Semantics (S3DS) (ARMENI *et al.*, 2017) - Enorme *dataset* com cerca de 25 mil imagens RGB-D, captadas de forma semelhante ao Active Vision, porém, com a câmera Matterport, que possui função de rotação para captura 360°, porém com resolução inferior ao Kinect v2. As imagens são disponibilizadas com tratamento de igualdade de resolução entre a imagem RGB e D, e também aplicação de algoritmos para correção da imagem D e eliminação de ruídos, e também imagens sem tratamento (originais).

A Figura 41 apresenta um conjunto RGB-D de imagens de exemplo de cada *dataset*, com uma terceira coluna apresentando a imagem D com histograma equalizado, tornando possível a visualização das diferentes camadas de profundidade da imagem. As duas primeiras linhas são imagens RGB-D na resolução 1920x1080, a terceira em 1280x1024 e a quarta 1080x1080. Dessa forma possível é observar as diferenças entre imagens de câmeras RGB-D, tanto em relação à resolução, quanto ao tratamento e à informação da imagem D, como a suavização das camadas de profundidade na S3DS C/T, existência de pontos cegos nas imagens do Kinect v2 e ruídos aparentes entre as imagens D dos 3 primeiros conjuntos.

Figura 41 – Exemplo de imagens RGB, D e D com histograma equalizado de cada *dataset*. Da primeira à quarta linha, respectivamente, PUTKK, Active Vision, S3DS sem tratamento, S3DS com tratamento.



O *dataset* S3DS informa que possui GT semântico das imagens, o que o tornaria o *dataset* a ser escolhido para os experimentos. Porém, após a análise do GT, foi constatado que as *labels* (classes do conjunto verdade) são inconsistentes, com problemas de *smoothing* (suavização), gerando anotações inconsistentes, e, além disso, com o posicionamento incorreto das máscaras semânticas em relação aos objetos que constam na imagem, tornando-o não adequado para uso.

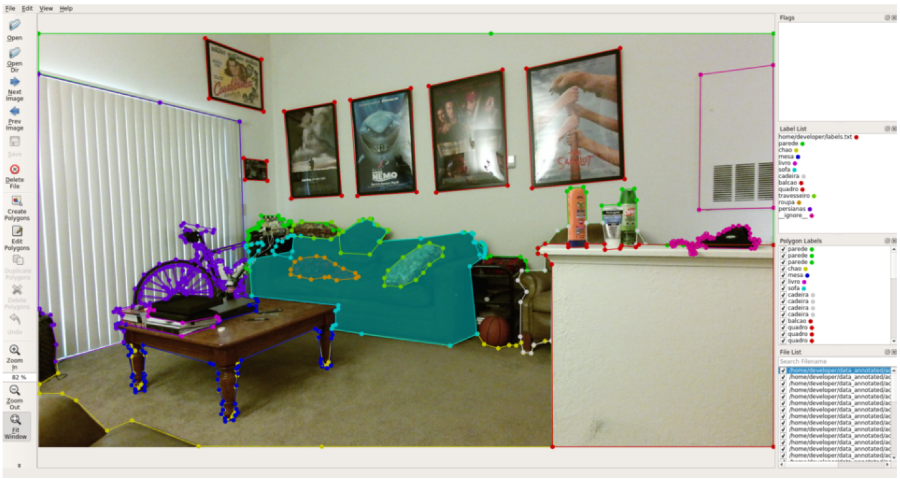
Diante dessas análises e dificuldades na definição do *dataset*, foi decidida a formação de um *dataset*, a partir da seleção de algumas imagens RGB-D de cada um dos três *datasets* de interesse, sendo o S3DS dividido entre conjuntos de imagens com tratamento e sem tratamento. Como não será possível treinar todos os algoritmos ML no *dataset* formado, não há garantia de que as imagens RGB-D estejam nas mesmas condições das imagens SUN-RGBD que foram treinadas pelos algoritmos. Com a mesclagem de imagens de diferentes *datasets*, esse problema poderá ser diluído, e será possível observar se o resultado de imagens em uma determinada condição se sobressaem em relação às outras.

Portanto, a solução para a formação do *dataset* foi selecionar um conjunto de imagens de quatro *datasets* diferentes (PUTKK, Active Vision, S3DS sem tratamento, S3DS com tratamento). Como poderá haver diferença de desempenho dos algoritmos entre esses diferentes conjuntos, as métricas serão coletadas separadamente, possibilitando constatar também se houve impacto nos resultados pela diferença de captação e condição das imagens RGB-D.

4.1.2 Criação do conjunto verdade (GT)

Conforme descrito no processo de criação do *dataset* (SONG; LICHTENBERG; XIAO, 2015), para a criação de GT de segmentação semântica, um especialista deve informar manualmente os objetos contidos em cada imagem, píxel a píxel, de forma a ser atribuída uma *label* (classe) a cada píxel. Dessa forma, conjuntos, ou regiões, de uma mesma classe são formados, representando objetos na imagem. Para auxiliar nesse processo, foi pesquisada e encontrada a ferramenta labelme (WADA, 2016), escrita em Python e que apresentou os recursos necessários para essa criação. Uma imagem da ferramenta em uso pode ser vista na Figura 42.

Figura 42 – *Screenshot* da ferramenta labelme com uma imagem do *dataset* e as *labels* definidas manualmente.



Como os algoritmos ML selecionados possuem treinamento no *dataset* SUN-RGBD (SONG; LICHTENBERG; XIAO, 2015), as imagens foram anotadas com as mesmas classes deste *dataset*, 38 classes de objetos, incluindo uma classe chamada *ignore*, onde é anotada a qualquer outro objeto não pertencente às demais classes. É importante essa anotação das imagens com as mesmas classes e da mesma forma com as quais foram anotadas as imagens do *dataset* SUN-RGBD, pois estes mesmos algoritmos serão validados no *dataset* formado, permitindo o cálculo das métricas com o conjunto verdade.

Para isso, foram estudadas e analisadas as imagens do SUN-RGBD, de forma a entender como os objetos foram anotados, e também entender quais objetos são considerados para cada classe, o que não se demonstrou uma tarefa trivial. Um exemplo é o objeto cômoda, comum em quartos, próximo à uma cama, mas que pode ser classificado também como um armário. Essa inconsistência se apresenta inclusive no próprio *dataset*, e que pode ser explicada pelas diferentes formas com as quais as imagens foram anotadas no SUN-RGBD. As imagens foram captadas por quatro câmeras RGB-D diferentes (Kinect v1 e Kinect v2, Intel RealSense e Asus Xtion), e devido à grande quantidade de imagens, foram contratadas 18 pessoas remotamente, treinadas via Skype, para realizarem a criação do GT de todo o *dataset*. Os autores do SUN-RGBD garantiram a qualidade e inspeção das anotações, mas após a avaliação de algumas imagens, algumas inconsistências podem ser encontradas.

Após conhecidas as classes e a forma de anotar os objetos, o próximo passo foi criar as anotações das imagens RGB selecionadas dos quatro conjuntos definidos. Logo nas primeiras imagens foi possível compreender a complexidade e o quão trabalhosa é a tarefa de criar as anotações semânticas das imagens. Em muitas das imagens foi necessário mais de duas horas para a anotação completa dos objetos, contando o trabalho de revisão das classes e do formato dos objetos. Após algumas horas, o cansaço e o trabalho monótono tornam ainda mais difícil o processo e, principalmente, a manutenção do padrão de qualidade das anotações. Outro problema encontrado, em alguns casos, certos objetos pequenos ou parcialmente exibidos só poderiam ser anotados por quem os tinha em conhecimento da cena real, o que pode acrescentar uma pequena variável de erro a qualquer GT de um *dataset*.

Ao final foram anotadas no total 75 imagens: 40 imagens do *dataset* Active Vision, 9 do *dataset* PUTKK (por possuir imagens de apenas um ambiente), e 26 do *dataset* S3DS, sendo 13 do conjunto sem tratamento e 13 do conjunto com tratamento.

O critério para a seleção destas 75 imagens foi selecionar algumas sequências de imagens de um mesmo ambiente dos *datasets* Active Vision e PUTKK, e algumas imagens de diferentes ambientes do *dataset* S3DS, ambientes diferentes como escritórios, auditórios, corredores, salas de espera, dentre outros. Deste modo, houve uma criação mista de imagens de um ambiente em diferentes ângulos, mas também imagens de ambientes diferentes.

Devido a dificuldade e demora na criação dos GTs, foi necessária essa mescla de imagens em diferentes ângulos e imagens em um único ângulo para tirar proveito dos três *datasets*, além de realizar os experimentos em mais de um ambiente diferente. Por esse motivo, também foram criados apenas 75 GT, devido ao pouco tempo disponível. A Figura 43 apresenta exemplos das imagens selecionadas em diferentes ângulos de um mesmo ambiente, e também imagens de diferentes ambientes.

Figura 43 – Exemplos das imagens selecionadas. Respectivamente por linha: Active Vision, PUTKK e S3DS.

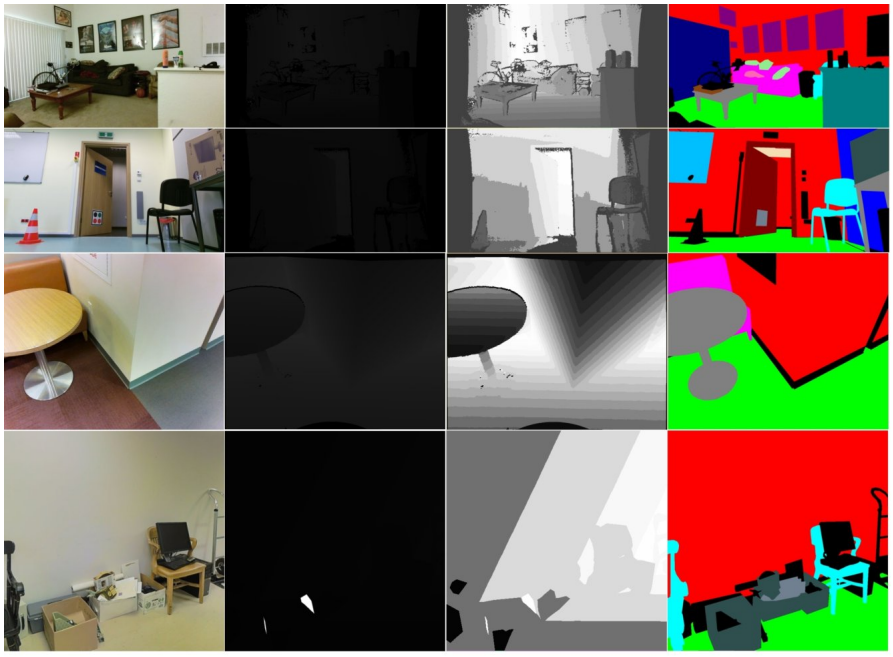
Após realizadas as anotações das imagens, foi necessário escrever um código em Python para especificar o formato de GT desejado. Ou seja, um código para converter as anotações realizadas, que estão em uma determinada estrutura de dados que o programa gera, em imagens com o padrão desejado, no caso, em imagens em escala de cinza contendo as anotações (*labels*) semânticas. A ferramenta disponibiliza outros formatos, como anotações de *bouding box*, ou considerando a instância de cada objeto. A ferramenta fornece exemplos de códigos que geram a imagem no formato de diversos *datasets* públicos e conhecidos, como o Pascal VOC Dataset (EVERINGHAM *et al.*, 2012). Esse código de exemplo foi adaptado para usar as mesmas *labels* do *dataset* SUN-RGBD, e adaptado para ler e gerar separadamente as imagens RGB de cada conjunto definido.

Figura 44 – Imagem gerada pela ferramenta labelme para demonstrar as anotações realizadas em uma imagem RGB. Constam na legenda todas as classes anotadas na respectiva imagem.

Além de gerar imagens coloridas das anotações, o código gera a imagem no

mesmo padrão das imagens GT do SUN-RGBD, onde cada classe recebe valores de 0 a 38, armazenadas no formato PNG em escala de cinza. Apesar de exigir um pouco de codificação, a criação dos GTs através da ferramenta labelme foi satisfatória e de forma simples. O número limitado de imagens anotadas foi devido ao tempo requerido para o trabalho. A Figura 45 apresenta um exemplo das seleções para cada *dataset*, em sequência a imagem RGB, D, D com histograma equalizado (apenas para melhor visualização das camadas de distância) e GT. Para consulta, tanto o código-fonte utilizado, quanto os GT criados, estão disponibilizados em (ZIS, 2019f).

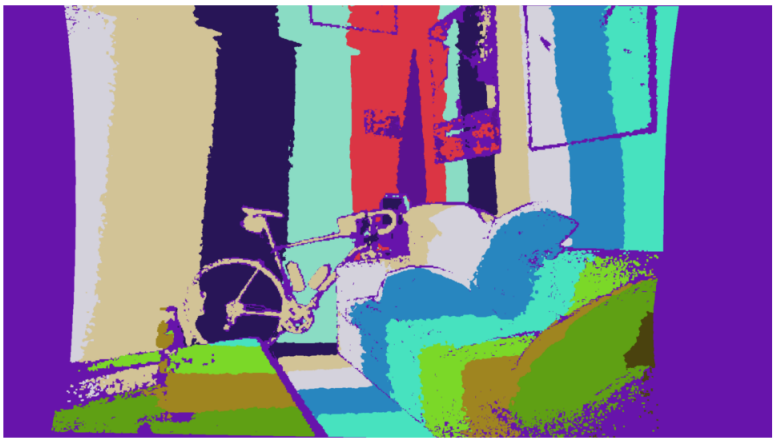
Figura 45 – Amostra de uma imagem RGB, D, D com histograma equalizado e GT dos *datasets* Active Vision, PUTKK, S3DS S/T e S3DS C/T.



4.1.3 Recortes de áreas cegas

Nos conjuntos selecionados dos *datasets* Active Vision e PUTKK, gerados pelo Kinect v2, a imagem D possui pontos cegos do lado esquerdo e direito, devido a diferença de resolução da imagem RGB e D, a câmera capta uma área menor para a imagem D. Esse problema pode ser visualizado na Figura 46, onde para cada valor de cinza foi atribuída uma cor randômica. As faixas roxas da esquerda e da direita indicam o problema. A cor roxa aparece em outros pontos da imagem, mas são indicativos de erro na captura ou ruído. Para evitar predições errôneas dos algoritmos, foi decidido realizar um corte central, de forma a eliminar essas extremidades.

Figura 46 – Imagem D colorida apresentando áreas cegas, em cor roxa, no lado esquerdo e direito.



4.2 EXECUÇÃO DOS ALGORITMOS SELECIONADOS

Após o ranqueamento realizado com os trabalhos encontrados, os melhores ranqueados foram estudados para verificar a possibilidade de seu uso, ou seja, se os resultados podem ser simulados e se o código-fonte está disponível.

Trabalhos específicos para identificação e segmentação de objetos para manipulação robótica. Apesar de interessantes, a aplicabilidade dessas técnicas é específica em relação ao desejado para esta pesquisa, onde pretende-se segmentar objetos, paredes, chão, dentre outros elementos.

Conforme detalhado na seção Seção 4.2.1, a ferramenta Docker (DOCKER, 2018) foi muito útil para criar as diferentes configurações de sistema operacional e bibliotecas para os projetos que permitem a sua execução em ambiente Linux (não incluindo projetos Matlab). O arquivo *Dockerfile* contendo a especificação da máquina e bibliotecas consta em (ZIS, 2019a), com ela, é possível executar todos os 6 trabalhos selecionados.

Em relação à maquina física, os experimentos foram realizados em uma máquina *desktop*, com as seguintes configurações: Processador Intel Core i3-8100 Coffee Lake 8a Geração, Cache 6MB, 3.6GHz; Placa de vídeo NVIDIA GeForce GTX 1050 Ti SC Gaming 4GB; Memória DDR4 16GB 2400Mhz; HD SSD; Sistema operacional Linux Mint.

A seguir serão listados os 6 trabalhos que foram analisados, testados e adaptados para a realização dos experimentos, subdivididos em abordagens tradicionais e abordagens ML.

4.2.1 Abordagens tradicionais

Houve uma grande dificuldade em encontrar abordagens tradicionais entre os trabalhos bem ranqueados da pesquisa. Atualmente, os trabalhos em ML estão dominantes entre as pesquisas de segmentação de objetos. Ainda há uma vasta pesquisa para abordagens tradicionais com segmentação não-supervisionada aplicadas a tarefas mais específicas, como em áreas da medicina, indústrias, reconhecimento de caracteres, dentre outras (GONZALEZ; WOODS, 2011). Áreas onde pode-se definir um escopo e estabelecer uma modelagem para resolver um problema conhecido. Também em aplicações onde não deseja-se ou não seja possível a criação de um *dataset*, requerido em abordagens ML.

No caso de segmentação de objetos, as possibilidades e variáveis são infinitas, tanto na questão da captação da imagem e luz do ambiente, quanto ao próprio ambiente e aos objetos presentes. É difícil um algoritmo desenvolvido para segmentar objetos e cenários previamente identificados, apresentar resultados semelhantes para imagens com configurações completamente diferentes do previsto, com outros objetos e formatos, outros ambientes.

Por esses motivos, os trabalhos de abordagens tradicionais não foram selecionados pensando em performance na sua execução. Muitos trabalhos são experimentais e desenvolvidos no Matlab, o que os tornariam inviáveis para serem aplicados em robótica móvel, por exemplo, sem antes serem reescritos e otimizados em outra linguagem. A seguir, serão apresentados em detalhe dois trabalhos em Matlab, e um trabalho em C++.

RGBD Proposals - Unsupervised object region proposals for RGB-D indoor scenes

Neste trabalho dos autores (DENG; TODOROVIC; JAN LATECKI, 2017), o trabalho foi desenvolvido na ferramenta Matlab. Neste trabalho os autores, além das imagens RGB-D, utilizam também nuvem de pontos, em formato PCL, para o cálculo e proposição das segmentações. Porém, diferente do Nyuv2 *dataset*, utilizado pelos autores, o *dataset* formado e utilizado para os experimentos deste trabalho não possui a nuvem de pontos das imagens.

Após pesquisa, foi descoberto o método de (REN; BO; FOX, 2012) para gerar a nuvem de pontos 3D a partir da imagem D. Porém, o método necessita de informações de parâmetros intrínsecos da câmera: comprimento focal (x, y) e o centro óptico (x, y) (FUSIELLO, 2005). Essas informações devem ser as configurações do momento em que as imagens foram capturadas. Como o *dataset* possui diferentes modelos de câmeras, foram utilizados os dados do Kinect v2 disponibilizados pelo *dataset* PUTKK para os conjuntos Active Vision e PUTKK, e utilizados dados do S3DS S/T para os

conjuntos S3DS, de modo a utilizá-los como uma informação aproximada.

Durante os experimentos, foi encontrado uma chamada a um executável externo, presente no código-fonte, mas que apresentou erro. Após estudo do código, foi identificado que os autores faziam uso do executável para realizar um agrupamento hierárquico euclidiano na informação da nuvem de pontos. Após a chamada, o código-fonte espera, em um arquivo texto criado pelo executável externo, os agrupamentos e os índices da nuvem de pontos pertencentes a cada agrupamento. Após pesquisa, foi encontrada função similar disponível no Matlab. A função *pcsegdist* do Matlab recebe por parâmetro a nuvem de pontos e o valor mínimo de distância euclidiana, e retorna o número de agrupamentos e os pontos pertencentes.

Após esses ajustes no código-fonte, foi possível a execução do trabalho dos autores no *dataset* formado. Mais ajustes foram realizados com o objetivo de realizar a leitura dos conjuntos do *dataset* e também para adicionar a leitura de algumas informações para a formação das métricas, que serão apresentadas no Capítulo 5. O código-fonte completo, com todos os ajustes realizados, pode ser encontrado em (ZIS, 2019h).

Aplicação ao dataset

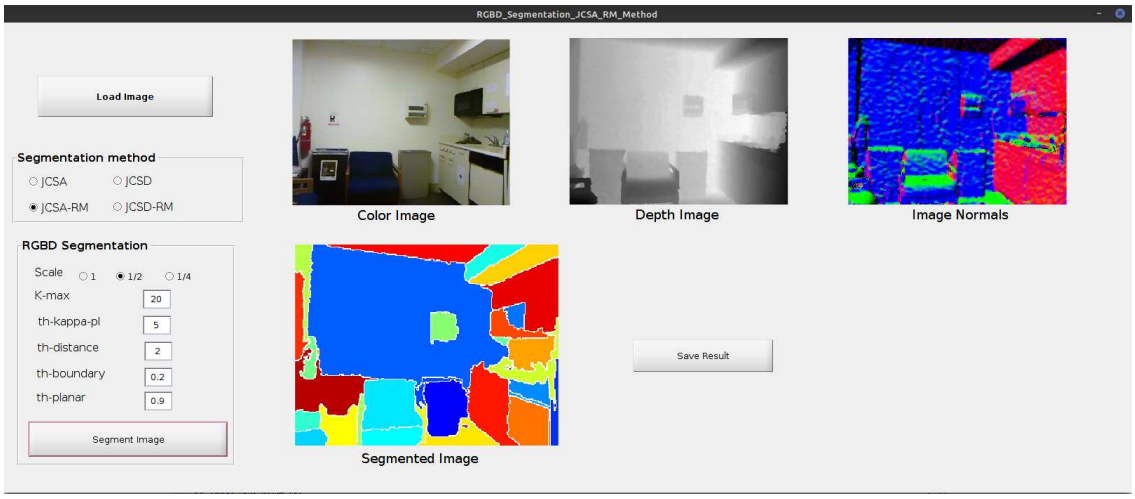
Para este experimento, as imagens do *dataset* foram recortadas da seguinte forma: Active Vision - recorte central de 1440x1080; PUTKK - recorte central de 1440x1080; S3DS C/T original 1080x1080; e S3DS S/T original 1280x1024. Apenas foram removidas as áreas cegas dos conjuntos Active Vision e PUTKK. Ainda durante o processamento, a resolução das imagens foram diminuídas pela metade, visando melhorar a performance dos algoritmos, tornando por exemplo, imagens de 1080x1080 para 540x540. A Figura 47 apresenta uma predição para cada conjunto obtidas por esse algoritmo.

Figura 47 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T.

JCSA-RM - Unsupervised RGB-D image segmentation using joint clustering and region merging

Neste trabalho dos autores (HASNAT; ALATA; TREMEAU, 2016), o código-fonte também foi desenvolvido no Matlab. Neste trabalho são utilizadas as informações RGB, D, nuvem de pontos e os normais de superfície da imagem D, também disponibilizadas pelo *dataset* Nyuv2. A nuvem de pontos foi gerada do mesmo método que para o trabalho RGBD Proposals. Já a geração dos normais de superfície para o *dataset* formado, foi utilizado o mesmo algoritmo ao qual os autores do Nyuv2 *dataset* utilizaram, o algoritmo dos autores (REN; BO; FOX, 2012), onde os normais de superfície são calculados a partir da imagem D. Adicionadas essas informações, a demonstração disponibilizada pelos autores foi facilmente executada. A imagem Figura 48 apresenta a interface desenvolvida pelos autores, contendo o resultado da segmentação de imagens de demonstração incluídas no projeto.

Figura 48 – *Screenshot* com a interface visual desenvolvida pelos autores com as imagens de demonstração disponibilizadas.



O próximo passo foi adaptar o código Matlab para segmentação das imagens do *dataset* formado. Porém, para algumas imagens, foi observada grande demora na segmentação dos objetos. Em alguns casos, levando mais de uma hora para a segmentaçaõ de apenas uma imagem, inviabilizando os experimentos. Após análise, foi observado que os erros ocorriam em laços de repetição que, ou ficavam muito tempo aplicando cálculos e não atingiam o limiar para interrupção do laço, ou o laço era interrompido antes do processamento necessário. Foram realizados alguns ajustes aumentando o número máximo de tentativas e iterações dos laços, permitindo a conclusão do cálculo dos algoritmos e prosseguimento da segmentação.

Além disso, houve alguns problemas referentes a compartilhamento de variáveis utilizados na imagem processada anteriormente que tiveram que ser corrigidos, já que

o algoritmo foi aplicado a todas as imagens do *dataset* sequencialmente. Após todos esses ajustes, foi possível a execução do código no *dataset* formado. O código-fonte completo pode ser encontrado em (ZIS, 2019e).

Aplicação ao *dataset*

Para este experimento, as imagens do *dataset* foram recortadas da mesma forma em que o experimento do RGBD Proposals: Active Vision - recorte central de 1440x1080; PUTKK - recorte central de 1440x1080; S3DS C/T original 1080x1080; e S3DS S/T original 1280x1024. Durante o processamento, a resolução das imagens foram diminuídas em um terço para permitir uma execução mais rápida do algoritmo. Dos algoritmos selecionados, este foi o que apresentou a pior performance em relação ao tempo de processamento. A Figura 49 apresenta os resultados obtidos por esse algoritmo, por conjunto.

Figura 49 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T.

Graph Canny Segmentation - Fast Graph-Based Object Segmentation for RGB-D Images

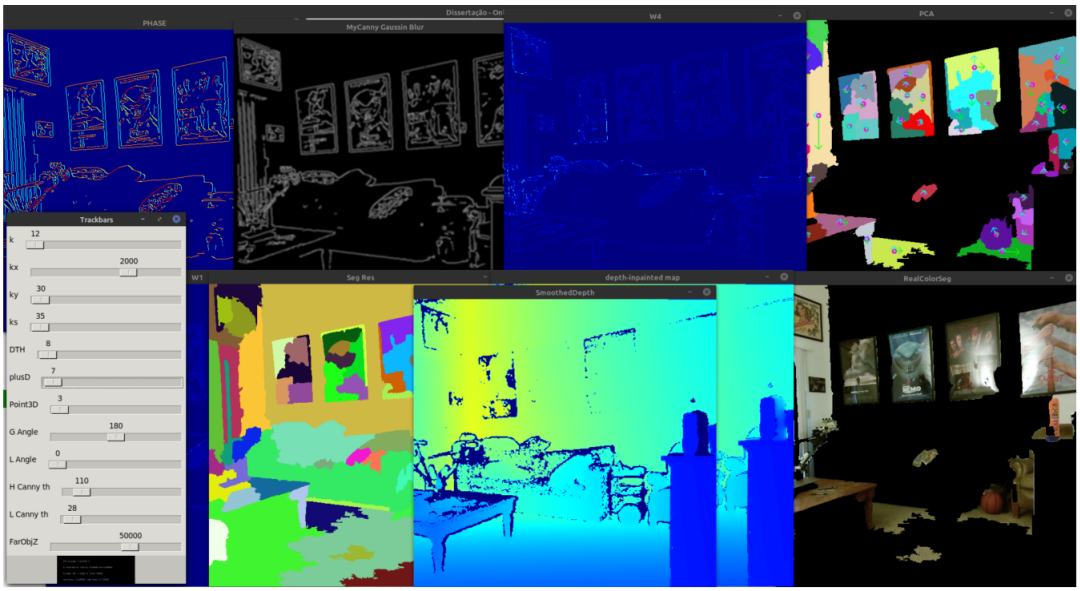
Os autores (TOSCANA; ROSA, 2016) realizaram um grande esforço de implementação usando a linguagem C++ para o desenvolvimento da proposta. Foi criado o ambiente necessário (bibliotecas requeridas) para a execução do código em Docker. O Docker permite ao usuário especificar em código toda a infraestrutura de *software* que deseja, ou seja, especificar o sistema operacional e as bibliotecas que deseja instalar, através de linhas de comando do Docker e também linhas de comando Linux. Deste modo, o Docker virtualiza a máquina desejada, permitindo a sua operação independente da máquina hospedeira.

O código-fonte foi adaptado para servir como uma biblioteca passível de ser chamada através da linguagem Python. Foi adaptado também para ler os parâmetros dos algoritmos em um arquivo texto de propriedades, facilitando os testes e a adaptação dos parâmetros para o *dataset* formado. Também foi adicionada a condição para gravar a imagem com a segmentação completa, antes de serem executadas as operações de pós-segmentação.

Após esses ajustes e desenvolvimento inicial do *framework*, foi decidido diminuir o escopo da pesquisa, já que o *framework* visava oferecer recursos que por si só necessitavam de uma nova pesquisa sistemática, tornando-os inviáveis para o momento. Apesar da ideia ser descontinuada, os experimentos para esse algoritmo foram realizados no *framework* Python desenvolvido, pois seria mais simples o desenvolvimento dos ajustes finais (captação das métricas e monitoramento do processamento) na linguagem Python. Tanto o código C++ adaptado dos autores, quanto o *framework* Python incompleto desenvolvido, estão disponibilizados em (ZIS, 2019b) e (ZIS, 2019a), respectivamente.

A Figura 50 apresenta a interface desenvolvida pelos autores para o algoritmo, ela possibilita a alteração dos parâmetros em tempo de execução, além de permitir a visualização de todas as características geradas pelo algoritmo, em cada etapa, e o resultado final.

Figura 50 – Interface desenvolvida pelos autores, que possibilita testes e ajustes dos parâmetros de segmentação em tempo de execução.



Aplicação ao *dataset*

Neste experimento, as imagens do *dataset* foram recortadas da mesma forma que os algoritmos anteriores: Active Vision - recorte central de 1440x1080; PUTKK - recorte central de 1440x1080; S3DS C/T original 1080x1080; e S3DS S/T original 1280x1024. Durante o processamento, as resoluções também foram diminuídas pela metade. Devido à proposta desse algoritmo ser diferente, os parâmetros originais da pesquisa tiveram que ser ajustados, devido a diferença das imagens providas ao algoritmo. No Apêndice A estão listados os parâmetros alterados e utilizados do algoritmo, ajustados conforme os resultados obtidos durante os experimentos no *dataset*.

A Figura 51 apresenta os resultados da segmentação de objetos de interesse de cada conjunto. Já a Figura 52 apresenta uma figura para cada conjunto dos resultados da segmentação de regiões, antes de realizar a pós-segmentação.

Figura 51 – Predições de objetos de interesse obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T.

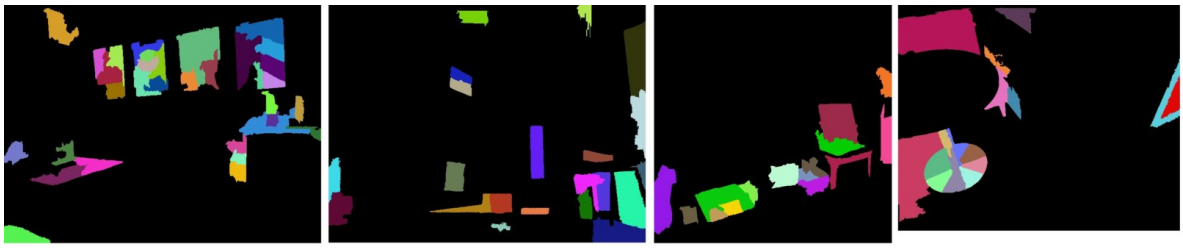
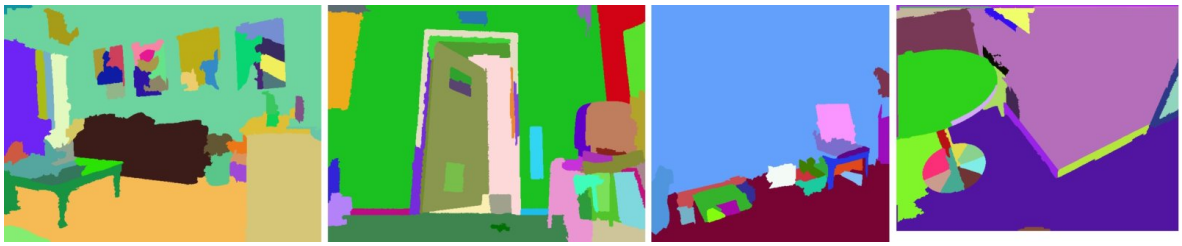


Figura 52 – Predições de segmentação de regiões obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T.



4.2.2 Abordagens Machine Learning

Nos trabalhos bem ranqueados, não foram observados trabalhos com outras abordagens ML, além de RNA. Alguns dos trabalhos encontrados realizam uma mescla entre RNA e SVM para segmentação e classificação dos objetos. Os trabalhos encontrados com data de publicação mais recente, em geral, são mais fáceis para executar, pois a maioria dos trabalhos utilizam Python e construídos utilizando *frameworks*. Uma outra característica encontrada nos trabalhos é, além da segmentação, realizam também a classificação dos objetos, ou seja, informam a que classe de objeto pertencem.

Portanto, foram selecionados três trabalhos para os experimentos, detalhados a seguir.

FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-based CNN Architecture

Neste trabalho (HAZIRBAS *et al.*, 2016), escrito em Python, não houve grandes ajustes. Após estudo do código e do *framework* PyTorch (PYTORCH, 2018), foi possível adaptar o código para o *dataset* formado. Foram adicionadas também a coleta de

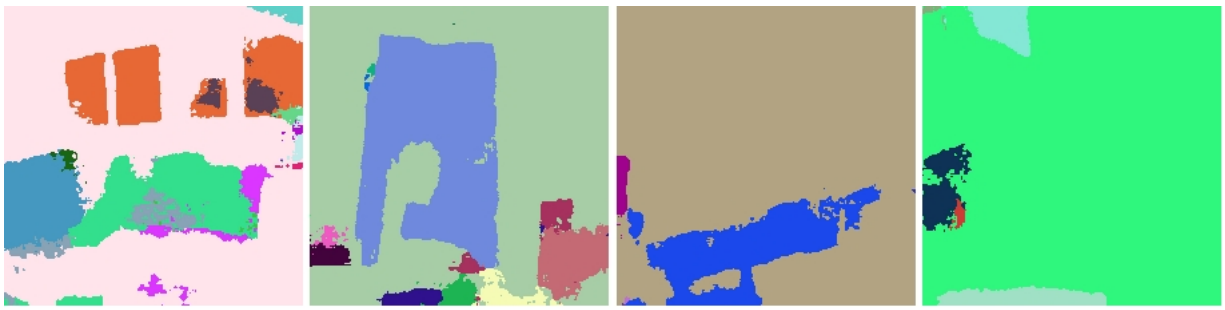
outras informações, como uso da GPU, memória e tempo de predição por imagem. O código-fonte final está disponível em (ZIS, 2019d).

Aplicação ao *dataset*

Neste experimento, como a RNA aceita como entrada imagens na resolução 224x224, as imagens do *dataset* foram recortadas, para possuírem a mesma quantidade de largura e altura, da seguinte forma: Active Vision - recorte central de 1080x1080; PUTKK - recorte central de 1080x1080; S3DS C/T original 1080x1080; e S3DS S/T original 1024x1024. Durante a execução, as resoluções são diminuídas automaticamente pelo código-fonte para a resolução em que a rede neural aceita como entrada, 224x224. O custo computacional das redes neurais convolucionais é alto, e quanto maior a resolução a ser aceita pela rede, maior serão o número de neurônios e parâmetros necessários, exigindo mais poder computacional e memória. Em geral, usa-se resoluções baixas em arquiteturas ML.

Uma outra característica dos trabalhos em ML, por fazer uso de uma biblioteca robusta de ML, todo o processamento é realizado na GPU - placa de vídeo, executando cálculos mais rápidos e em paralelo em relação aos cálculos processados apenas pelo processador. A Figura 53 apresenta os resultados obtidos por esse algoritmo, por conjunto.

Figura 53 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T.



RedNet: Residual Encoder-Decoder Network for indoor RGB-D Semantic Segmentation

Este trabalho dos autores (JIANG *et al.*, 2018) foi o mais fácil de ser executado, dentre os selecionados. Escrito em Python e usando PyTorch, o projeto pode facilmente ser executado. Porém, não foi possível realizar o treino da rede neural, pois durante o treino, a rede exige mais memória GPU do que a máquina tem disponível. Não houve problema para a predição individual de imagens. Como os autores disponibilizaram um arquivo com a rede treinada para o *dataset* SUN-RGBD, foi possível executar os

experimentos. Apenas as informações de tempo de treino não foram obtidas para esta rede neural.

De forma simples, o código foi adaptado para a leitura do *dataset* formado, e adicionadas as métricas, assim como para os trabalhos anteriores. O código-fonte completo está disponível em (ZIS, 2019g).

Aplicação ao *dataset*

Neste experimento, como a RNA aceita como entrada imagens na resolução 640x480, as imagens do *dataset* foram recortadas da seguinte forma: Active Vision - recorte central de 1440x1080; PUTKK - recorte central de 1440x1080; S3DS C/T original 1080x810; e S3DS S/T original 1280x960. Os recortes dos conjuntos S3DS C/T e S/T são apenas na altura, sendo a janela de recorte aplicada de baixo para cima, descartando informações do topo das imagens, que em geral tratam-se de objetos altos, paredes e teto.

Deste modo, são mantidos os objetos de interesse da cena, e não haverá distorção quando as imagens forem redimensionadas para a resolução de entrada da RNA. O processamento é realizado com a GPU. A Figura 54 apresenta uma imagem dos resultados obtidos por esse algoritmo, por conjunto.

Figura 54 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T.

FCN - Fully Convolutional Networks for Semantic Segmentation

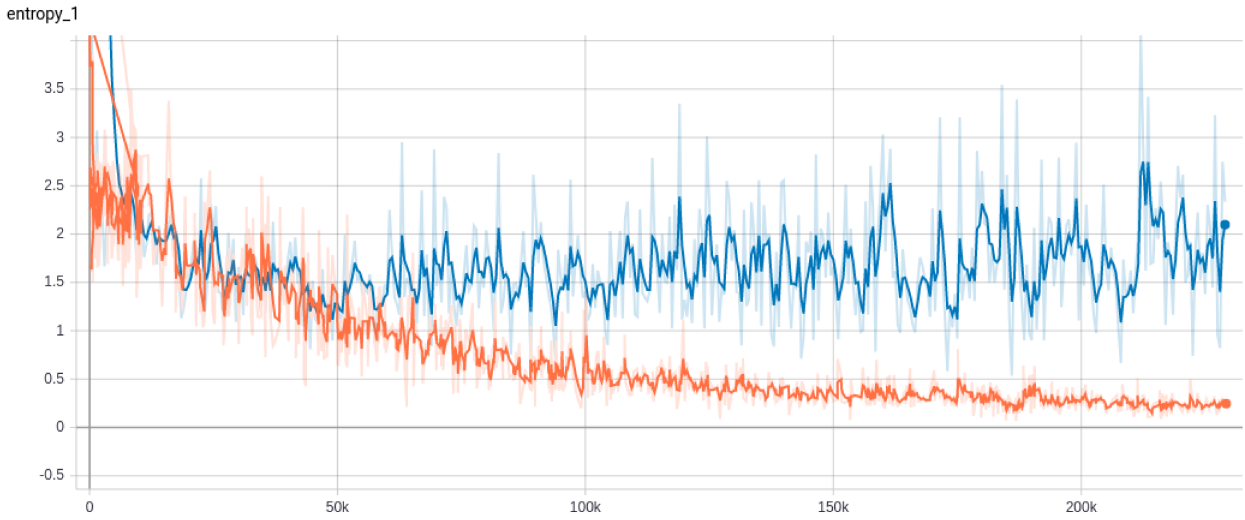
Neste trabalho os autores (LONG; SHELHAMER; DARRELL, 2015) disponibilizaram o código-fonte original, escrito em C++ com o *framework* Caffe, e posteriormente disponibilizaram uma atualização para Python e Tensorflow. Foi utilizado a versão em Python para os experimentos. Neste trabalho, foi necessário adaptar o código e treiná-lo para o *dataset* SUN-RGBD. Foram encontrados poucos códigos-fonte desse tipo de rede FCN no *dataset* SUN-RGBD. Como é desejado obter os resultados através do mesmo processo em que os autores publicaram, partindo com a rede neural com os pesos originais, foi decidido adaptar o código-fonte em Python e treiná-lo no *dataset* SUN-RGBD.

Para o treinamento no *dataset* SUN-RGBD, as imagens de treino e validação foram divididas conforme a sugestão do próprio *dataset*, sendo que, devido a haver muitas imagens com diferentes resoluções, foram implementados recortes randômicos nas imagens RGB e GT antes de serem enviadas à RNA. Esses recortes tem o objetivo gerar 4 possibilidades diferentes de imagens na resolução 448x448 de todas as imagens, cada possibilidade refere-se ao recorte com início nos quatro cantos da imagem. Dessa forma, com os recortes com os mesmos valores de largura e altura, o redimensionamento destas para a resolução de entrada (sugerida pelos autores em 224x224) não ocasionará em distorção dos objetos, além de gerar 4 novas possibilidades de imagens para as 10 mil imagens do *dataset*, gerando 40 mil variações de imagens para treinamento e validação. Resoluções superiores a 224x224 não foram suportadas pela GPU de 4GB de memória.

Este processo de recorte é semelhante ao realizado pelo projeto RedNet, porém, no RedNet ainda são aplicadas variações de escala e rotação da imagem, essas operações são chamadas de *data augmentation* e auxiliam na capacidade de generalização da rede. Também foram adicionados no código-fonte as métricas de acurácia e IoU no sumário de treinamento, para informação dessas métricas durante o treinamento, conforme apresentado na Figura 56.

O treinamento foi realizado até 230 mil iterações e 87 épocas realizadas (87 vezes iteradas o *dataset* de treino), ponto em que foi observado baixo aprendizado, com erro e acurácia constantes.

Figura 55 – Gráfico da função de perda do treinamento no *dataset* SUN-RGBD.

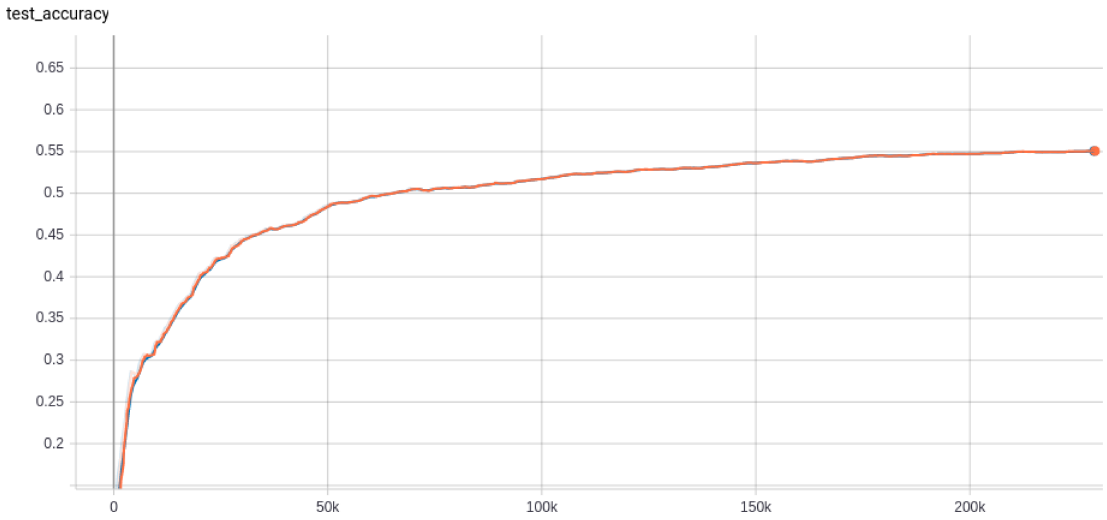


A Figura 55 apresenta o gráfico de perda, gerado pelo cálculo da *Cross Entropy* (entropia cruzada) em relação ao número de iterações. Entre as iterações 200 mil e 230 mil, a perda de treino apresenta valores constantes, em 0,5, já na validação, a

perda é próxima a 2.

A Figura 56 apresenta o gráfico de evolução da acurácia conforme o número de iterações durante o treinamento. Nota-se que entre cerca de 160 mil iterações a 230 mil, a variação da acurácia é entre 0,52 a 0,55. Deste modo, não houve motivo para prolongar o treinamento da RNA.

Figura 56 – Gráfico de acurácia durante o treinamento no *dataset* SUN-RGBD.

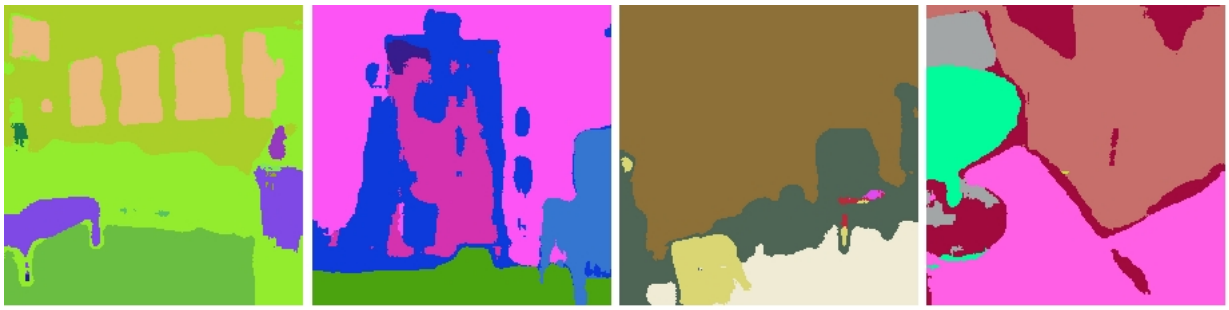


Por fim, com o treinamento finalizado, como nos demais trabalhos, os mesmos monitoramentos de execução foram adicionados ao código. A adaptação completa do código-fonte, adaptado para treinamento no *dataset* SUN-RGBD, consta em (ZIS, 2019c).

Aplicação ao *dataset*

Para este experimento, como a RNA foi treinada com imagens na resolução 224x224, as imagens do *dataset* foram recortadas da seguinte forma: Active Vision - recorte central de 1080x1080; PUTKK - recorte central de 1080x1080; S3DS C/T original 1080x1080; e S3DS S/T original 1024x1024. Durante o processamento, as resoluções são diminuídas para 224x224. O processamento também é realizado com a GPU. A Figura 57 apresenta os resultados obtidos por esse algoritmo, por conjunto.

Figura 57 – Predições obtidas para uma imagem dos *datasets* Active Vision, PUTKK, S3DS C/T e S3DS S/T.



4.3 DESENVOLVIMENTO DAS MÉTRICAS NOS RESULTADOS OBTIDOS

Houve uma grande preocupação em avaliar de modo justo os resultados dos algoritmos, embora possuam suas diferenças. No caso dos algoritmos não-supervisionados, as predições geradas são baseadas em regiões, sem realizar nenhum tipo de classificação sobre essas regiões, não sendo possível, por exemplo, saber qual objeto está contido na região. No caso dos algoritmos ML, as imagens são segmentadas e atribuídas classes a cada conjunto de píxeis definidas, informando a classe daquele objeto.

Portanto, as predições ML geram imagens contendo valores entre 0 e 38 (conforme as classes do SUN-RGBD), em escala de cinza. Já as predições dos algoritmos não-supervisionados, as predições são geradas em RGB, pois as imagens podem conter bem mais regiões do que as 256 variações possíveis que a escala de cinza permite. Os GTs criados manualmente variam também entre 0 e 38 em escala de cinza. Ou seja, não é diretamente possível a comparação entre esses resultados, sendo necessária uma adaptação dos resultados.

Em relação às métricas, foram escolhidas com base nas mais utilizadas nos trabalhos encontrados. Grande parte dos trabalhos fornecem as métricas de Acurácia, Precisão e IoU para a avaliação dos resultados. Outras métricas comuns para análise são a *Recall* e F1. Para a comparação deste trabalho, essas métricas serão aplicadas aos experimentos e disponibilizadas nos resultados.

Para o cálculo dessas métricas será utilizado o código-fonte em Python da biblioteca (SEIF, 2019). Dessa forma, o algoritmo foi amplamente utilizado e testado. A biblioteca utiliza uma outra biblioteca, a (PEDREGOSA *et al.*, 2011), esta muito conhecida e utilizada em Python, no cálculo das métricas Precisão, *Recall* e F1. Além disso, para garantir a precisão do algoritmo, foram comparados os resultados com outros dois algoritmos, não havendo diferença nos resultados.

Nas duas seções seguintes serão detalhadas como as métricas foram aplicadas

para os algoritmos tradicionais e para os algoritmos ML. A primeira seção explanando como as predições dos algoritmos ML foram comparados entre si, e na segunda seção, como essas predições foram adaptadas para a comparação entre todos os algoritmos.

4.3.1 Métricas para segmentação semântica por classe

Uma vez que os algoritmos ML, além de realizarem a segmentação dos objetos, realizam também a classificação dos objetos, foi realizada uma comparação dos resultados entre si, sendo os resultados agrupados pelos quatro conjuntos definidos no *dataset*. Foram atribuídas as métricas Acurácia, Precisão, *Recall*, F1 e IoU.

Para as métricas Precisão, *Recall* e F1, em cada imagem as predições e o GT foram aplicadas considerando o cálculo com média simples para as classes presentes em cada par de imagem (predição e GT), que podem totalizar em até 38 classes. Conforme descrito pelas funções *recall_score*, *precision_score* e *f1_score* de (PEDREGOSA *et al.*, 2011), as métricas são calculadas para cada *label* (classe), sendo calculada uma média simples em todas as classes existentes no par. A métrica Acurácia e a métrica IoU também realizam uma média simples apenas entre as classes existentes em cada imagem predição e GT do par. Deste modo, todas as classes possuem o mesmo peso.

O cálculo final, após a aplicação das métricas em todas as 75 imagens, é realizado também com uma média simples em todas as métricas. São geradas também as métricas de Acurácia por classe, agrupadas em todas as 75 imagens, e ao final, realizada uma média simples pela quantidade de valores encontrados, uma vez que as imagens não possuem todas as 38 classes.

4.3.2 Métricas para segmentação por região

Para permitir a comparação dos resultados de todas as abordagens, foi estabelecida a avaliação dos resultados a nível de instância dos objetos (regiões), assim como optado pelos autores (DENG; TODOROVIC; JAN LATECKI, 2017). Para isso, foi desenvolvido um algoritmo para separar as predições dos algoritmos ML em regiões, através do cálculo da distância euclidiana. Píxeis de uma mesma classe que possuem a distância euclidiana maior que 1 foram separados para uma nova região, de modo a deixar a imagem fragmentada em regiões, semelhante às imagens geradas pelos algoritmos não-supervisionados. O mesmo foi realizado com a imagem GT, de modo a gerar as instâncias dos objetos.

$$de = \sqrt{(x2-x1)^2 + (y2-y1)^2} \tag{37}$$

$$\forall p \subset R, p \in R' \rightarrow de(p,r) \leq 1 | r \in R' \tag{38}$$

A Figura 58 apresenta uma imagem que passou por esse processo de fragmentação das predições de classes em regiões. Os objetos da classe quadro da imagem à esquerda, em cor azul, são classificados como objetos diferentes, de forma a criar diversas regiões únicas na imagem, tornando-a semelhante às imagens geradas pelos algoritmos das abordagens tradicionais. As cores representadas foram geradas aleatoriamente, devido a isso, os elementos como parede e chão estão em cores diferentes, mas continuam representando a mesma região.

Figura 58 – Imagem à esquerda gerada pelos algoritmos ML. À direita, imagem convertida em regiões.

Com as predições dos algoritmos ML convertidas em regiões, o próximo passo foi realizar a mesma segmentação em regiões na imagem GT, gerando uma imagem GT_r segmentada em regiões distintas. Deixando em igualdade todos os resultados das abordagens e também os respectivos GT, permitindo a comparação das segmentações por instância.

Para o cálculo das métricas, deve-se selecionar região a região, da imagem de predição, e comparar com a respectiva região na imagem GT. Foram estabelecidos os seguintes passos para o cálculo de métricas por região:

1. Para cada região R'_{pred} da imagem de predição, é encontrada a região R'_{gt} da imagem GT_r , com base na região que mais combina com a região R'_{pred}. A combinação é feita através de uma contagem das classes (valores RGB) que mais constam nos mesmos índices de R'_{pred} em GT_r, sendo identificadas as respectivas classes LR_{pred} e LR_{gt}. Com a LR_{gt}, R'_{gt} é obtida através de um recorte retangular, de modo a englobar todos os píxeis iguais a LR_{gt}, identificando a região R'_{pred} com o respectivo objeto real;

2. Obtém-se também a classe mais presente na imagem GT original (classe de objetos), conforme posição de R'_{gt};

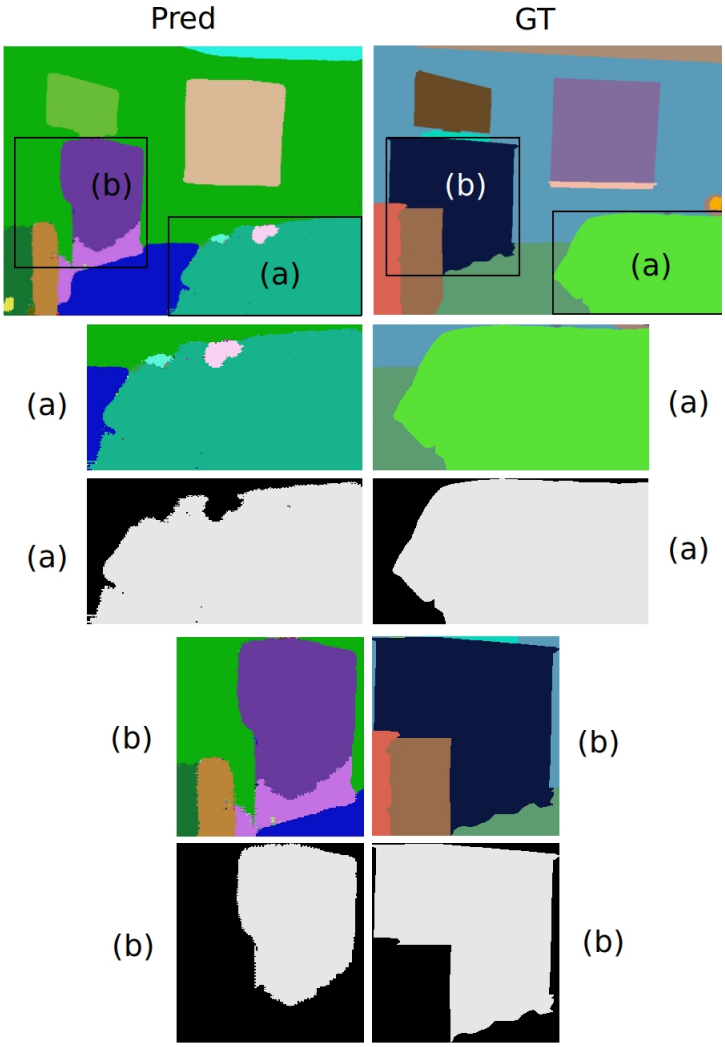
3. Após estabelecida a relação, um recorte é realizado em ambas as regiões, de dimensão $R'_{pred} \cup R'_{gt}$, gerando dois recortes C_{pred} e C_{gt};

4. Os recortes C_{pred} e C_{gt} são binarizados, onde é atribuído 1 aos píxeis iguais a, respectivamente, L_{pred} e L_{gt}, e 0 para os demais, resultando nas imagens B_{pred} e B_{gt}; e

5. As métricas são aplicadas a B_{pred} e B_{gt}, sendo medidas para cada região.

A Figura 59 demonstra visualmente o processo de seleção das regiões a e b, o recorte e a binarização das regiões.

Figura 59 – Exemplificação do processo de recorte de duas regiões. As regiões a e b são recortadas da imagem Pred e comparadas com o respectivo local na imagem GT.



Com o par de recortes binarizados, as métricas Acurácia, Precisão, Recall, F1 e IoU são aplicadas, considerando apenas as medições para a *label* 1 (representada pelo valor 255, em escala de cinza, na figura binarizada). Os resultados são agrupados por classe e por instância da respectiva classe, formando sub-instâncias. Regiões da imagem de predição com a quantidade de píxeis menores que 11 são descartadas

e não prosseguem para o cálculo das métricas. Esse descarte foi necessário, pois as predições das abordagens tradicionais geraram muitas regiões, em alguns casos, até 1300 regiões em uma mesma imagem. Deste modo, esses ruídos podem ser diminuídos, visto que não são suficientes para a representação de objetos possíveis de serem segmentados em imagens deste modo, sendo então descartados.

Outra condição aplicada na seleção de regiões da imagem de predição é para o algoritmo Graph Canny para somente objetos. Como ele segmenta pequenas regiões de interesse, deixando as demais partes da imagem sem valor (0), as regiões de *label* 0 são ignoradas para este caso. De modo a avaliar apenas as regiões onde o algoritmo indica a existência de objetos.

Após obtidas as métricas de todas as regiões propostas da imagem de predição, é obtida a média das métricas. As métricas por imagem são calculadas de duas formas:

1. Conforme descrito, as regiões da imagem de predição são agrupadas por classe e por instância, identificadas na imagem GT, formando uma sub-instância. A média das métricas por instância de uma classe é obtida por uma média ponderada, onde o peso é a quantidade de píxeis pertencentes à sub-instância. A média ponderada é necessária para que grandes sub-instâncias tenham maior relevância na apuração das métricas de uma instância, uma vez que é comum nos algoritmos tradicionais a existência de diversas pequenas regiões. Após a média ponderada por instância, é encontrada a média final da imagem através de uma média simples de todas as métricas separadas para todas as classes e suas instâncias, agrupadas por classe.

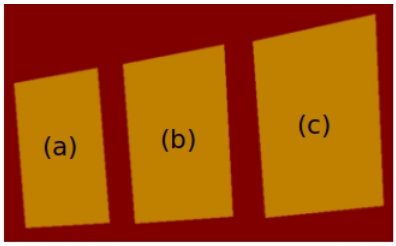
2. Semelhante a forma anterior, porém, ao invés de ser realizada uma média ponderada das sub-instâncias por classe e instância, são selecionadas as métricas da sub-instância com melhor IoU para ser consideradas para a respectiva instância. Dessa forma, a média final é calculada através de uma média simples para todas as melhores métricas de cada instância encontrada das classes.

A Figura 60 demonstra como são consideradas as classes, instâncias e sub-instâncias de uma imagem. A média ponderada indica que, por exemplo, na instância C com três sub-instâncias, as métricas da instância maior (i) terão mais influência que as métricas das sub-instâncias ii e iii. Do mesmo modo, no cálculo através da sub-instância de melhor IoU, as métricas da sub-instância i serão utilizadas para a instância c, desconsiderando as métricas das sub-instâncias i e ii.

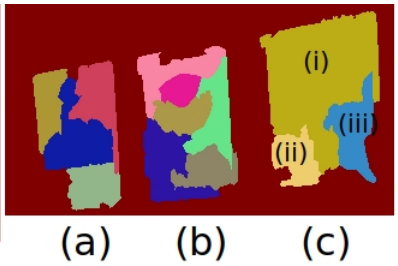
Portanto, são obtidas métricas com médias globais por imagem e também métricas por classe e por imagem. Por fim, as métricas gerais obtidas após o cálculo de todas as imagens do *dataset* são obtidas por média simples. São geradas também as métricas por classe, agrupadas em todas as 75 imagens, e ao final, realizada uma média simples pela quantidade de classes encontradas em cada métrica.

Figura 60 – Exemplo de três instâncias de quadros a,b e c, cada uma com suas respectivas sub-instâncias segmentadas.

Imagem RGB



GT - classe quadro com 3 instâncias a, b e c.



Predição com três instâncias:
a) 4 sub-instâncias
b) 6 sub-instâncias
c) 3 sub-instâncias

5 RESULTADOS

Após a definição do *dataset*, da seleção e execução dos algoritmos e aplicação das métricas, os resultados foram coletados e analisados. Através das métricas Acurácia, Precisão, Recall, F1 e IoU, os resultados podem ser comparados entre si e também com os resultados cujo os algoritmos foram desenvolvidos pelos autores. Também permitem a análise da eficácia dos algoritmos aplicados ao *dataset* formado. Portanto, nesta seção os resultados serão apresentados e discutidos.

Uma amostra dos resultados pode ser visualizada na Figura 61, podemos observar uma grande quantidade de regiões propostas nas imagens dos algoritmos tradicionais. Já os resultados das abordagens ML apresentam algumas falhas e ruídos na predição. As cores em comum nas imagens ML representam as mesmas classes. A cor preta nos resultados do algoritmo FCN indica a classe 0 (desconhecida). Os algoritmos RedNet e FuseNet identificaram a existência das persianas corretamente, já o algoritmo FCN consideraram como cortinas.

A seguir são apresentados e discutidos os resultados quantitativos obtidos dos experimentos.

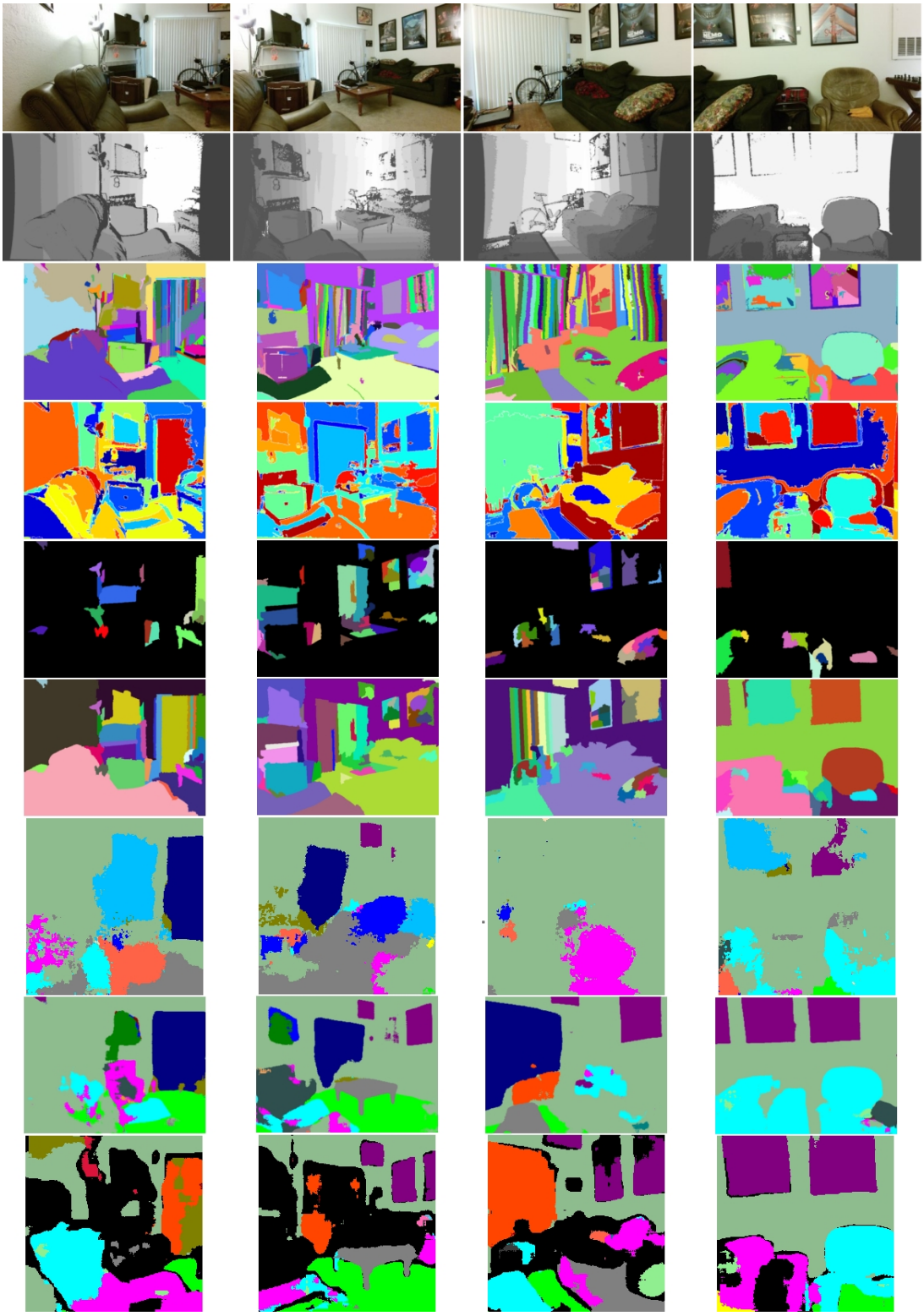
5.1 RESULTADOS DO CONSUMO DE RECURSOS

A Tabela 2 apresenta a performance coletada dos algoritmos. Foram coletadas as seguintes informações dos algoritmos:

- Tempo total de processamento para a predição das 75 imagens;
- Número de execuções realizadas;
- Tempo por imagem para o algoritmo segmentar a imagem;
- Média do uso de memória durante o processamento;
- Média do percentual de uso da CPU durante o processamento;
- Média do número de *threads* abertas durante o processamento;
- Média do percentual de uso da CPU por núcleo;
- Média de uso da memória da GPU; e
- Média do percentual de uso de processamento da GPU.

Para os algoritmos com 10 execuções, os valores foram calculados por média simples, de forma a obter as informações através de mais de uma tentativa, de forma a aproximar os valores do valor médio real. Os algoritmos RGB-D Proposals e JCSA-RM

Figura 61 – Seleção de quatro imagens dos resultados no *dataset* Active Vision. Respectivamente por linha: Imagem RGB, D, RGBD Proposals, JSCA-RM, Graph Canny Obj, Graph Canny Reg, FuseNet, RedNet e FCN.



possuem apenas uma execução devido ao longo tempo exigido para a sua execução. Como foram realizadas diversas execuções durante testes, não foi observada situação

Tabela 2 – Comparação de tempos e custos de execução dos algoritmos tradicionais e algoritmos ML com suporte a GPU

Métricas	**Algoritmos**					
	RGBD Proposals	**JCSA RM**	**Graph Canny**	**FuseNet**	**RedNet**	**FCN**
Tempo total proc. (seg)	17232	25955	177,6	**24,3**	50,1	42,6
Número de execuções	1	1	10	10	10	10
Segundos por imagem	229,76	346,06	1,13	**0,11**	0,21	0,21
Uso de memória (MB)	8.947,67	3.449,26	**2.474,68**	2.668,19	2.577,19	4.785,56
Uso de CPU (%)	75,24	33,97	101,92	**177,29**	176,46	103,17
Threads abertas	-	-	**4**	7	13	27
Uso CPU Núcleo 0 (%)	-	-	5,73	29,45	**41,21**	22,80
Uso CPU Núcleo 1 (%)	-	-	20,96	41,18	**45,08**	24,42
Uso CPU Núcleo 2 (%)	-	-	53,41	**67,08**	57,10	29,24
Uso CPU Núcleo 3 (%)	-	-	23,41	41,76	**42,69**	28,45
Uso de memória GPU (MB)	0	0	0	**1.639,15**	2.674,61	4.152,49
Uso de proc. GPU (%)	0	0	0	43,53	23,92	**74,63**

em que esses algoritmos sejam executados em menos tempo. As colunas com o valor "-" indicam que não foi possível obter a informação, como nos trabalhos em Matlab.

As informações do consumo de recursos de *hardware* apresentadas não representam valores precisos dos algoritmos, pois o monitoramento destas métricas não é realizado de forma constante, e sim captados em certos pontos durante o processamento, de forma a realizar um monitoramento superficial dos recursos e obter uma base de consumo de processamento e recursos.

Na Tabela 2 nota-se o longo tempo em que os algoritmos RGB-D Proposals e JCSA-RM levaram para segmentar 75 imagens. Respectivamente, quase 5 horas e pouco mais de 7 horas. Aqui são identificados um grande problema no uso dessas abordagens tradicionais em segmentação de objetos: o alto grau de recursos computacionais exigidos pelos algoritmos, porém, geralmente implementados de uma maneira pouco eficiente. Embora possível, há uma grande dificuldade em desenvolver algoritmos bem otimizados, em linguagens de programação eficientes, como C++ ou até Python. Porém, fazer isso, e ainda fazer de modo a executar em paralelo utilizando recursos da GPU e dos núcleos da CPU não é uma tarefa trivial, e principalmente, inviável para pesquisadores que, em muitos casos, não possuem o tempo e conhecimento de programação necessário. Existem *frameworks* com implementações de alguns dos algoritmos da literatura, mas nem todos realizam paralelismo apropriadamente, e além disso, muitas vezes os autores customizam os algoritmos para o problema em questão, necessitando de uma implementação própria.

Em contrapartida, o algoritmo Graph Canny rodou bem mais rápido, mas mesmo que tenha a vantagem de ser escrito em C++, a abordagem dos autores é mais simples do que a abordagem dos dois primeiros citados, envolvendo menos informações e

menos algoritmos aplicados.

A performance dos algoritmos ML foram as mais rápidas, sendo capazes de, em cerca de 200 milissegundos, segmentarem uma imagem RGB-D. Essas abordagens devem muito aos bons *frameworks* desenvolvidos, capazes de utilizarem e executarem em paralelo os recursos disponíveis na máquina. Devido a natureza das redes neurais em ser uma arquitetura que pode ser configurada de diferentes maneiras, os *frameworks* foram criados e desenvolvidos para disponibilizar esses recursos computacionais de forma transparente aos usuários.

A abordagem RedNet, apesar de não ter sido possível treiná-la no equipamento utilizado na pesquisa, por requerer mais de 4GB de memória GPU, apresenta resultados equilibrados diante dos outros dois algoritmos ML, e conforme visto adiante, apresentando resultados superiores dentre os algoritmos ML.

Os *frameworks* PyTorch e Tensorflow possuem a funcionalidade de executarem com e sem suporte a GPU, realizando o processamento de forma mista entre a CPU e a GPU, ou apenas com a CPU da máquina. A Tabela 3 apresenta as mesmas métricas para os 3 algoritmos ML, executados sem suporte a GPU.

Tabela 3 – Comparação de tempos e custos de execução dos algoritmos ML sem suporte a GPU

Métricas	**Algoritmos**		
	FuseNet	**RedNet**	**FCN**
Tempo total proc. (seg)	66,9	187,6	**59,4**
Número de execuções	10	10	10
Segundos por imagem	0,42	1,23	**0,37**
Uso de memória (MB)	**1.352,96**	2.907,95	2.490,22
Uso de CPU (%)	**364,18**	346,10	351,79
Threads abertas	5	7	**23**
Uso CPU Núcleo 0 (%)	**92,22**	83,30	87,86
Uso CPU Núcleo 1 (%)	**91,85**	83,09	89,59
Uso CPU Núcleo 2 (%)	**89,62**	87,39	88,63
Uso CPU Núcleo 3 (%)	92,94	**93,71**	88,53

Pode-se observar que, em média, o tempo de predição por imagem aumentou em até 4 vezes. O consumo de memória diminuiu, possivelmente por realizarem *swaps* entre a memória RAM e HD. O consumo de CPU aumentou consideravelmente, utilizando, em média, 88,5% da capacidade total do processador. O *framework* Tensorflow apresentou um resultado um pouco diferente em relação ao PyTorch, o Tensorflow criou mais *threads* durante a sua execução. Porém, nos experimentos sem suporte a GPU, o tempo para predição de uma imagem aumentou apenas 1,76%, tornando-se o algoritmo mais rápido para segmentar uma imagem dentre os três experimentos. A falta de GPU não foi tão impactante para o algoritmo FCN no *framework* Tensorflow.

Conforme discutido, os *frameworks* trazem uma grande vantagem de extrair o máximo dos recursos da máquina de forma transparente. As abordagens tradicionais

utilizaram apenas 17,6% da capacidade total do processador da máquina, 5 vezes menos que os algoritmos ML sem suporte a GPU.

Deste modo, para sistemas autônomos que não possuem uma placa de vídeo disponível, as abordagens ML podem ser utilizadas e podem ainda oferecer resultados interessantes, em um tempo um pouco maior, mas ainda equivalentes à abordagem Graph Canny, por exemplo, escrita em C++.

5.2 RESULTADOS DAS ABORDAGENS ML

Em relação aos resultados quantitativos, obtidos através da aplicação das métricas, a Tabela 4 apresenta os resultados aplicados às predições dos algoritmos ML. É interessante analisá-los separadamente, pois os algoritmos ML selecionados também classificam os píxeis, indicando a que classe de objeto os píxeis pertencem. Os valores em negrito representam o melhor resultado para a coluna, por conjunto.

Tabela 4 – Resultados dos algoritmos ML no *dataset* formado.

Algoritmo	**Conjunto**	**Acurácia**	**Precisão**	**Recall**	**F1**	**IoU**
FuseNet	Active Vision	42,12	14,85	11,88	9,93	12,16
FuseNet	PUTKK	30,02	12,21	9,49	7,52	9,46
FuseNet	S3DS S/T	38,20	13,56	12,21	8,39	8,68
FuseNet	S3DS C/T	34,61	16,96	13,87	9,44	10,75
RedNet	Active Vision	**61,85**	29,32	25,88	24,34	**29,60**
RedNet	PUTKK	55,86	26,98	24,40	22,89	28,22
RedNet	S3DS S/T	**79,55**	**38,80**	**36,26**	**35,16**	**45,09**
RedNet	S3DS C/T	50,42	22,84	19,95	17,22	26,19
FCN	Active Vision	59,28	**32,29**	**28,31**	**25,55**	29,25
FCN	PUTKK	**56,64**	**32,26**	**30,48**	**28,28**	**29,28**
FCN	S3DS S/T	52,69	26,35	23,35	22,01	21,59
FCN	S3DS C/T	**56,66**	**30,78**	**28,01**	**25,11**	**27,49**

A Figura 62 apresenta os resultados da métrica IoU, possibilitando visualmente identificar os algoritmos com melhor resultado. Observa-se os algoritmos RedNet e FCN com valores próximos, com exceção ao conjunto S3DS S/T, onde o algoritmo se destacou. Já o FuseNet apresentou resultados bem abaixo aos demais. É interessante observar que o algoritmo FCN apresentou a melhor precisão e *recall* em 3 dos 4 conjuntos, indicando maior precisão de acerto quando classifica um píxel para uma *label*

Na Tabela 5 são apresentadas os resultados originais dos autores, obtidos no *dataset* SUN-RGBD. O tempo total e erro foram obtidos executando o treinamento dos algoritmos FuseNet e FCN. Observa-se que de fato o algoritmo RedNet manteve o bom desempenho perante os demais, porém, apresentou em média um IoU de 32,28, um resultado 32,5% inferior quando aplicado ao *dataset* formado. O algoritmo FCN, com média de IoU de 26,90, teve um desempenho 34,7% abaixo. Já o algoritmo FuseNet, com média de IoU em 10,26, teve um desempenho 71% abaixo ao resultado original.

Figura 62 – Gráfico com as métricas IoU por algoritmo e conjunto.

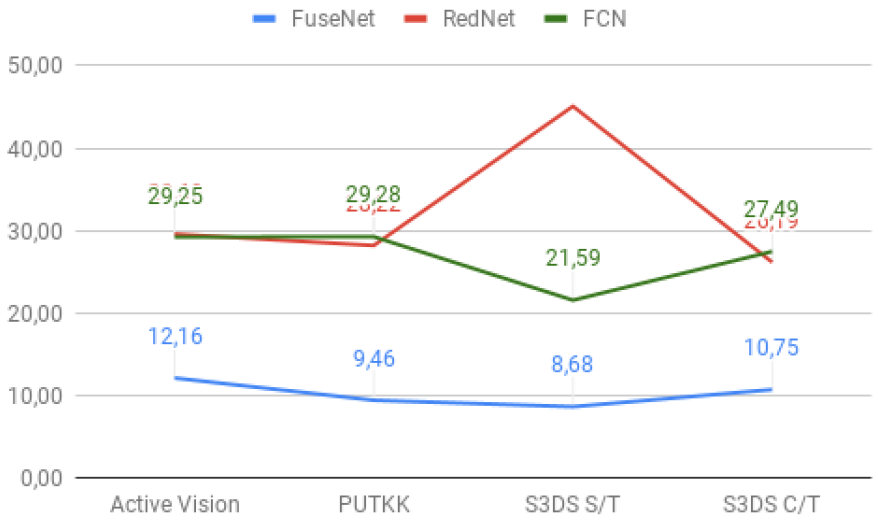


Tabela 5 – Resultados divulgados pelos autores no *dataset* SUN-RGBD

Algoritmo	**Dataset**	**Tempo total**	**Erro**	**Acurácia**	**IoU**
FuseNet	SUN-RGBD	81h16m	2,261	47,00	35,48
RedNet(ResNet-50)	SUN-RGBD	-	-	**60,30**	**47,80**
FCN	SUN-RGBD	56h	2,098	53,90	41,20

É esperado que a aplicação em um *dataset* diferente, com câmeras e captação de imagens diferentes interfira nos resultados originais. Em média, os resultados foram 33% inferiores nos algoritmos FCN e RedNet.

Apenas o algoritmo FuseNet acabou apresentando um desempenho muito abaixo, sem motivo aparente. Analisando a forma de treinamento das abordagens, o FuseNet é o único onde utiliza o *dataset* SUN-RGBD com as imagens já em resolução 224x224, em contraste com os outros dois que leem as imagens originais do *dataset* e realizam recortes e redimensionamentos durante o treinamento. Pode ser que essa forma de treinamento afeta a capacidade de generalização da RNA, resultando em predições piores a imagens de outros *datasets*. Pode ser ainda que o ponto de interrupção do treinamento da rede, sugerido pelos autores, muito longo, o que afetou a generalização da rede. Para validação, foi comparado os resultados com a rede já treinada, disponibilizada pelos autores, mas os resultados acabaram sendo semelhantes.

O desempenho competitivo do algoritmo FCN com o RedNet pode indicar que a diversidade de padrões, tratamentos e ruídos entre as imagens D dos *datasets* podem influenciar negativamente, não contribuindo para a melhoria dos resultado. O *dataset* SUN-RGBD é formado por quatro câmeras diferentes, com diferentes parâmetros

intrínsecos, assim como as imagens selecionadas do *dataset* formado. Visto que a imagem D é bem sensível a todas essas condições, pode-se entender que essas variações de imagens D não contribuem para melhores resultados. Talvez *datasets* formados por apenas um tipo de câmera e tratamento da imagem D contribua para maximizar os resultados de segmentação.

Apesar dessas observações em relação ao uso ou não da imagem D, com o experimento realizado, não pode-se afirmar se a imagem D contribui ou não para melhores resultados, pois as duas técnicas (FCN e RedNet) possuem uma arquitetura de rede neural diferente entre si. Para observações nesse sentido, deve-se realizar experimentos com uso e não uso da imagem D em arquiteturas semelhantes.

Pode-se concluir, com base nestes experimentos, que em um cenário hipotético onde deseja-se aplicar esses algoritmos a um ambiente em que um robô móvel irá interagir, e caso deseja-se extrair o melhor resultado possível destes algoritmos, estes teriam de ser treinados a um novo *dataset* deste ambiente, de forma em que a câmera e suas configurações sejam conhecidas e constantes.

5.3 RESULTADOS DAS ABORDAGENS TRADICIONAIS E ML

Para a comparação dos resultados entre todas as abordagens, foi estabelecido para que o resultado dos algoritmos ML sejam adaptados para formarem regiões, conforme descrito na Seção 4.3.2, assim como o GT.

Em igualdade de condições, as métricas foram coletadas por região. Portanto, nesta seção o objetivo é comparar a qualidade das regiões propostas pelos algoritmos, em relação ao GT, sem considerar a que classe ou objeto pertencem. A Tabela 6 apresenta uma compilação de todas as métricas obtidas para cada algoritmo e conjunto, em negrito os melhores valores obtidos por métrica e conjunto.

Os melhores resultados podem ser visualizados na Figura 63, que apresenta um gráfico de barras para a métrica IoU. Para o conjunto Active Vision, o algoritmo Graph Canny Reg (considerando a segmentação total da imagem) obteve o melhor desempenho, seguido do FCN, RedNet e RGB Proposals. Para o PUTKK, o Graph Canny Reg, seguido do RGB Proposals e, mais abaixo, o algoritmo RedNet. Para o conjunto S3DS S/T, o RedNet apresentou disparado o melhor desempenho, seguido do Graph Canny Reg e JCSA-RM. Por último, o algoritmo FCN obteve o melhor desempenho para o conjunto S3DS C/T, seguido de JCSA-RM e RGBD Proposals.

As abordagens tradicionais e ML apresentaram resultados próximos, dividindo a liderança para cada conjunto. É interessante observar a diferença de desempenho entre os conjuntos, indicando como os algoritmos podem apresentar resultados diferentes em imagens de diferentes câmeras e ambientes.

A Tabela 7 apresenta a média de cada métrica para os algoritmos, com o algoritmo Graph Canny apresentando os melhores resultados, apesar de, dentre as

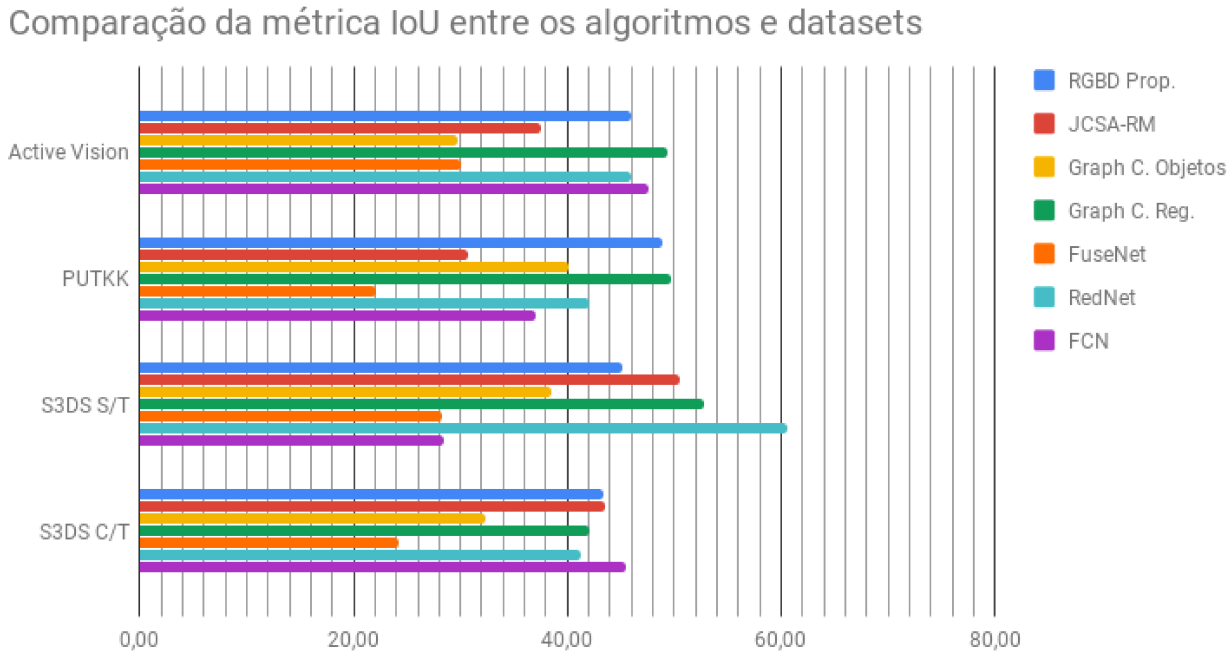
Tabela 6 – Resultados dos algoritmos baseados das predições de regiões, por conjunto.

Algoritmo	Conjunto	Acurácia	Precisão	Recall	F1	IoU	Nr. Obj. GT	Nr. Obj. Pred.
RGBD Proposals	Active Vision	52,17	85,45	52,17	56,27	45,97	17,60	96,13
RGBD Proposals	PUTKK	54,89	**86,92**	54,89	59,74	48,91	39,78	135,00
RGBD Proposals	S3DS S/T	50,97	85,97	50,97	56,53	45,09	16,46	76,31
RGBD Proposals	S3DS C/T	49,48	**82,24**	49,48	54,29	43,39	17,85	79,08
JCSA-RM	Active Vision	43,08	83,94	43,08	48,35	37,44	15,68	139,35
JCSA-RM	PUTKK	36,98	7,96	36,98	41,35	30,67	36,89	171,67
JCSA-RM	S3DS S/T	58,95	84,45	58,95	62,33	50,59	13,46	54,92
JCSA-RM	S3DS C/T	52,44	80,16	52,44	54,94	43,47	16,85	71,92
Graph Canny - Somente Obj.	Active Vision	34,01	**86,30**	34,01	39,11	29,76	8,95	25,03
Graph Canny - Somente Obj.	PUTKK	47,73	81,80	47,73	50,00	40,19	21,44	38,11
Graph Canny - Somente Obj.	S3DS S/T	43,61	**87,40**	43,61	48,67	38,57	10,31	21,69
Graph Canny - Somente Obj.	S3DS C/T	39,25	81,21	39,25	42,80	32,30	8,31	21,00
Graph Canny - Regiões	Active Vision	**57,68**	84,86	57,68	**60,46**	**49,37**	12,03	39,10
Graph Canny - Regiões	PUTKK	**60,08**	79,88	**60,08**	**61,71**	**49,75**	26,22	58,00
Graph Canny - Regiões	S3DS S/T	59,51	86,60	59,51	63,85	52,77	13,23	36,92
Graph Canny - Regiões	S3DS C/T	53,51	77,84	53,51	54,03	42,00	12,15	32,38
FuseNet	Active Vision	41,38	76,61	41,38	40,13	30,02	6,28	**22,00**
FuseNet	PUTKK	35,15	68,02	35,15	31,94	22,16	11,56	31,78
FuseNet	S3DS S/T	39,41	78,03	39,41	38,37	28,27	8,08	**18,08**
FuseNet	S3DS C/T	38,27	74,01	38,27	33,26	24,21	8,38	**21,77**
RedNet	Active Vision	54,60	85,70	54,60	55,30	45,98	10,55	41,83
RedNet	PUTKK	53,14	79,98	53,14	52,91	42,07	18,00	47,67
RedNet	S3DS S/T	**68,87**	87,12	**68,87**	**69,67**	**60,60**	12,08	34,23
RedNet	S3DS C/T	52,41	81,35	52,41	51,04	41,19	10,54	53,46
FCN	Active Vision	57,05	84,69	**57,05**	58,29	47,66	8,10	25,13
FCN	PUTKK	47,51	77,64	47,51	47,93	36,96	14,11	**30,11**
FCN	S3DS S/T	38,22	71,20	38,22	39,95	28,51	8,54	29,62
FCN	S3DS C/T	**55,89**	80,81	**55,89**	**56,93**	**45,42**	9,92	24,46

abordagens tradicionais, ser o mais simples e de ser originalmente criado para segmentar apenas os objetos de interesse. Essa diferença pode ter ocorrido devido às informações adicionais que os algoritmos RGBD Proposals e JCSA-RM exigem e que foram calculados com base em parâmetros intrínsecos genéricos das câmeras, o que pode ter influenciado negativamente os resultados destes.

Alguns autores consideram apenas a melhor região por instância da classe nas comparações. Para tanto, no Apêndice B constam os resultados com base nas regiões

Figura 63 – Gráfico com as métricas IoU por algoritmo e conjunto obtidas das regiões propostas pelos algoritmos.



com melhor IoU, por instância, agrupadas por algoritmo e por conjunto. A Tabela 8 apresenta um resumo desses resultados, agrupando apenas por algoritmo.

Tabela 7 – Resultado médio dos algoritmos baseados das predições de regiões.

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1**	**IoU**	**Nr. Obj. GT**	**Nr. Obj. Pred.**
RGBD Proposals	51,88	**85,15**	51,88	56,71	45,84	22,92	96,63
JCSA-RM	47,86	81,63	47,86	51,74	40,54	20,72	109,47
Graph Canny - Somente Obj.	41,15	84,18	41,15	45,15	35,21	12,25	26,46
Graph Canny - Regiões	**57,70**	82,29	**57,70**	**60,01**	**48,47**	15,91	41,60
FuseNet	38,55	74,17	38,55	35,93	26,17	8,57	**23,41**
RedNet	57,25	83,54	57,25	57,23	47,46	12,79	44,30
FCN	49,67	78,59	49,67	50,78	39,64	10,17	27,33

A escolha pela melhor região beneficiou mais os resultados das abordagens tradicionais, aumentando o IoU em até 14%, no caso do RGBD Proposals. Isso é explicado pela coluna de número médio de objetos das imagem de predição (Nr. Obj. Pred.) da tabela Tabela 6. Os resultados dos algoritmos das abordagens tradicionais possuem, em média, 68,54 regiões, 58% a mais de regiões em relação aos algoritmos ML. Isso indica a predição de mais de uma região para um mesmo objeto.

As abordagens tradicionais produziram resultados interessantes. Entretanto, apresentaram resultados mais sensíveis, muitas vezes segmentando um objeto em

Tabela 8 – Resultado médio dos algoritmos baseados das predições de regiões com melhor IoU de cada instância das classes.

Algoritmo	**Acurácia**	**Precisão**	**Recall**	**F1**	**IoU**
RGBD Proposals	58,46	**87,05**	58,46	64,55	52,24
JCSA-RM	52,82	83,99	52,82	57,77	45,36
Graph Canny - Somente Obj.	43,20	85,54	43,20	47,86	37,20
Graph Canny - Regiões	**62,77**	83,32	**62,77**	**65,73**	**53,20**
FuseNet	40,24	74,36	40,24	37,87	27,57
RedNet	59,33	83,93	59,33	59,79	49,48
FCN	52,78	79,31	52,78	54,25	42,43

diversas partes, conforme pode ser visualizado na Figura 64. Os resultados não foram inferiores, por conta dessas mini-regiões em excesso, pois as métricas, quando obtidas por instâncias pertencentes a uma região da imagem GT, são obtidas através de média ponderada pela quantidade de píxeis da sub-instância, ou seja, o impacto dessas regiões menores é menor se existir uma região maior, que irá puxar a média. Porém, a média ponderada é necessária para que uma predição de uma região pequena não interfira de forma igualitária uma região maior.

Figura 64 – Comparação das imagens, respectivamente: RGB, predição do algoritmo RGBD Proposals e predição do algoritmo RedNet.

O fato de gerarem diversas regiões menores causaram impacto positivo nas métricas de precisão destes algoritmos tradicionais, pois essas regiões menores tendem a ter uma maior taxa de acerto quando comparadas à sua respectiva instância no GT, uma vez que a precisão é obtida considerando apenas os conjuntos TP e FP.

Além do excesso de regiões, a performance menor, e, no caso das abordagens RGBD Proposals e JCSA-RM, exigirem informações adicionais para serem calculadas em tempo real (na aplicação de movimentação robótica, por exemplo), diminuem a possibilidade de uso dessas abordagens.

5.4 CONSIDERAÇÕES FINAIS

Após a avaliação dos experimentos, podemos concluir que os algoritmos ML são mais adequados para a segmentação de objetos, pois são capazes de entregar uma segmentação interessante, com boa performance, e com o adicional de classificarem os objetos, em contraste com as abordagens tradicionais, que não realizam essa etapa no mesmo algoritmo.

O bom desempenho da segmentação total da imagem do algoritmo Graph Canny, tanto em performance, quanto nos resultados, sugere uma possibilidade de um uso combinado dessas abordagens. Uma opção viável para uma solução de movimentação robótica é realizar uma segmentação geral da imagem com os algoritmos ML, e quando houver necessidade de ter um entendimento mais preciso de um objeto ou parte da imagem, ser aplicado um algoritmo de abordagem tradicional para segmentar novamente o objeto em questão, de modo a obter uma segmentação mais sensível em relação à forma do objeto.

Uma outra possibilidade é o desenvolvimento de RNAs adicionais para serem aplicadas apenas nos refinamentos. Porém, as abordagens tradicionais, para a segmentação de objetos, se mostraram interessantes quando deseja-se uma segmentação mais sensível aos contrastes e bordas dos objetos.

Devido a complexidade do problema, as soluções devem ser estudadas e propostas, não havendo ainda uma solução capaz de atender cenários diversos. A necessidade é quem dita a solução a ser desenvolvida. Porém, com esta pesquisa, espera-se que os dados e conclusões apresentadas contribuam para o direcionamento e busca dessas soluções.

6 CONCLUSÃO

Nesta pesquisa foram elaboradas uma revisão bibliográfica sobre os métodos existentes para a segmentação de objetos através de abordagens tradicionais e ML (Machine Learning - aprendizado de máquina). Foi realizado um levantamento sobre os principais trabalhos estado-da-arte sobre o tema, sendo analisados sob a perspectiva de aplicação, especificamente para o auxílio de sua compreensão dos objetos em cenas de ambientes internos, de modo a avaliar o uso dessas técnicas em ambientes reais.

O primeiro desafio em criar ou encontrar o *dataset* de cenas de ambientes internos nos mostrou a dificuldade em, primeiro: uso e captação própria de imagens RGB-D com a câmera Kinect v2; segundo: tendo sucesso na captação das imagens, a dificuldade em criar o conjunto verdade dessas imagens, de modo a possibilitar treinamentos e validações por parte dos algoritmos ML para o ambiente desejado; e terceiro: a dificuldade para encontrar *datasets* públicos de imagens RGB-D, por serem poucos os disponíveis, podem não atender aos requisitos desejados.

Embora, em um primeiro momento, coloque-se em dúvida o custo-benefício e da aplicabilidade das câmeras RGB-D, devido a primeira má impressão na tentativa de uso da câmera Kinect v2, atualmente a própria Microsoft continua investindo na tecnologia, de forma a torná-la atrativa para o seu uso no meio acadêmico e também pela indústria. Anunciado ao final de 2019, o Kinect Azure (MICROSOFT AZURE, 2020) é uma resposta da Microsoft ao amplo desejo demonstrado em suas câmeras predecessoras, que antes criadas para o entretenimento, agora para uso geral e independente. Celulares recentes estão surgindo com a tecnologia TF para cálculo de profundidade, o que poderá contribuir para a evolução da tecnologia, além de possibilitar a popularização de imagens RGB-D.

Cita-se a câmera Kinect v2 por ser a câmera mais disponível para ser adquirida no Brasil. Apesar de ainda não serem tão acessíveis, as opções de câmeras RGB-D estão aumentando, e poderão tornar-se atrativas para uso em áreas como na robótica móvel. Conforme os resultados apresentados no Capítulo 5, os modelos RGB-D apresentaram bons resultados em relação ao algoritmo que utilizou apenas as imagens RGB, embora apenas um trabalho de imagem RGB tenha sido aplicado.

Ainda há muito a ser desenvolvido no uso de imagens RGB-D. A adição da imagem D na solução adiciona algumas questões ao problema, como a existência de pontos cegos, como os identificados no Capítulo 4, objetos com distância maior do que a capacidade de identificação, ou ainda, fontes emissoras que podem interferir na captação, o que deixam a informação da imagem D com diversos ruídos. Mas podemos entender esses problemas como os primeiros problemas do uso de câmeras RGB, como os problemas de foco, distorção, baixa qualidade, ruídos, etc, que atualmente

estão cada vez menores. Embora não explorado nesta pesquisa, a informação D adicional permite também o cálculo de distância entre a fonte de captação das imagens e os objetos.

O desenvolvimento das câmeras RGB-D devem diminuir também o problema identificado nos experimentos, na variedade de padrões na imagem D entre diferentes modelos de câmeras e diferentes parâmetros intrínsecos, que dificultam o desenvolvimento das soluções.

Em relação à comparação entre abordagens tradicionais e abordagens em ML, podemos entender o motivo do crescente uso das abordagens ML na segmentação de objetos. Além da capacidade de segmentar, eles são capazes de também classificar os objetos, identificando não apenas aonde, mas o que é o objeto. As abordagens tradicionais apresentam grande custo em sua modelagem, como também na seleção e configurações dos parâmetros para cada contexto, sendo restrito a sua aplicação em outros contextos. Ainda são ótimas soluções para problemas com um escopo menor e mais definido, como em áreas da medicina, indústria, etc.

Por outro lado, os algoritmos ML apresentam bons resultados, mas tendem a apresentar resultados diferentes, em geral, inferiores, quando aplicados à imagens de outros *datasets*, devido a diferenças nas imagens, captação e cenas. Podem ser aplicados tanto para casos genéricos em diversos ambientes (sem ajustes de parâmetros como nas abordagens tradicionais), quanto restritos a um ambiente, de modo a aumentar a acurácia dos objetos presentes. Possuem as vantagens que algoritmos ML fornecem, como a capacidade de aprender com o tempo, e de existirem muitos *frameworks* que auxiliam na construção de redes neurais, além de fornecer arquiteturas prontas para uso ou adaptação, o que facilitam o desenvolvimento e o compartilhamento de soluções e aplicações.

Entretanto, conforme visto, a exigência computacional de algoritmos ML é grande, o que pode requerer o uso de placas gráficas para problemas com alta complexidade, como o caso de segmentação de objetos. Por isso, em muitos problemas mais restritos e com menor escopo, algoritmos tradicionais ainda apresentam grande interesse.

Nos algoritmos selecionados da abordagem tradicional, nenhum é executado na placa gráfica, como também, nem sempre fazem uso e estão otimizados para operar em todas as *threads* disponíveis pelo processador da máquina, não sendo justa a comparação de desempenho de tempo entre os algoritmos. Porém, isso mostra a dificuldade existente na criação de algoritmos otimizados, e também, demonstra a importância da existência de *frameworks* que auxiliem nessas tarefas, e que fornecem essa otimização de forma transparente, como é o caso dos *frameworks* de ML.

Esta pesquisa também demonstra a dificuldade para implementar a tarefa de segmentação de objetos em um *dataset*. Caso seja de interesse aplicar esse tipo de solução, ainda há um grande trabalho a ser realizado, como na seleção e teste

de algoritmos, seleção ou criação do *dataset*, de forma a tentar obter o máximo de resultado que as escolhas informam. É uma área em constante evolução e que ainda há muito espaço para desenvolvimento de novas soluções e *frameworks*, de forma a tornarem a tarefa de segmentação de objetos cada vez mais precisa e customizável a diferentes problemas.

Portanto, ao final desta pesquisa, podemos inferir os ótimos resultados obtidos por algoritmos ML em relação aos algoritmos tradicionais na tarefa de segmentação de objetos em imagens, além de avaliar a viabilidade de aplicação. Também podemos identificar os pontos positivos da adição da imagem D nas soluções.

O interesse acadêmico no Kinect v2 trouxe efeitos positivos na tecnologia, e a perspectiva é que câmeras cada vez melhores sejam produzidas, tornando melhor o custo-benefício e tornando-as mais atrativas, com mais trabalhos sendo desenvolvidos e novas soluções sejam alcançadas.

6.1 TRABALHOS FUTUROS

Após o desenvolvimento desta pesquisa, foram identificadas as seguintes sugestões para trabalhos futuros, de modo a contribuir com a área de Visão Computacional e Robótica Móvel, e, principalmente, orientar aplicações acadêmicas e comerciais sobre as tecnologias emergentes:

- Realização de pesquisa para comparação dos benefícios do uso de imagens RGB-D em relação ao uso de somente imagens RGB, com o objetivo de esclarecer os benefícios e as aplicações onde possuem maior eficácia. Em um primeiro momento, sem limitação de hardware. Existem bons algoritmos RGB (WU *et al.*, 2019) com requerimento de GPU com 12GB de memória que poderiam apresentar resultados interessantes comparados aos algoritmos RGB-D, com rede neural igualmente robusta.

- Aplicação de algoritmos RGB-D em robótica móvel, utilizando hardware recentes, como NVIDIA Jetson (NVIDIA, 2020), que permitem o uso de placas gráficas em robôs autônomos. Essas novas unidades de processamento móveis poderão expandir as possibilidades e aplicações dos robôs autônomos.

- Realização de pesquisa em inovações na área de *labeling*, ou criação do conjunto verdade, das imagens. Visto que os algoritmos ML apresentam resultados muito bons na segmentação de objetos, mas que requerem treinamento em *datasets* que possuem conjunto verdade. Se essa criação do conjunto verdade for viabilizada, muitas aplicações terão os seus resultados otimizados, além de permitir a ampliação dos *datasets* públicos.

- Realização de pesquisa em algoritmos para correção da imagem D. Através da imagem D e da imagem RGB, formas de remover ou minimizar os ruídos e pontos cegos da imagem D.

Sobre os trabalhos futuros com foco em câmeras RGB-D, uma outra orientação seria elaborar com base em câmeras recentes, como o Azure Kinect, ou outro modelo recente, diferente dos modelos aqui apresentados. A própria evolução desse *hardware*, como visto entre o Kinect v1 e Kinect v2, requer novas pesquisas e soluções. O amadurecimento da tecnologia reflete também em questões de desenvolvimento, como no caso do Azure Kinect, com o SDK multiplataforma e *Open Source* disponibilizado pela Microsoft, em contraste com o SDK das versões anteriores, restritos à plataforma Windows.

REFERÊNCIAS

AGGARWAL, Charu C. **Neural Networks and Deep Learning**: A Textbook. Cham: Springer, 2018. p. 497. ISBN 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0.

ALPAYDIN, Ethem. **Introduction to Machine Learning**. 2nd. [S.l.]: The MIT Press, 2010. ISBN 026201243X.

AMMIRATO, Phil *et al.* A Dataset for Developing and Benchmarking Active Vision. *In*: IEEE International Conference on Robotics and Automation (ICRA). [S.l.: s.n.], 2017.

ARMENI, I. *et al.* Joint 2D-3D-Semantic Data for Indoor Scene Understanding. **ArXiv e-prints**, fev. 2017. arXiv: 1702.01105 [cs.CV].

BARBOSA, Flávio Gabriel Oliveira. **Sistema de localização, mapeamento e registro 3d para robótica móvel baseado em técnicas de visão computacional**. 2017. Diss. (Mestrado) – Universidade Federal de Santa Catarina, Florianópolis.

BERKELEY AI RESEARCH. **Caffe**. [S.l.: s.n.], 15 out. 2018. https://caffe.berkeleyvision.org/.

DENG, Zhuo; TODOROVIC, Sinisa; JAN LATECKI, Longin. Unsupervised Object Region Proposals for RGB-D Indoor Scenes. **Comput. Vis. Image Underst.**, Elsevier Science Inc., USA, v. 154, n. 100, p. 127–136, jan. 2017. ISSN 1077-3142. DOI: 10.1016/j.cviu.2016.07.005.

DOCKER. **Empowering App Development for Developers**. [S.l.: s.n.], 2018. https://www.docker.com/.

EVERINGHAM, M. *et al.* **The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results**. [S.l.: s.n.], 2012. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

FIRMAN, Michael. **RGBD Datasets: Past, Present and Future**. [S.l.: s.n.], 2016. arXiv: 1604.00999 [cs.CV].

FOOLADGAR, Fahimeh; KASAEI, Shohreh. A survey on indoor RGB-D semantic segmentation: from hand-crafted features to deep convolutional neural networks. **Multimedia Tools and Applications**, maio 2019. DOI: 10.1007/s11042-019-7684-3.

FUSIELLO, Andrea. Elements of Computer Vision : Multiple View Geometry. *In*:

GARCIA-GARCIA, Alberto *et al.* A Survey on Deep Learning Techniques for Image and Video Semantic Segmentation. **Applied Soft Computing**, v. 70, maio 2018. DOI: 10.1016/j.asoc.2018.05.018.

GHOSH, Swarnendu *et al.* **Understanding Deep Learning Techniques for Image Segmentation**. [S.l.: s.n.], 2019. arXiv: 1907.06119 [cs.CV].

GONZALEZ, R.C.; WOODS, R.E. **Digital Image Processing**. [S.l.]: Pearson Education, 2011. https://books.google.com.br/books?id=MaYuAAAAQBAJ. ISBN 9780133002324.

GRAUPE, Daniel. **Principles of Artificial Neural Networks**. 2nd. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2007. ISBN 9812706240.

HASNAT, Md. Abul; ALATA, Olivier; TREMEAU, Alain. Joint Color-Spatial-Directional Clustering and Region Merging (JCSD-RM) for Unsupervised RGB-D Image Segmentation. **IEEE Trans. Pattern Anal. Mach. Intell.**, IEEE Computer Society, USA, v. 38, n. 11, p. 2255–2268, nov. 2016. ISSN 0162-8828. DOI: 10.1109/TPAMI.2015.2513407.

HAZIRBAS, Caner *et al.* FuseNet: Incorporating Depth into Semantic Segmentation via Fusion-Based CNN Architecture. *In*: ASIAN Conference on Computer Vision (ACCV). [S.l.: s.n.], 2016. DOI: 10.1007/978-3-319-54181-5_14.

JABREF. **JabRef**. [S.l.: s.n.], 1 jun. 2018. https://www.jabref.org/.

JIANG, Jindong *et al.* **RedNet: Residual Encoder-Decoder Network for indoor RGB-D Semantic Segmentation**. [S.l.: s.n.], 2018. arXiv: 1806.01054 [cs.CV].

JING, Changjuan *et al.* A comparison and analysis of RGB-D cameras' depth performance for robotics application. *In*: p. 1–6. DOI: 10.1109/M2VIP.2017.8211432.

KHAN, Salman *et al.* A Guide to Convolutional Neural Networks for Computer Vision. **Synthesis Lectures on Computer Vision**, v. 8, p. 1–207, fev. 2018. DOI: 10.2200/S00822ED1V01Y201712COV015.

KITCHENHAM, Barbara. **Procedures for Performing Systematic Reviews**. Department of Computer Science, Keele University, UK, 2004.

KRAFT, Marek *et al.* Toward Evaluation of Visual Navigation Algorithms on RGB-D Data from the First- and Second-Generation Kinect. **Mach. Vision Appl.**, Springer-Verlag, Berlin, Heidelberg, v. 28, 1–2, p. 61–74, fev. 2017. ISSN 0932-8092. DOI: 10.1007/s00138-016-0802-6.

LAPLANTE, Phillip A. (Ed.). Encyclopedia of Image Processing. *In*: **Encyclopedia of Image Processing**. [S.l.]: CRC Press, 2018. DOI: 10.1201/9781351032742.

LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. *In*: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], jun. 2015. p. 3431–3440. DOI: 10.1109/CVPR.2015.7298965.

MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento Digital de Imagens**. Rio de Janeiro, Brazil: Editora Brasport, 1999.

MICHELUCCI, Umberto. **Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection**. [S.l.: s.n.], jan. 2019. ISBN 978-1-4842-4975-8. DOI: 10.1007/978-1-4842-4976-5.

MICROSOFT AZURE. **Azure Kinect DK – Desenvolva modelos IA**. [S.l.: s.n.], 11 fev. 2020. https://azure.microsoft.com/pt-br/services/kinect-dk/.

MINAEE, Shervin *et al.* **Image Segmentation Using Deep Learning: A Survey**. [S.l.: s.n.], 2020. arXiv: 2001.05566 [cs.CV].

MITCHELL, Thomas M. **Machine Learning**. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 9780070428072.

MORAIS ALONSO, Ewerton Eyre de *et al.* Introdução ao Kinect For Windows v2. **Tendências e Técnicas em Realidade Virtual e Aumentada**, volume, n. 5, p. 261–274, 2015.

NEIVA, Frâncila; SILVA, Rodrigo. Revisão Sistemática da Literatura em Ciência da Computação - Um Guia Prático, jun. 2016. DOI: 10.13140/RG.2.1.1445.3361.

NVIDIA. **NVIDIA Embedded Systems for Next-Gen Autonomous Machines**. [S.l.: s.n.], 11 fev. 2020. https://www.nvidia.com/pt-br/autonomous-machines/embedded-systems/.

PEDREGOSA, F. *et al.* Scikit-learn: Machine Learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PEDRINI, H.; SCHWARTZ, W. R. **Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações**. [S.l.]: Editora Thomson Learning, 2007. p. 528. ISBN 978-85-221-0595-3.

PYTORCH. **An open source machine learning framework that accelerates the path from research prototyping to production deployment**. [S.l.: s.n.], 21 out. 2018. https://pytorch.org/.

RAHMAN, Mohammad *et al.* Recent Advances in 3D Object Detection in the Era of Deep Neural Networks: A Survey. **IEEE Transactions on Image Processing**, PP, p. 1–1, nov. 2019. DOI: 10.1109/TIP.2019.2955239.

REN, X.; BO, L.; FOX, D. RGB-(D) scene labeling: Features and algorithms. *In*: 2012 IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2012. p. 2759–2766.

RUSSELL, Stuart; NORVIG, Peter. **Artificial Intelligence: A Modern Approach**. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 9780136042594.

RUSU, Radu Bogdan; COUSINS, Steve. 3D is here: Point Cloud Library (PCL). *In*: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA). Shanghai, China: [s.n.], maio 2011.

SARBOLANDI, Hamed; LEFLOCH, Damien; KOLB, Andreas. **Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect**. [S.l.: s.n.], 2015. arXiv: 1505.05459 [cs.CV].

SEIF, George. **Semantic Segmentation Suite in TensorFlow**. [S.l.: s.n.], 2019. https://github.com/GeorgeSeif/Semantic-Segmentation-Suite.

SIEGWART, Roland; NOURBAKHSH, Illah R.; SCARAMUZZA, Davide. **Introduction to Autonomous Mobile Robots**. 2nd. [S.l.]: The MIT Press, 2011. ISBN 9780262015356.

SILBERMAN, Nathan *et al.* Indoor Segmentation and Support Inference from RGBD Images. *In*: PROCEEDINGS of the 12th European Conference on Computer Vision - Volume Part V. Florence, Italy: Springer-Verlag, 2012. (ECCV'12), p. 746–760. DOI: 10.1007/978-3-642-33715-4_54.

SOLOMON, Chris; BRECKON, Toby. **Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab**. 1st. [S.l.]: Wiley Publishing, 2011. ISBN 9780470844731.

SONG, Shuran; LICHTENBERG, Samuel; XIAO, Jianxiong. SUN RGB-D: A RGB-D scene understanding benchmark suite. *In*: p. 567–576. DOI: 10.1109/CVPR.2015.7298655.

THE DOCUMENT FOUNDATION. **LibreOffice - A melhor suite office livre**. [S.l.: s.n.], 1 jun. 2018. https://pt-br.libreoffice.org/.

THE MATHWORKS. **Matlab**. [S.l.: s.n.], 3 nov. 2019. https://www.mathworks.com/products/matlab.html.

TOSCANA, Giorgio; ROSA, Stefano. **Fast Graph-Based Object Segmentation for RGB-D Images**. [S.l.: s.n.], 2016. arXiv: 1605.03746 [cs.CV].

WADA, Kentaro. **labelme: Image Polygonal Annotation with Python**. [S.l.: s.n.], 2016. https://github.com/wkentaro/labelme.

WARD, Isaac Ronald; LAGA, Hamid; BENNAMOUN, Mohammed. **RGB-D image-based Object Detection: from Traditional Methods to Deep Learning Techniques**. [S.l.: s.n.], 2019. arXiv: 1907.09236 [cs.CV].

WU, Yuxin *et al.* **Detectron2**. [S.l.: s.n.], 2019. https://github.com/facebookresearch/detectron2.

ZIS, Luiz Eduardo. **A Object Recognition Research Framework for RGB-D Segmentation**. [S.l.]: GitHub, 2019. https://github.com/zisluiz/obeach/.

ZIS, Luiz Eduardo. **Fast Graph-Based Object Segmentation for RGB-D Images**. [S.l.]: GitHub, 2019. https://github.com/zisluiz/graph-canny-segm/.

ZIS, Luiz Eduardo. **FCN Tensorflow**. [S.l.]: GitHub, 2019. https://github.com/zisluiz/FCN.tensorflow/.

ZIS, Luiz Eduardo. **FuseNet implementation in PyTorch**. [S.l.]: GitHub, 2019. https://github.com/zisluiz/fusenet-pytorch/.

ZIS, Luiz Eduardo. **JCSA-RM RGBD Image Segmentation and Analysis Method**. [S.l.]: GitHub, 2019. https://github.com/zisluiz/JCSA_RM_Image_Seg/.

ZIS, Luiz Eduardo. **Labelme**. [S.l.]: GitHub, 2019. https://github.com/zisluiz/labelme.

ZIS, Luiz Eduardo. **RedNet**. [S.l.]: GitHub, 2019. https://github.com/zisluiz/RedNet/.

ZIS, Luiz Eduardo. **RGBD-object-propsal**. [S.l.]: GitHub, 2019. https://github.com/zisluiz/RGBD-object-propsal/.

APÊNDICE A – PARÂMETROS UTILIZADOS PARA O GRAPH CANNY SEGMENTATION

Os parâmetros originais dos autores foram ajustados para os experimentos desta pesquisa. Os autores originalmente fornecem três configurações diferentes, criados para os seguintes *datasets*: ACCV (Asian Conference on Computer Vision) *dataset*, Rutgers *dataset* e ICCV Challenge (IEEE International Conference on Computer Vision) *dataset*.

Para este trabalho, a resolução das imagens e o contexto das imagens são diferentes, o que indica a necessidade de avaliar os parâmetros com os resultados, de forma a identificar os melhores ajustes. Portanto, foram utilizados os seguintes parâmetros nos experimentos apresentados no Capítulo 5:

- fx: 1078,68499[1], 759,680[2] (572.41140) - Comprimento focal x da câmera;
- fy: 1076,4742562[1], 759,680[2] (573.57043) - Comprimento focal x da câmera;
- cx: 952,6592286[1], 540,0[2] (325.26110) - Centro óptico x da câmera;
- cy: 530,7386644[1], 540,0[2] (242.04899) - Centro óptico y da câmera;
- k: 12 (38) *threshold* dos pesos para unificar regiões adjacentes;
- kx: 2000 (2000);
- ks: 50 (50);
- ky: 30 (30);
- kdv: 4,5 (4,5);
- kdc: 0,1 (0,1);
- min size: 500,0 (500,0);
- sigma: 0,8 (0,8);
- max ecc: 0,978 (0,978);
- max L1: 3800,0 (3800,0);
- max L2: 950,0 (950,0);
- DTH: 30 (30);
- plusD: 30 (30);

- point3D: 4 (5) - para o *threshold* de borda entre regiões após a aplicação do Canny modificado;
- g angle: 154 (154);
- l angle: 56 (56);
- Lcanny: 28 (50) - *threshold* mínimo do algoritmo Canny;
- Hcanny: 110 (75) - *threshold* máximo do algoritmo Canny; e
- FarObjZ: 25000 (1800) - distância máxima dos objetos a serem considerados.

Os valores em parênteses são os originais utilizados pelos autores para o *dataset* ACCV. Os valores indicados com [1] são os valores utilizados para os *datasets* Active Vision e PUTKK, já os indicados com [2] são utilizados para os conjuntos S3DS.

APÊNDICE B – INFORMAÇÕES COMPLEMENTARES DOS RESULTADOS

A Tabela 9 apresenta os resultados obtidos por algoritmo e por conjunto, mas considerando apenas a melhor região (melhor IoU), por instância de classe.

Tabela 9 – Resultados dos algoritmos baseados das predições de regiões com melhor IoU de cada instância das classes.

Algoritmo	**Conjunto**	**Acurácia**	**Precisão**	**Recall**	**F1**	**IoU**
RGBD Proposal	Active Vision	58,41	87,11	58,41	63,82	52,02
RGBD Proposal	PUTKK	61,47	**88,48**	61,47	**67,45**	**55,30**
RGBD Proposal	S3DS S/T	58,56	87,73	58,56	65,32	52,44
RGBD Proposal	S3DS C/T	55,40	**84,87**	55,40	**61,63**	**49,18**
JCSA-RM	Active Vision	48,86	86,55	48,86	55,62	43,16
JCSA-RM	PUTKK	41,53	78,90	41,53	46,72	34,81
JCSA-RM	S3DS S/T	64,17	87,24	64,17	68,46	55,71
JCSA-RM	S3DS C/T	56,72	83,27	56,72	60,27	47,73
Graph Canny - Somente Obj.	Active Vision	36,34	**87,40**	36,34	42,23	32,01
Graph Canny - Somente Obj.	PUTKK	49,52	82,69	49,52	52,21	41,86
Graph Canny - Somente Obj.	S3DS S/T	45,58	**88,96**	45,58	51,25	40,58
Graph Canny - Somente Obj.	S3DS C/T	41,35	83,09	41,35	45,75	34,35
Graph Canny - Regiões	Active Vision	**62,90**	85,63	**62,90**	**66,33**	**54,21**
Graph Canny - Regiões	PUTKK	**65,04**	81,09	**65,04**	67,18	54,32
Graph Canny - Regiões	S3DS S/T	64,97	87,75	64,97	69,96	58,03
Graph Canny - Regiões	S3DS C/T	58,20	78,79	58,20	59,45	46,25
FuseNet	Active Vision	43,60	76,80	43,60	42,67	31,94
FuseNet	PUTKK	36,51	68,70	36,51	33,60	23,28
FuseNet	S3DS S/T	41,20	78,01	41,20	40,31	29,71
FuseNet	S3DS C/T	39,63	73,92	39,63	34,92	25,36
RedNet	Active Vision	56,68	85,85	56,68	57,82	47,91
RedNet	PUTKK	54,81	80,65	54,81	55,01	43,71
RedNet	S3DS S/T	**71,22**	87,57	**71,22**	**72,36**	**62,88**
RedNet	S3DS C/T	54,62	81,65	54,62	53,99	43,42
FCN	Active Vision	60,34	85,22	60,34	62,11	50,82
FCN	PUTKK	49,60	78,37	49,60	50,38	38,87
FCN	S3DS S/T	42,55	71,88	42,55	44,52	32,08
FCN	S3DS C/T	**58,64**	81,77	**58,64**	60,00	47,96