UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MESTRADO ACADÊMICO EM ENGENHARIA ELÉTRICA

FRANCISCO GEILSON DE LIMA XAVIER

UMA ABORDAGEM PARA ESTIMATIVA DE LOCALIZAÇÃO E SUPORTE DE NAVEGAÇÃO AUXILIADO POR MAPA DE ESTIMATIVA DE PROFUNDIDADE

FORTALEZA

2023

FRANCISCO GEILSON DE LIMA XAVIER

UMA ABORDAGEM PARA ESTIMATIVA DE LOCALIZAÇÃO E SUPORTE DE NAVEGAÇÃO AUXILIADO POR MAPA DE ESTIMATIVA DE PROFUNDIDADE

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia Elétrica do Programa de Pós-Graduação em Engenharia Elétrica do Centro de Ciências e Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia Elétrica. Área de Concentração: Sinais e Sistemas

Orientador: Prof. Dr. Pedro Pedrosa Rebouças Filho

Coorientador: Prof. Dr. Pedro Henrique Feijó de Sousa

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

X19a Xavier, Francisco Geilson de Lima.
Uma abordagem para estimativa de localização e suporte de navegação auxiliado por mapa de estimativa de profundidade / Francisco Geilson de Lima Xavier. – 2023.
80 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Fortaleza, 2023.
Orientação: Prof. Dr. Pedro Pedrosa Rebouças Filho.
Coorientação: Prof. Dr. Pedro Henrique Feijó de Sousa.

1. Estimativa de profundidade. 2. Transfer Learning. 3. Robôs Móveis. 4. Redes Neurais. I. Título.
CDD 621.3

FRANCISCO GEILSON DE LIMA XAVIER

UMA ABORDAGEM PARA ESTIMATIVA DE LOCALIZAÇÃO E SUPORTE DE NAVEGAÇÃO AUXILIADO POR MAPA DE ESTIMATIVA DE PROFUNDIDADE

Dissertação apresentada ao Curso de Mestrado Acadêmico em Engenharia Elétrica do Programa de Pós-Graduação em Engenharia Elétrica do Centro de Ciências e Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Engenharia Elétrica. Área de Concentração: Sinais e Sistemas

Avaliada em: 30/06/2023

BANCA EXAMINADORA

Prof. Dr. Pedro Pedrosa Rebouças Filho (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Pedro Henrique Feijó de Sousa (Coorientador)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Prof. Dr. Fabrício Gonzalez Nogueira
Universidade Federal do Ceará (UFC)

Prof. Dr. Roger Moura Sarmento
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

À minha família, por estarem sempre ao meu lado em todos os momentos da vida, pelo apoio incondicional e constante incentivo.

AGRADECIMENTOS

À minha família, por todo amor, paciência, carinho e apoio moral que sempre me prestaram em todas as etapas de minha vida.

Ao meu orientador e amigo, Prof. Dr. Pedro Pedrosa Rebouças Filho, pela orientação, companheirismo, conselhos e ensinamentos ao decorrer desta pesquisa.

Ao meu coorientador, Prof. Dr. Pedro Henrique Feijó de Sousa, pela orientação, ajuda e ensinamentos ao decorrer desta pesquisa.

Aos professores e colaboradores de pesquisa do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Ceará (PPGEE-UFC).

À todos os amigos e parceiros do Laboratório de Processamento de Imagens, Sinais e Computação Aplicada (LAPISCO) e do Grupo de Pesquisa em Automação, Controle e Robótica (GPAR), que contribuíram diretamente para minha evolução pessoal e profissional.

Assim como a todos aqueles que contribuíram de alguma maneira com o desenvolvimento desta pesquisa.

“Acredite em si mesmo e tudo será possível.”

(Elvis Presley)

RESUMO

A detecção de profundidade é essencial para muitas tarefas robóticas, com mapeamento, localização e prevenção de obstáculos. Em pequenas plataformas robóticas, as limitações de peso, volume e consumo de energia são alguns dos problemas que motivam o uso de estimativa de profundidade usando uma câmera monocular em vez de sensores de profundidade. Esta dissertação propõe uma abordagem para localização de robôs móveis autônomos em um ambiente interno usando visão monocular auxiliada por mapas de estimativa de profundidade a partir de uma única imagem de entrada RGB aplicada ao conceito de Transfer Learning vinculado a Redes Neurais Convolucionais (CNNs). O desempenho dos classificadores na estimativa da localização foi observado e comparado usando uma configuração única de imagens RGB-D transformadas em uma imagem em mosaico. As imagens foram combinadas com o poder descritivo das CNNs nos seguintes cenários: profundidade capturada pelo sensor Kinect e estimativa de profundidade gerada pelo bloco AdaBins. Os resultados mostram que a abordagem proposta atingiu 99,8% em Acurácia e F1-Score. Com os resultados, os desempenhos foram analisados com base no tempo de extração de características e treinamento, atingindo 7,929ms e 0,022s, respectivamente, para a melhor combinação de arquitetura e classificador na abordabem proposta.

Palavras-chave: Estimativa de Profundidade, *Transfer Learning*, Robôs Móveis e Redes Neurais.

ABSTRACT

Depth detection is essential for many robotic tasks, including mapping, localization, and obstacle prevention. On small robotic platforms, weight, volume, and power consumption limitations are some of the challenges that motivate the use of depth estimation using a monocular camera rather than depth sensors. This dissertation proposes an approach for the localization of autonomous mobile robots in an indoor environment using monocular vision aided by depth estimation maps from a single RGB input image, applied to the concept of Transfer Learning tied to Convolutional Neural Networks (CNNs). The performance of the classifiers in estimating the location was observed and compared using a unique configuration of RGB-D images transformed into a mosaic image. The images were combined with the descriptive power of the CNNs in the following scenarios: depth captured by the Kinect sensor and depth estimation generated by the AdaBins block. The results show that the proposed approach achieved 99.8% in Accuracy and F1-Score. Based on these results, the performances were analyzed concerning feature extraction time and training, achieving 7.929ms and 0.022s, respectively, for the best combination of architecture and classifier in the proposed approach.

Keywords: Depth Estimation, *Transfer Learning*, Mobile Robots and Neural Networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GPS	*Global Positioning System*
RFID	*Radio-Frequency Identification*
RGB-D	*RED, GREEN, BLUE AND DEPTH*
LiDAR	*Light Detection and Ranging*
RADAR	*Radio Detection and Ranging*
SONAR	*Sound Navigation and Ranging*
CNN	*Convolutional Neural Networks*
kNN	*k-Nearest Neighbor*
RF	*Random Forest*
MLP	*Multilayer Perceptron*
SVM	*Support Vector Machine*
DDQN	*Double Deep Q-Network*
2D	*Two-Dimensional*
SLAM	*Simultaneous localization and mapping*
3D	*Three-Dimensional*
ROS	*Robot Operating System*
RGB	*RED, GREEN AND BLUE*
RTAB-Map	*Real-Time Appearance-Based Mapping*
YOLO V3	*You Only Look Once, Version 3*
PCA	*Principal component analysis*
GNSS-INS	*Inertial Navigation System*
DT	*Decision Trees*
ILSVRC	*ImageNet Large Scale Visual Recognition Challenge*
ReLU	*Rectified Linear Unit*
NAS	*Neural Architecture Search*
CIFAR-10	*Canadian Institute for Advanced Research, 10 classes*
DenseNet	*Densely Connected Convolutional Networks*
RNN	*Recurrent Neural Network*
U-Net	*Convolutional Networks for Biomedical Image Segmentation*
VP	Verdadeiro Positivo

VN	Verdadeiro Negativo
FP	Falso Positivo
FN	Falso Negativo

SUMÁRIO

1 INTRODUÇÃO

Aplicações de Robótica Móvel podem ser encontradas em ambientes agrícolas, industriais, militares, domésticos, entre outros. Devido à grande variedade de usos desta tecnologia, a tarefa de localização é um dos tópicos mais pesquisados em relação a robôs móveis (LI; SHI, 2018), já que esta é um meio para se chegar há um fim, ou seja, busca e salvamento, mineração, entrega de pessoas e cargas, todas essas atividades dependem dela.

A tarefa de localização deve ser executada por um sistema apropriado de acordo com o ambiente operacional. Em ambientes externos, o *Global Positioning System* (GPS) é um dos métodos mais utilizados (KURDI *et al.*, 2017). No entanto, por possuir precisão em metros e não operar adequadamente em ambientes fechados, não é um equipamento recomendado para realização de localização de robês móveis em ambientes internos (DIOP *et al.*, 2016). Para ambientes internos, alguns dos equipametnos mais utilizados são o Wi-Fi, o *Radio-Frequency Identification* (RFID), o Ultrassom e o Bluetooth (SONG *et al.*, 2011).

De acordo com Zhang *et al.* (2016) e Wang *et al.* (2020), o Reconhecimento de Imagem baseado em um sistema de visão computacional é um dos métodos mais recentes para localizar robôs móveis em ambientes internos. Seu uso é vantajoso, pois requer menos espaço para instalação, é robusto às condições ambientais, captura imagens RGB, é uma tecnologia amplamente difundida, tem consumo eficiente de energia e baixo custo (MA; KARAMAN, 2018).

Mas a tarefa de localização não é apenas a definição da posição absoluta no espaço (ELARABY *et al.*, 2018). Também é necessário elaborar um mapa do ambiente a ser navegado e, a partir dele, determinar o ponto que o robô se encontra. Para isso, são utilizadas abordagens geométricas e topológicas. O mapeamento utilizando abordagem geométrica possui o espaço de navegação descrito a partir de um sistema de coordenadas globais (JIANG *et al.*, 2017) enquanto na abordagem topológica é tratado como um grafo (CHIN *et al.*, 2016).

Além disso, é extremamente importante que as plataformas robóticas tenham a capacidade de explorar pistas de profundidade do ambiente. O sistema de percepção deve ser capaz de estimar a distância em que os objetos estão posicionados na cena e, a partir disso, evitar obstáculos e realizar um deslocamento mais seguro em superfícies navegáveis de diversos cenários.

Sensores *RED, GREEN, BLUE AND DEPTH* (RGB-D), *Light Detection and Ranging* (LiDAR), câmeras estéreo, *Radio Detection and Ranging* (RADAR) e *Sound Navigation and*

Ranging (SONAR) são tecnologias amplamente comercializadas para detecção de profundidade. No entanto, existem limitações quanto ao uso dessas tecnologias para aplicações de percepção de profundidade. Os sensores LiDAR têm custo proibitivo e realizam medições de profundidade esparsas e ruidosas em longas distâncias, os sensores Kinect são sensíveis à luz e têm um limite de medição de distância menor quando comparados ao LiDAR, as câmeras estéreo falham em regiões reflexivas e de baixa textura, enquanto o RADAR e o SONAR são geralmente usados como auxiliares (MENDES *et al.*, 2021).

Diante do cenário esposto acima, a abordagem de estimativa de profundidade baseada em sistemas de visão tem sido avaliada por vários investigadores em todo o mundo, e o problema tem sido um campo de pesquisa interessante (MASOUMIAN *et al.*, 2022). Com o progresso dos modelos recentes de aprendizado profundo, a estimativa de profundidade baseada em modelos aprendizado profundo conseguiu demonstrar sua notável eficiência em muitas aplicações (KHAN *et al.*, 2020; TOSI *et al.*, 2019; RAMAMONJISOA; LEPETIT, 2019).

Diante do esposto, este trabalho propõe uma aobordagem para localização e navegação de robôs móveis por visão computacional auxiliada por estimativa de profundidade, utilizando *Convolutional Neural Networks* (CNN)s aliadas ao conceito de *Transfer Learning*, que permite que este método seja aplicado como um extrator de recursos. Os classificadores testados foram: Classificador Bayesiano, *k-Nearest Neighbor* (*k-Nearest Neighbor* (kNN)), *Random Forest* (*Random Forest* (RF)), *Multi-layer Perceptron* (*Multilayer Perceptron* (MLP)) e *Support Vector Machine* (*Support Vector Machine* (SVM)).

Para este trabalho, um ambiente interno foi mapeado usando a abordagem topológica. As imagens foram adquiridas pelo sensor Microsoft Kinect RGB-D e os seguintes conjuntos de dados foram construídos: imagens RGB; imagens de profundidade capturadas; imagens de estimativa de profundidade; imagens de mosaico RGB-D com a profundidade capturada e imagens de mosaico RGB-D com a profundidade estimada.

A estratégia desta solução é baseada na estimativa de localização por visão computacional usando as imagens em mosaico da combinação das imagens RGB e estimativa de profundidade com aplicações práticas diretamente ligadas a robôs móveis usando a visão monocular, como alternativa aos sensores mais robustos como LiDAR e sensores de profundidade.

1.1 Estado da arte

Nos últimos anos, vários trabalhos abordaram o tema de localização e navegação de robôs móveis a partir de diferentes métodos e abordagens. Em Silva *et al.* (2020) o sensor Kinect é utilizado para obter a localização do robô móvel através de imagens RGB e profundidade dispostas em mosaico. Os autores utilizam o conceito de transferencia de aprendizado para utilizar CNNs como extrator de características e métodos de aprendizado de máquina para a tarefa de localização e navegação.

Yokoyama e Morioka (2020) desenvolveram um sistema de navegação de robô móvel autônomo baseado em aprendizado por reforço profundo, *Double Deep Q-Network* (*Double Deep Q-Network* (DDQN)), com uma câmera monocular, sem *Two-Dimensional* (2D)-LiDAR. O sistema requer os dados de entrada como estados de DDQN que incluem os dados de alcance ao redor do robô. Esses dados são estimados a partir de uma câmera monocular. Dessa forma, o sistema proposto converte as imagens de profundidade estimadas geradas a partir da câmera monocular em dados de alcance 2D que são inseridos no modelo aprendido com base no plano 2D.

Loo *et al.* (2021) propõem um sistema *Simultaneous localization and mapping* (SLAM) monocular denso, chamado *DeepRelativeFusion*, que é capaz de recuperar uma estrutura *Three-Dimensional* (3D) globalmente consistente. Para esse fim, utilizam um algoritmo SLAM visual para recuperar de forma confiável as poses da câmera e os mapas de profundidade semi-densos dos quadros-chave e, em seguida, utilizam a previsão de profundidade relativa para densificar os mapas de profundidade semi-densos e refinar o gráfico de poses do quadro-chave. A melhoria dos mapas de profundidade semi-densos é realizada por filtragem adaptativa, que é um filtro de suavização de média ponderada que preserva a estrutura e leva em consideração a intensidade do pixel e a profundidade dos pixels vizinhos. Já a densificação dos mapas de profundidade é realizada pelo modelo *DeepFusion* com uma função de custo aprimorada e o uso da previsão de profundidade relativa de imagem única.

Faria e Moreira (2021) apresentam uma plataforma móvel baseada em *Robot Operating System* (ROS) e avaliam a possibilidade de navegação com câmeras monoculares dentro de um ambiente industrial. Para isso os autores propõem a utilização de um sistema de inteligência artificial para criar uma imagem de profundidade estimada a partir das imagens 2D *RED, GREEN AND BLUE* (RGB). Também utilizando uma plataforma de desenvolvimento ROS, Kulkarni *et al.* (2021) propõem o uso do método tradicional Visual SLAM acompanhado pela detecção de

objetos utilizando CNNs pré-treinadas com o intuito de aprimorar as capacidades de um robô *Turtlebot* de navegar com eficiência e realizar uma percepção 3D robusta em ambientes internos. O Visual SLAM é executado a partir da abordagem *Real-Time Appearance-Based Mapping* (RTAB-Map) (baseada em gráfico RGB-D) e a detecção de objetos é realizada pelo modelo *You Only Look Once, Version 3* (YOLO V3).

Taha e Cansever (2021) realizam a localização robusta de robôs reduzindo as imagens 3D em imagens 2D usando o algoritmo *Principal component analysis* (PCA) e usando CNN para extração de características e classificação das imagens.

Papa *et al.* (2022) apresentam, uma arquitetura codificador-decodificador de agrupamento piramidal separável projetada para obter desempenhos de frequência em tempo real. O modelo proposto é uma arquitetura profunda de processamento rápido para estimativa de profundidade monocular capaz de obter estimativas de profundidade com alta precisão a partir de imagens de baixa resolução utilizando recursos mínimos de hardware. O modelo proposto explora duas camadas de agrupamento piramidal separáveis em profundidade, que permitem aumentar a frequência de inferência enquanto reduzem a complexidade computacional geral.

Patil *et al.* (2022) propõem a utilização de dados de profundidade baseados em mapas como uma entrada adicional ao invés de sensores de profundidade. Para isso apresentam um método de mapeamento que funciona com conjuntos de dados comuns de direção autônoma e permite uma localização precisa usando uma mistura de *Inertial Navigation System* (GNSS-INS) e técnicas baseadas em imagem. O método lida com incompatibilidades de primeiro e segundo plano entre a profundidade anterior baseada no mapa e a cena real.

Jin *et al.* (2022) apresentam um sistema SLAM completo baseado em câmera monocular. Os autores propõem um método que aprende a estimativa de profundidade por DenseNet e CNN para um sistema SLAM monocular. Para isso, utilizam uma arquitetura de codificador-decodificador baseada em transferência de aprendizado e redes neurais convolucionais com o intuito de estimar a informação de profundidade de imagens RGB monoculares. Paralelamente, por meio da extração de recursos ORB de *front-end* e do método de otimização de ajuste de pacote RGB-D direto de back-end, é possível obter poses de câmera precisas e o mapeamento interno denso utilizando as informações de profundidade estimada. O modelo de estimativa de profundidade monocular proposto apresenta bons resultados em comparação aos modelos atuais da literatura. Por esse motivo, o erro de estimativa de pose da câmera também é menor do que as soluções SLAM monoculares tradicionais.

Zhang *et al.* (2022) propõem utilizar os resultados da estimativa de profundidade monocular para gerar mapas de pseudoprofundidade 16 *bits* e combiná-los com as imagens monoculares originais criando imagens pseudo RGB-D para utiliza-lás em SLAM. Os experimentos foram realizados no conjunto de dados KITTI utilizando o algorítimo de estimação de profundidade *MonoDepth2* (GODARD *et al.*, 2019) e *framework* ORB-SLAM3 que é baseado em câmera RGB-D. Os resultados indicam que a abordagem proposta consegue alcançar resultados tão satisfatórios em SLAM quanto em computação estéreo.

Uma abordagem prática e de baixo custo para evitar colisões de um sistema multi-robô utilizando uma única câmera é aprensentada em Rocchi *et al.* (2023). Os autores fornecem uma solução alternativa para medição de alcance e detecção de ambiente, substituindo sensores de distância comuns, como sensores LiDAR e sensores ultrassônicos. Para isso, uma CNN é aplicada para estimar a profundidade dos obstáculos detectados a partir da imagem de uma câmera monocular, auxiliando uma equipe de robôs móveis a detectar obstáculos em um ambiente desconhecido, determinando estratégias de navegação e superando a limitação do sensor LiDAR integrado. A navegação é realizada por um controlador de desvio foi projetado sobre um método de campo de potencial artificial (APF) modificado, levando os robôs a evitar obstáculos para alcançar o ponto objetivo.

Já o método proposto por Kumar *et al.* (2023) realiza a navegação autônoma usando um sistema embarcado leve e câmeras monoculares, combinando recursos de processamento de imagem e compartilhamento de recursos. A arquitetura proposta faz uso de sinais monoculares e modelo de estimativa de profundidade MiDaS para evitar obstáculos.

Desta forma, as abordagens citadas acima servem de base para o desenvolvimento deste trabalho. Neste trabalho, utilizamos o sensor Kinect como um dispositivo de visão monocular. A partir das imagens RGB coletadas, geramos imagens de estimativas de profundidade e criamos um mosaico RGB-D para obter a localização e realizar a navagaçao do robô móvel, dispensando o uso de outros sensores.

Assim, o foco do trabalho proposto é usar o *Transfer Learning* para criar mapas de profundidade a partir de modelos pré-treinados em conjuntos de dados internos populares. Avaliar o uso de um mapa de imagem de visão monocular e mapa de imagem de estimativa de profundidade com configuração de imagem em mosaico para estimar a localização e realizar a navegação do robô em um ambiente interno usando CNNs como extrator de recursos aplicando também o conceito de *Transfer Learning*, assim como avaliar diferentes métodos de aprendizado

de máquina.

As principais contribuições deste trabalho são:

- Uso eficiente de uma imagem em mosaico RGB-D;
- Uso e avaliação de CNNs como extratores de recursos combinados com métodos clássicos de aprendizado de máquina;
- Uso e avaliação de um estimador de profundidade na tarefa de localização e navegação de robôs móveis;
- Avaliação e comparação da estimativa de localização interna, baseada em visão computacional, com o uso de sensor de profundidade e estimador de profundidade;
- Excelentes resultados com a aplicação da técnica de *Transfer Learning*.

1.2 Objetivos

O objetivo principal deste trabalho é analisar uma abordagem para localização e navegação utilizando um estimador de profundidade monocular e redes neurais convolucionais baseadas no conceito de *Transger Learning*.

1.2.1 Objetivos específicos

De maneira mais específica, este trabalho tem os seguintes objetivos:

- Analisar o uso de imagens de profundidade obtidas a partir sensor Kinect, na tarefa de localização de robôs móveis;
- Analisar o uso de imagens obtidas a partir do estimador de profundidade AdaBins pré-treinado com o conjunto de dados NYU DEPTH V2, na tarefa de localização de robôs móveis;
- Utilizar as imagens obtidas a partir do sensor Kinect e estimador de profundidade em configuração de mosaico RGB-D na tarefa de localização de robôs móveis;
- Utilizar técnicas de Visão Computacional para o reconhecimento de local interno;
- Explorar várias arquiteturas de CNN como extratores de características, com base no conceito de *Tranfer Learning*;
- Extrair atributos, a partir dos conjuntos de imagens trabalhados, usando 11 modelos diferentes de CNNs pré-treinadas, como: VGG16, VGG19, MobileNet, InceptionV3, Xception, ResNet50, InceptionResNetV2, NASNetMobile, DenseNet121, DenseNet169 e

DenseNet201;

- Aplicar 5 métodos de classificação: Naive Bayes, *k*-Vizinhos Mais Próximos, Floresta Aleatória, Perceptron Múltiplas Camadas e Máquina de Vetores de Suporte;
- Avaliar um sistema localização e suporte à navegação por visão monocular com o uso de estimador de profundidade em mapas topologicos para robos móveis usando CNN, incorporando o conceito de *Tranfer Learning*;
- Analisar a relevância da abordagem proposta utilizando métricas de avaliação como Acurácia e F1-Score, além do tempo de processamento.

1.3 Organização do trabalho

Esta dissertação está organizada em cinco capítulos. Além do Capítulo 1, o qual introduz este trabalho, no Capítulo 2 está disposta a fundamentação teórica com a descrição das técnicas utilizadas. Em seguida, no Capítulo 3 a metodologia é apresentada com todos os processos realizados sendo destacados. O Capítulo 4 apresenta uma análise dos resultados obtidos com os experimentos. Por fim, o Capítulo 5 trás as conclusões alcançadas e propostas para futuros trabalhos.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 são descritos os principais métodos de aprendizado de máquinas para a classificação. Na seção 2.2 é destaque as redes neurais convolucionais, do inglês, Convolutional Neural Network (CNN). E por fim, na seção 2.3 são abordados técnicas e detalhes de entrada e saída de dados presentes na tarefa de estimativa de profundidade baseado em aprendizado profundo.

2.1 *Métodos de Aprendizagem de Máquinas*

A aprendizagem de máquina, também conhecida como machine learning, é uma técnica proposta para solucionar tarefas complexas para os seres humanos. Ao fornecer uma grande quantidade de dados, o algoritmo de aprendizagem explora e busca modelos que alcancem um objetivo específico. Redes neurais são abordagens computacionais que empregam modelos matemáticos baseados na estrutura neural interconectada, funcionando de maneira análoga aos neurônios do cérebro humano. Utilizando algoritmos, ela é capaz de identificar padrões e descobrir correlações em dados brutos, bem como agrupar e classificar esses dados. Com o passar do tempo, ela aprende e melhora continuamente (KOVÁCS, 2002).

A tarefa de classificação pode ser realizada nas características profundas retornadas pelas CNNs. Esta seção resume as cinco técnicas de aprendizado de máquina usadas neste trabalho.

2.1.1 Classificador Bayesiano

O Classificador Bayesiano é uma técnica probabilística e supervisionada. É usado para categorizar as amostras pela proporção de cada amostra que pertence a uma determinada classe (THEODORIDIS; KOUTROUMBAS, 2008). Para isso marca as amostras com base no valor de probabilidade a posteriori que é calculado a partir de densidades condicionais e probabilidade a priori Theodoridis2008PatternRF. O teorema de Bayes afirma a relação formulada na Equação 2.1.

$$P(c|f_1,...,f_k) = \frac{P(c) \cdot P(f_1,...,f_k|c)}{P(f_1,...,f_k)}, \tag{2.1}$$

onde c é a variável de classe, $P(c)$ é Probabilidade *a Priori* que pode ser interpretada como a frequência relativa da classe no conjunto de treinamento, k é o número de atributos e de f_1 a f_k

são os elementos vetoriais de atributos dependentes. Com base na proposição da independência, pode-se afirmar que:

$$P(f_i|c, f_1, ..., f_{i-1}, f_{i+1}, ..., f_k) = P(f_i|c). \tag{2.2}$$

Dessa forma, o teorema de Bayes pode ser re-escrito de acordo com a Equação 2.3.

$$P(c|f_1, ..., f_k) = \frac{P(c) \cdot \prod_{i=1}^{k} P(f_i|c)}{P(f_1, ..., f_k)}. \tag{2.3}$$

No entanto, $P(f_1, ..., f_k)$ é constante para todas as classes possíveis. Assim, utilizando a regra de decisão de Máxima Probabilidade *a Posteriori* (MAP), formulada na Equação 2.4, a Equação 2.3 pode ser ainda mais simplificada como formulado na Equação 2.5.

$$P(c|f_1, ..., f_k) \propto P(c) \cdot \prod_{i=1}^{k} P(f_i|c) \tag{2.4}$$

$$\widehat{c} = argmaxcP(c) \cdot \prod_{i=1}^{k} P(f_i|c). \tag{2.5}$$

Como exemplo, siguinifica dizer que para classificar corretamente uma imagem, será necessário executar três etapas. Primeiro será calculado o produto da probabilidade de ocorrência das características (probabilidade *a posteriori*) para cada classe. Em seguida, haverá a multiplicação dessas probabilidades pela frequência relativa (no conjunto de treinamento) de sua respectiva classe (probabilidade *a priori*). E, por fim, será selecionada a classe com maior probabilidade.

Neste trabalho foi utilizado o classificador Naive Bayes Gaussiano. De uma forma geral, ele assume que a probabilidade *a priori*, ou mesmo a função densidade de probabilidade, tem distribuição Gaussiana, que também é conhecida como distribuição Normal. A distribruição Gaussiana é formulada de acordo com a Equação 2.6, onde μ_c e σ_c representam a média e o desvio padrão de uma determinada classe e são calculados a partir das amostras no conjunto de treinamento.

$$P(f_i|c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} \cdot \exp\left(-\frac{(f_i - \mu_c)^2}{2\sigma_c^2}\right) \tag{2.6}$$

2.1.2 k-Vizinhos Mais Próximos

k-Vizinhos Mais Próximos (*k-Nearest Neighbors*, kNN) é um método de aprendizado de máquina supervisionado, baseado em instância e com fundamentação em um algoritmo

de aprendizado não-paramétrico (DUDA *et al.*, 2001). Descreve a categoria a que pertence uma amostra de acordo com as características dos k vizinhos mais próximos do conjunto de treinamento. A variável k é o número de vizinhos mais próximos usados para descobrir a qual categoria a amostra proposta pertence.

De forma geral, o processo de inferência do kNN pode ser descrito como um processo de votação majoritária entre as k amostras de treinamento mais próximas da instância invisível, onde o voto de uma determinada amostra de treinamento é o rótulo/classe. Quanto mais próximo, menor é o valor para uma determinada métrica de dissimilaridade. Dentre as mérticas de dissimilaridade estão as distâncias euclidiana, Mahalanobis, Minkowski, Hamming e outras. Uma das métricas de dissimilaridade mais populares é a distância euclidiana, que possui a formulação de acordo com a Equação 2.7, onde $\mathbf{X}$ e $\mathbf{X^i}$ são dois pontos/instâncias arbitrários em um espaço dimensional p e d é uma função de medida de dissimilaridade.

$$d(X,X^i) = \sqrt{(x_1 - x_1^i)^2 + (x_2 - x_2^i)^2 + ... + (x_p - x_p^i)^2} \tag{2.7}$$

O algoritmo de classificação kNN pode ser dividido formalmente em três etapas, assumindo que $\mathbf{X}$ é uma observação invisível e $k \in \mathbb{N}^*$. Na primeira etapa, o kNN calcula a distância entre $\mathbf{X}$ e cada amostra de treinamento, onde as k amostras de treinamento com valores mais baixos de distância pertencem a um conjunto $\mathbf{A}$. Na segunda etapa a probabilidade condicional é estimada para cada classe possível. De forma específica, a probabilidade de uma observação invisível $\mathbf{X}$ pertencer à classe c é dada pela taxa de amostras do subconjunto $\mathbf{A}$ (com k elementos) que possuem um rótulo igual a c. Por fim, a instância invisível é rotulada com a classe mais provável, a de valor mais alto.

A Equação 2.8 formula o processo de estimação descrito para a segunda etapa de classificação do kNN, onde c é uma determinada classe do problema, y_i é um determinado rótulo da amostra $\mathbf{A}$, $\mathbf{A}$ são as amostras de treinamento mais próximas k e I é uma função indicadora que retorna 1 (um), quando suas entradas são iguais, ou 0 (zero), caso contrário.

$$P(c) = \frac{1}{K} \cdot \sum_{i \in A} I(y_i, c) \tag{2.8}$$

2.1.3 Floresta Aleatória

O algoritmo Floresta Aleatória (*Random Florest* - RF) é baseado no Método da Árvore de Decisão (*Decision Trees* - *Decision Trees* (DT)s). Ele introduz o aprendizado supervisionado e visa criar árvores de decisão a partir de uma coleção de características escolhidas

arbitrariamente do conjunto original. O aprendizado é feito com uma abordagem de conjunto chamada *Random Bagging*, um meta-algoritmo que melhora a seleção e regressão do modelo de acordo com a estabilização e precisão do modelo (BREIMAN, 2004).

Cada Árvore de Decisão pode ser vista como uma vizinhança ponderada (*weighted neighborhood*) (LIN; JEON, 2006) sendo expressa de acordo com a Equação 2.9, onde X' é uma observação invisível, $\hat{y}$ é seu rótulo estimado, X_i é a $i-sima$ amostra de treinamento, y_i é seu respectivo rótulo e W é a função de peso.

$$\hat{y} = \sum_{i=1}^{n} W(X_i, X_i^{`}) \, y_i, \tag{2.9}$$

A função de peso é alterada de acordo com o método de classificação utilizado. No contexto de Árvores de Decisão a função de peso é dada pela formulação matemática da Equação 2.10, onde K são pontos de k na mesma folha que X'.

$$W(X_i, X_i^{`}) = \begin{cases} \frac{1}{k} & , \, if \, X_i \in K \\ 0 & , \, otherwise \end{cases}, \tag{2.10}$$

Nota-se que a única diferença entre essa função de peso dada pela Equação 2.10 e a calculada para o método kNN na subseção 2.1.2 é que K é um conjunto composto pelos pontos de k mais próximos da observação invisível. Também, é possível estender a equação 2.10 para compor a predição da Floresta Aleatória, que é dada pela Equação 2.11, onde m é o número de estimadores (árvores) usados e W_j são suas respectivas funções de peso.

$$\hat{y} = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} W_j(X_i, X_i^{'}) \, y_i, \tag{2.11}$$

2.1.4 Perceptron Múltiplas Camadas

Perceptron Múltiplas Camadas (*Multilayer Perceptron*, MLP) é uma estrutura de rede neural multicamadas do tipo supervisionada não-paramétrica. É uma composição Perceptron para lidar com questões não linearmente separáveis (HAYKIN, 2010). Os impulsos se propagam para as camadas subsequentes a partir da camada de entrada, que inicialmente recebe o vetor de características. Ajustados por pesos, os pulsos podem ser transmitidos através dos enlaces neuronais (HAYKIN, 2010).

A Equação 2.12 formula a previsão de Perceptron, onde φ é uma função de ativação (por exemplo, logística/*softmax*), x é um vetor de entrada, p é o número de atributos de uma determinada entrada, w é o vetor de pesos, que combinado com b, são parâmetros aprendíveis do Perceptron.

$$\hat{y} = \varphi\left(\sum_{i=1}^{p} x_i w_i + b\right), \tag{2.12}$$

O processo de treinamento desse classificador é realizado de forma interativa de acordo com a Equação 2.13, em que t é uma determinada etapa do tempo de treinamento, y é o rótulo verdadeiro, $\hat{y}$ é o rótulo previsto e η é a taxa de aprendizado.:

$$w_i(t+1) = w_i(t) + \eta \cdot (y - \hat{y}(t)) \cdot x_i \; \forall \; i = 1, \ldots, p, \tag{2.13}$$

No entanto, o Perceptron não é capaz de executar uma classificação não linear. Para superar esse problema foi proposto o Perceptron Múltiplas Camadas que consiste em um gráfico computacional composto por uma pilha de camadas de perceptron. Cada uma dessas camadas, exceto a última, possui suas funções de ativação logística, as quais são empregadas para criar um classificador linear, substituídas por outra função não-linear, como a função sigmóide, que é usada para executar uma transformação não-linear no espaço de atributos.

A regra de decisão do MLP pode ser matematicamente formulada de acordo com a Equação 2.14, onde x é o vetor de entrada de p dimensões, l é um determinado índice de camada, D é o número de neurônios na camada $l-1$ e φ é a função não-linear.

$$\hat{y}_{l_k} = \begin{cases} \varphi_{0_k}(\sum_{i=1}^{p} x_i \cdot w_{0_{k_i}} + b_{0_k}) & \text{if } \; l = 0 \\ \varphi_{l_k}(\sum_{i=1}^{D} \hat{y}_{l-1_i} \cdot w_{l_{k_i}} + b_{k_l}) & \text{otherwise} \end{cases}, \tag{2.14}$$

Em geral, os parâmetros aprendíveis do MLP, pesos e viés, são inicializados com valores aleatórios próximos de zero. O MLP geralmente utiliza matematicamente um processo iterativo para aumento de desempenho, como formulado da Equação 2.15. Onde η é a taxa de aprendizado e $\delta_{l_k}(t)$ é o gradiente local do $k-simo$ neurônio da camada $l-sima$.

$$\hat{w}_{l_{k_i}}(t+1) = \begin{cases} w_{0_{k_i}}(t) + \eta \cdot \delta_{1_i}(t) \cdot x_k(t) & \text{if } \; l = 0 \\ w_{l_{k_i}}(t) + \eta \cdot \delta_{l+1_i}(t) \cdot \hat{y}_{l_k}(t) & \text{otherwise} \end{cases}, \tag{2.15}$$

O gradiente local é obtido pela Equação 2.16, onde z é a última camada, M é o número de neurônios na última camada (z) e J é o número total de neurônios na camada $l-sima$.

$$\delta_{l_k}(t) = \begin{cases} \varphi'_{l_k}[u_{l_k}(t)] \cdot \frac{1}{2M} \sum_{o=1}^{M} [y_{l_o}(t) - \hat{y}_{l_o}(t)]^2 & \text{if } \; l = z \\ \varphi'_{l_k}[u_{l_k}(t)] \cdot \sum_{j=1}^{J} w_{l_{k_j}}(t) \cdot \delta_{l+1_j}(t) & \text{otherwise} \end{cases}, \tag{2.16}$$

No entanto, observe que a derivada $\varphi'_i[u_{l_i}(t)]$ possui valores diferentes, a deepender da função de ativação escolhida, como mostrado na Equação 2.17:

$$\varphi'_{l_k}[u_{l_k}(t)] = \begin{cases} y_{l_k}(t)[1-y_{l_k}(t)], & if\ logistic/sigmoid \\ \frac{1}{2}[1-(y_{l_k}(t))^2], & if\ hyperbolic\ tangent \end{cases}. \tag{2.17}$$

2.1.5 Máquina de Vetores de Suporte

Com base na Teoria de Aprendizagem Estatística desenvolvida por (VAPNIK, 2000), o classificador Máquina de Vetores de Suporte (Support Vector Machines, SVM) tem como principal objetivo identificar categorias de superfícies que aumentam o comprimento entre elas. Para tanto, ele deve demarcar os modelos espaciais e suas entradas proporcionalmente a um grande vetor de características. Para um melhor entendimento deste classificador é necessário entender os conceitos dos classificadores de hiperplano, de margem máxima e de margem suave.

Incialmente, vamos supor que cada instância de um determinado problema possa ser representada como um ponto **X** em um espaço dimensional p, onde cada dimensão/eixo desse espaço está associado a um recurso desse problema. Dessa forma, o ponto Este ponto **X** pode ser representado matermaticamente pela Equação 2.18.

$$\mathbf{X} = (x_1, x_2, ..., x_p) \tag{2.18}$$

Agora, suponha que exista um hiperplano dimensional $p-1$ que separe perfeitamente um conjunto de instâncias de um determinado problema por seus rótulos. Assim, podemos representá-lo matematicamente de acordo com a Equação 2.19, onde β são os coeficientes do hiperplano, $\alpha_1, ..., \alpha_p \in P_O$ e P_O é dado um ponto cruzado pelo hiperplano.

$$\beta_0 + \beta_1\alpha_1 + \beta_2\alpha_2 + ... + \beta_p\alpha_p = 0. \tag{2.19}$$

Portanto, de acordo com a Equação 2.20, é possível definir uma função de indicador que use o hiperplano como limite de acordo com a posição relativa de um determinado ponto **X** para um determinado hiperplano.

$$y = \begin{cases} +1, & \text{if } \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p > 0 \\ -1, & \text{if } \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p < 0 \end{cases}. \tag{2.20}$$

onde os pontos do lado positivo do hiperplano estão associados a um rótulo positivo ($y = +1$) e o oposto também é válido.

O Classificador de Hiperplano utiliza a equação 2.20 como um processo de estimativa onde os pontos são amostras invisíveis e a posição relativa ao hiperplano define a qual classe eles

pertencem. No entanto, se um conjunto de amostras for linearmente separável, poderá existir um número infinito de hiperplanos. Visando resolver esse problema selecionando os melhores hiperplanos dentre os existentes, foi proposto um método chamado Classificador de Margem Máxima.

O hiperplano de margem máxima, consiste no hiperplano mais distante (a soma com o maior valor da distância perpendicular de todas as amostras de treinamento a um determinado hiperplano) de todas as observações de treinamento. Enquanto isso, a menor dessas distâncias perpendiculares é conhecia como margem. A busca pelo melhor hiperplano pode ser entendida como um problema de maximização podendo ser formulada de acordo com as Equações 2.21 e 2.22, onde M é o comprimento da margem, n é o número de amostras de treinamento, $X_1, ..., X_n \in R_p$ e rótulos de classe $y_1, ..., y_n \in \{-1, +1\}$.

$$\underset{\beta_1,...,\beta_p}{maximize} M, \tag{2.21}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M, \forall i = 1, ..., n. \tag{2.22}$$

A aplicação deste hiperplano ideal à Equação 2.20 cria o Classificador de Margem Máxima. Contudo, esse classificador é sensível a observações individuais. Por esse motivo, um novo método foi proposto, o Classificador de Margem Suave. Esse novo método adiciona uma variável de folga ε para cada observação no conjunto de treinamento, o que permite que algumas amostras sejam classificadas incorretamente caso haja um aumento da margem do classificador.

O processo de otimização para encontrar o melhor Classificador de Margem Suave pode ser modelado como um problema de otimização de acordo com as Equações 2.23, 2.24 e 2.25, onde C é um hiperparâmetro maior que zero.

$$\underset{\beta_1,...,\beta_p,\varepsilon_1,...,\varepsilon_n}{maximize} M, \tag{2.23}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}) \geq M(1 - \varepsilon_i)), \tag{2.24}$$

$$\varepsilon_i \geq 0, \sum_{i=1}^{n} \varepsilon_i \leq C, \tag{2.25}$$

Diante do exposto acima, a regra de descisão do classificador SVM pode ser matematicamente formulada de acordo com a Equação 2.26, onde α é um coeficiente do SVM aprendido durante o treinamento e K é a transformação do espaço de destino.

$$f(X) = \beta_0 + \sum_{i=1}^{n} \alpha_i K(X^T, X^{'}), \tag{2.26}$$

Dessa forma, um *kernel* linear (transformação) pode ser expresso de acordo com a Equação 2.27, enquanto a transformação radial é definida pela Equação 2.28:

$$K(X,X^{'}) = \sum_{j=1}^{p} X_j^T, X_j^{'}, \tag{2.27}$$

$$K(X,X^{`}) = \exp(-\gamma \sum_{j=1}^{p} (X_j^T, X_j^{`})^2). \tag{2.28}$$

2.2 *Rede Neural Convolucional*

A Rede Neural Convolucional (Convolutional Neural Network, CNN) é um tipo de rede neural artificial amplamente empregada no campo do aprendizado profundo. Seu algoritmo captura dados de entrada e atribui importância às características, estabelecendo pesos correspondentes a essas características. Os métodos, operações e funções de ativação empregados na implementação, treinamento e validação das CNNs são detalhados nesta seção com o intuito de auxiliar na compreensão do funcionamento desses tipos de redes.

Desenvolvida originalmente por (LECUN *et al.*, 1989), as CNNs se tornaram conhecidas quando (KRIZHEVSKY *et al.*, 2012) usou o algoritmo no ImageNet 2012 *LargeScale Visual Recognition Challenge* (*ImageNet Large Scale Visual Recognition Challenge* (ILSVRC)-2012) e atingindo bons resultados. O desafio no ILSVRC-2012 era buscar soluções na tarefas de classificação num contexto de 1000 classes e mais de 1 milhão de amostras (KRIZHEVSKY *et al.*, 2012; DENG *et al.*, 2009).

A partir dos resultados obtidos pelo (KRIZHEVSKY *et al.*, 2012), outras arquiteturas de CNNs foram desenvolvidas a fim de melhorar os resultados obtidos. Entre as arquiteturas que surgiram estão a VGG (**??**), MobileNet (HOWARD *et al.*, 2017), Inception (SZEGEDY *et al.*, 2015), Xception (CHOLLET, 2017), NasNet (ZOPH; LE, 2017), ResNet (HE *et al.*, 2016), DenseNet (HUANG *et al.*, 2017) e Inception-Resnet (SZEGEDY *et al.*, 2016).

Em uma Rede Neural Convolucional, a convolução é uma operação linear que envolve a multiplicação de um conjunto de pesos com a entrada, semelhante a uma rede neural convencional (LECUN *et al.*, 1989). Essa técnica foi desenvolvida para lidar com dados de entrada bidimensionais, onde a multiplicação ocorre entre uma matriz de dados de entrada e uma matriz bidimensional de pesos, também conhecida como filtro ou *kernel* (LECUN *et al.*, 1989). Através do processo de convolução e operando em campos receptivos locais, a CNN preserva as correlações entre os *pixels* vizinhos da imagem, reduzindo assim a sensibilidade a translações,

rotações e distorções nas imagens (LECUN *et al.*, 1998). Por outro lado, outros tipos de redes neurais não conseguem manter essas relações, pois consideram as imagens como uma entrada unidimensional.

2.2.1 Estrutura

Segundo LeCun *et al.* (1998), as Redes Neurais Convolucionais (CNNs) são arquiteturas de redes neurais *feedforward* projetadas para garantir a invariância da rede em relação a translações, rotações e distorções na imagem de entrada. Elas são compostas por conexões locais, compartilhamento de pesos e diferentes camadas, como a camada convolutiva responsável pela extração de características da imagem, a camada de *pooling* que realiza sub-amostragem da imagem e camadas totalmente conectadas encarregadas de interpretar as características extraídas (LECUN *et al.*, 2015). Essa organização em camadas com funções distintas permite que as CNNs realizem a análise e o processamento eficiente de imagens.

2.2.1.1 Camada Convolucional

Na camada de convolução de uma rede neural, as unidades são organizadas em *feature maps*, sendo cada unidade conectada a uma região da camada anterior por meio de um conjunto de pesos conhecidos como filtros ou *kernel* (LECUN *et al.*, 2015). O uso de pesos compartilhados é empregado, o que significa que várias conexões são manipuladas pelo mesmo parâmetro (peso) (LECUN, 1989), o que reduz significativamente a quantidade de parâmetros da rede. Esse arranjo dos pesos nas conexões permite que o processo de convolução ocorra, permitindo a extração de características específicas das imagens, como linhas e contornos. Os filtros podem ser bidimensionais ou tridimensionais e são aplicados em partes da imagem denominadas campos receptivos locais. Nesse contexto, os filtros podem ser visualizados como pequenos quadrados que se deslocam pela matriz de *pixels* da imagem por meio de um valor de deslocamento chamado *stride*.

A Equação 2.29 formula o cálculo do tamanho do *feature map*, onde T_m representa o tamanho da *feature map*, M é a dimensão da imagem, k é o tamanho do filtro (*kernel*) e s consiste no passo do deslocamento (*stride*).

$$T_m = \frac{M-k}{s} + 1 \tag{2.29}$$

As variáveis T_m, M e k correspondem ao tamanho em uma das dimensões (horizontal

ou vertical). No entanto, o uso de imagens e filtros quadrados, com altura e largura iguais, é um padrão adotado.

Para garantir o poder de uma rede neural, é necessário incorporar não-linearidade, e nas Redes Neurais Convolucionais (CNNs), a função *Rectified Linear Unit (ReLU)* (Rectified Linear Unit) é comumente utilizada após a operação de convolução para introduzir essa não-linearidade. Assim, os valores finais dos *feature maps* são obtidos aplicando-se a função *ReLU* aos resultados das convoluções. A adoção generalizada da função ReLU foi motivada por experimentos realizados por Nair e Hinton (2010), que demonstraram que o uso de ReLUs acelera significativamente a convergência do treinamento em comparação com outras funções de não-linearidade, como a tangente logística ou a hiperbólica. Matematicamente, a função ReLU é definida de acordo com a Equação 2.30.

$$g(z) = max(0, z) \tag{2.30}$$

2.2.1.2 Camada de Pooling

A fim de reduzir a saída da camada convolucional sem adicionar parâmetros extras ao modelo, é comum utilizar a camada de *pooling*, uma vez que a camada convolucional não diminui o tamanho da entrada. A principal função dessa camada é atingir um certo nível de invariância ao deslocamento e reduzir a resolução dos *feature maps* (GU *et al.*, 2015). Ela opera de forma similar à camada convolucional, mas o *stride* geralmente tem o mesmo tamanho do filtro, ao contrário da camada convolucional que utiliza *stride* igual a 1. Isso resulta em campos receptivos completamente diferentes e aproximadamente 75% de redução nos *feature maps*. Os tipos mais comuns de *pooling* são: O *max pooling*, que seleciona o valor máximo do campo receptivo; e o *average pooling*, que calcula a média dos valores (BENGIO *et al.*, 2015).

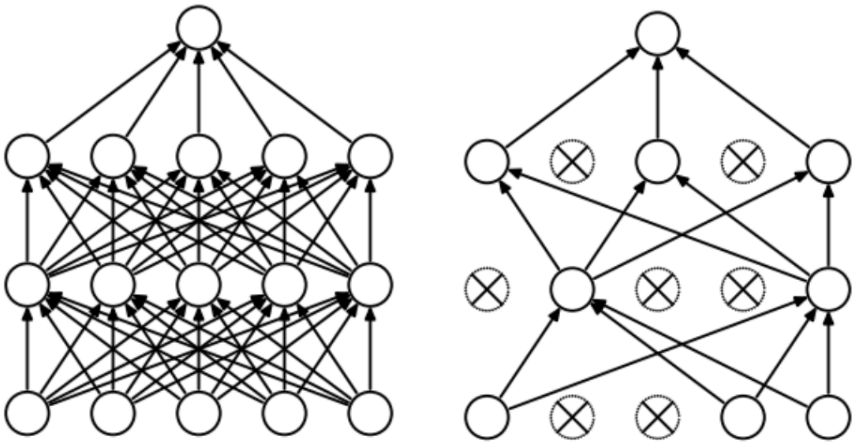
2.2.1.3 Dropout

Para evitar o *overfitting* (ou sobreajuste) em uma rede neural, é comum utilizar a técnica de *Dropout*. Essa técnica consiste em desativar aleatoriamente algumas unidades da rede durante o treinamento, como ilustrado na Figura 1, a fim de melhorar o desempenho da rede em imagens fora do conjunto de treinamento. Em 1 (a), a ilustração representa o fluxo de informações convencionais durante uma passagem direta de uma rede e, em 1 (b), esse fluxo é interrompido (definido como zero) em algum neurônio, consistindo no processo chamado de

Dropout.

Os neurônios desativados têm seus valores definidos como zero e não contribuem para a propagação do sinal na rede, nem têm seus pesos atualizados durante a retropropagação do gradiente. Uma probabilidade p é estabelecida, determinando a chance de cada unidade ter seu sinal propagado pela rede. Essa técnica de *Dropout* ajuda a regularizar a rede neural e a evitar o sobreajuste, melhorando sua generalização. (SRIVASTAVA *et al.*, 2014).

Figura 1 – Fluxo de informações em uma rede neural.



(a) Módulo simples de criação. (b) Rede Neural Padrão.

Fonte: (SRIVASTAVA *et al.*, 2014).

2.2.1.4 Camada Totalmente Conectada

A imagem de entrada passa por camadas convolutivas e de *pooling*, que identificam desde características simples até as mais complexas, culminando nas camadas totalmente conectadas. Normalmente, a camada precedente é uma camada de *pooling*, onde os *feature maps* possuem múltiplas dimensões e são convertidos em um vetor unidimensional para conectar-se à parte final da rede. As camadas totalmente conectadas atuam como uma rede neural *feedforward* multicamadas e são responsáveis por interpretar as características extraídas pelas camadas anteriores (LECUN *et al.*, 1998).

2.2.1.5 Softmax

A última camada da rede neural é responsável por fornecer os resultados da classificação. O número de neurônios nessa camada é determinado pela quantidade de classes do problema que esta sendo analisado. Cada neurônio produz uma saída que representa a probabilidade da

classe correspondente, variando de 0 a 1. Assim, a resposta final é obtida selecionando-se o neurônio com o maior valor na saída. Esse processo é realizado utilizando a função *Softmax*, que é formulada na Equação 2.31, onde a saída y do neurônio j passa a ser o valor dessa mesma saída dividido pelo somatório de todas as saídas dos neurônios dessa camada. A função *Softmax* normaliza as saídas dos neurônios para que somem 1, produzindo uma distribuição de probabilidades entre as classes.

$$y_j = \frac{e^{y(j)}}{\sum_{i=1}^{n} e^{y(i)}} \tag{2.31}$$

2.2.2 Arquiteturas

2.2.2.1 Arquiteturas VGG

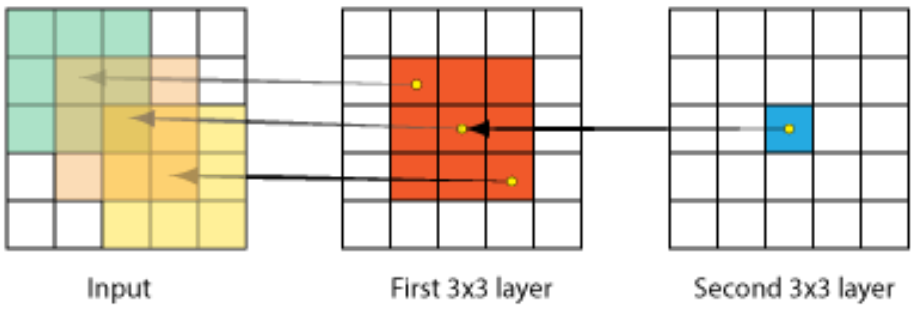
As experiências realizadas no artigo do Oxford Visual Geometry Group (VGG) (SIMONYAN; ZISSERMAN, 2014) revelaram uma correlação positiva entre o aumento da profundidade (número de camadas) e a precisão dos modelos no conjunto de dados *ImageNet*. Essa relação foi observada em todas as arquiteturas avaliadas (11, 13, 16 ou 19 camadas). Sendo que o aumento na profundidade dos modelos não resultou em superajuste, e isso foi possível graças à aplicação de uma estratégia de regularização conhecida como Convoluções Fatoradas.

Essa abordagem trouxe três aspectos importantes para justificar o domínio da VGG sobre os modelos anteriores. Primeiro, em uma convolução, cada ponto no mapa de ativação é influenciado apenas por uma sub-região conectada da entrada, ou seja, uma convolução 3×3 é afetada por uma janela de entrada 3×3. Em seguida, se uma segunda convolução 3×3 for aplicada ao primeiro mapa de ativação, cada ponto no novo mapa será influenciado por uma janela 3×3 do mapa anterior. No entanto, como cada ponto no mapa anterior combina informações de uma janela 3×3 da entrada, uma combinação de pontos 3×3 desse mapa será influenciada por uma janela 5×5 da entrada. Portanto, se esse processo for repetido mais uma vez, cada ponto de saída na pilha de convoluções 3×3 de três camadas será influenciado por uma sub-região de 7×7 da entrada. Em resumo, uma pilha de convoluções 3×3 de três camadas possui o mesmo campo receptivo efetivo de uma única convolução 7×7.

Um exemplo desse processo de equivalência pode ser visto na Figura 2.

Segundo, uma pilha de convolução de 3×3 de três camadas possui menos parâmetros que uma única camada convolucional de 7×7. E por fim, com cada camada possuindo sua própria não-linearidade, uma pilha de camadas convolucionais tem um poder discriminativo

Figura 2 – O campo receptivo efetivo de uma pilha de duas convoluções 3 × 3.



Fonte: (SIMONYAN; ZISSERMAN, 2014).

maior do que uma única camada. Essa característica contribuiu para a popularidade das redes VGG, não apenas devido aos seus bons resultados no ILSVRC 2014 (RUSSAKOVSKY *et al.*, 2015), mas também por sua arquitetura uniforme e consistente.

As operações fundamentais dessas arquiteturas são a *convolução* e o *max pooling*. A operação de convolução pode ser representada de acordo com a Equação 2.32, onde Y é a saída, X é a entrada, W é o filtro da convolução, e os índices $[i, j]$ percorrem as dimensões espaciais da entrada.

$$Y[i,j] = sum(sum(X[i-k : i+k, j-l : j+l] * W[k,l])) \tag{2.32}$$

Enquanto a operação de *max pooling* pode ser representada de acordo com a Equação 2.33, onde, novamente, Y é a saída, X é a entrada, e os índices [i, j] percorrem as dimensões espaciais da entrada..

$$Y[i,j] = max(X[2*i : 2*i+2, 2*j : 2*j+2]) \tag{2.33}$$

As principais contribuições das arquiteturas VGG incluem a demonstração de que redes mais profundas podem alcançar melhor desempenho, a introdução da ideia de arquiteturas "muito profundas", e o uso de convoluções de 3x3 em toda a rede. No entanto, a principal desvantagem dessas redes é o seu alto custo computacional e o grande número de parâmetros, o que leva a um uso intensivo de memória (CANZIANI *et al.*, 2017).

2.2.2.2 Arquitetuta MobileNet

Eficiente em termos computacionais, *MobileNet* é uma arquitetura de rede neural convolucional projetada especificamente para dispositivos móveis e aplicações embarcadas. Essa

arquitetura foi introduzida por Howard *et al.* (2017) e é caracterizada por duas técnicas principais: Convoluções separáveis em profundidade e multiplicadores de largura e resolução.

As convoluções separáveis em profundidade são uma forma eficiente de realizar convoluções. Elas dividem a operação de convolução em duas partes: (i) uma convolução em profundidade que aplica um único filtro a cada canal de entrada, seguida por (ii) uma convolução $1x1$ (convolução de ponto) que é responsável por construir novos recursos através da combinação de recursos de entrada. Esse procedimento, computacionalmente, é muito menos intensivo do que realizar uma convolução completa em cada passo.

Os multiplicadores de largura e resolução são um meio de ajustar a complexidade da rede. Eles permitem que ela seja mais larga ou mais fina, ou que a resolução de entrada seja aumentada ou diminuída. Isso torna a MobileNet uma rede muito flexível que pode ser adaptada a diferentes requisitos de recursos e desempenho. As equações para as convoluções separáveis em profundidade podem ser descritas matematicamente conforme as Equações 2.34 (convolução em profundidade) e 2.35 (convolução de ponto), oOnde Y é a saída, X é a entrada, W é o filtro da convolução, e os índices $[i, j]$ percorrem as dimensões espaciais da entrada, enquanto k percorre a dimensão em profundidade.

$$Y[i,j,k] = X[i-k1 : i+k1, j-k2 : j+k2, k] * W[k1,k2,k] \tag{2.34}$$

$$Y[i,j,k] = X[i,j] * W[k] \tag{2.35}$$

2.2.2.3 Arquiteturas Inception

O *Inception*, também conhecido como *GoogleNet*, obteve resultados de destaque na tarefa de detecção do ILSVRC 2014, alcançando uma taxa de erro de apenas 6,67% entre os 5 melhores classificados (RUSSAKOVSKY *et al.*, 2015). Além disso, as contribuições dos modelos Inception para o campo da pesquisa em Aprendizado Profundo incluem: O módulo Inception, a técnica de *Batch-Normalization* e as Convoluções Assimétricas.

Desde a introdução original da *GoogLeNet*, existiram várias versões subsequentes dessa arquitetura, cada uma com melhorias e modificações. Por exemplo, a Inception V2 introduziu a normalização em lote, a Inception V3 introduziu fatores de escala e convoluções assimétricas, e a Inception V4 introduziu conexões de atalho semelhante a arquitetura *ResNet*.

2.2.2.3.1 Módulo Inception

O projeto de uma arquitetura de Rede Neural Convolucional envolve a seleção das operações a serem utilizadas em cada camada, como convoluções de diferentes tamanhos (1×1, 3×3, 5×5) ou *pooling*. O Módulo *Inception* simples executa todas essas operações de uma só vez, de forma paralela, e depois realiza a concateneação das saídas. No entanto, depois de algumas camadas dessa forma, a quantidade de parâmetros adicionados à rede pode se tornar excessiva durante o treinamento, trazendo uma desvantagem para esse tipo de aplicação.

Entretanto, para lidar com a ineficiência decorrente do aumento de parâmetros, o *Google Brain* introduziu a convolução 1×1 para comprimir as entradas das camadas convolucionais e as saídas das camadas de *pooling*. Isto é conhecido como uma "bottleneck layer". Essa simples adaptação preservou a maior parte da precisão da abordagem paralela original, ao mesmo tempo em que reduziu significativamente o número de parâmetros livres necessários.

2.2.2.3.2 Camada *batch-normalization*

O Deslocamento Covariado Interno (*Internal Covariate Shift*), que é uma alteração na distribuição das entradas de camadas intermediárias durante o treinamento, é um dos problemas mais desafiadores na otimização de redes profundas (Ioffe; Szegedy, 2015). Isso ocorre devido à influência dos pesos das camadas anteriores nas entradas de uma determinada camada, que geralmente mudam durante a otimização. Essa mudança constante na distribuição de entrada pode resultar em uma convergência mais lenta do modelo. Para resolver esse problema, Ioffe e Szegedy (2015) propôs a técnica de normalização em lote (*batch normalization*), que incorpora a normalização das ativações diretamente na arquitetura da rede.

Com apenas dois parâmetros extras por ativação, o *batch-normalization* normaliza as saídas do produto interno entre pesos e entradas de uma determinada camada, com base em sua média e desvio padrão. Considerando γ e β os parâmetros a serem aprendidos no treinamento, a saída é calculada de acordo com a Equação 2.36, onde $\hat{x}$ representa a entrada normalizada..

$$y = \gamma\hat{x} + \beta, \tag{2.36}$$

O cálculo de $\hat{x}$ é realizado a partir da Equação 2.39, em um *mini-batch* $M = \{x_1, x_2, ..., x_b\}$, sendo b o seu tamanho. A constante λ tem como objetivo proporcionar estabilidade numérica e é necessário inicialmente calcular a média e a variância do *mini-batch*, através das Equações 2.37 e 2.38, respectivamente.

$$\mu_M \leftarrow \frac{1}{b}\sum_{i=1}^{b} x_i, \tag{2.37}$$

$$\sigma_M^2 \leftarrow \frac{1}{b}\sum_{i=1}^{b}(x_i - \mu_M)^2, \tag{2.38}$$

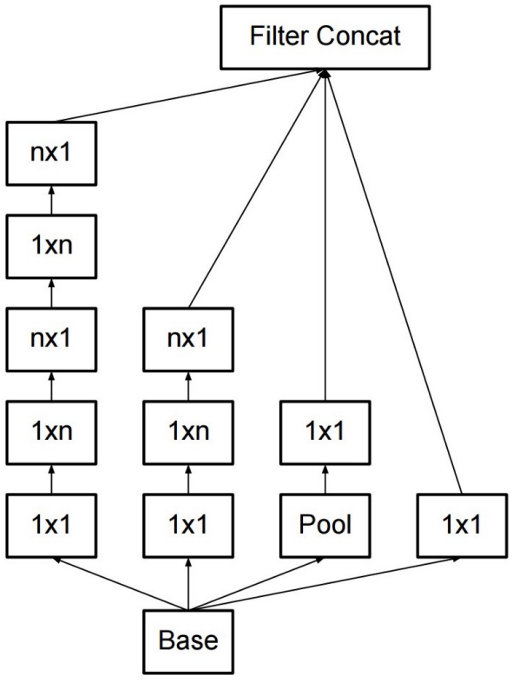
$$\hat{x}_i = \frac{x_i - \mu_M}{\sqrt{\sigma_M^2 + \lambda}}, \tag{2.39}$$

2.2.2.3.3 Convoluções Assimétricas

A equipe do *Google Brain* propôs uma nova versão dos Módulos *Inception*, baseada nos conceitos de Convoluções Fatoradas e *Flattened* CNNs, estendendo as ideias apresentadas nos artigos VGG (SIMONYAN; ZISSERMAN, 2014) e *Flattened* CNNs (JIN *et al.*, 2014). Para melhorar a eficiência dos parâmetros, eles introduziram Convoluções Assimétricas nos Módulos *Inception*, fatorizando convoluções de tamanho $n \times n$ em duas convoluções assimétricas: $n \times 1$ seguido de $1 \times n$.

A Figura 3 ilustra uma arquitetura genérica de um Módulo *Inception* com Convoluções Assimétricas. Essa abordagem visa melhorar a eficiência e o desempenho dos Módulos *Inception* nas redes neurais.

Figura 3 – Arquitetura do Módulo Inception com convoluções assimétricas.



Fonte: (SZEGEDY *et al.*, 2016).

2.2.2.4 Arquiteturas Xception

Introduzida por Chollet (2017) a arquitetura *Xception (Extreme Inception)* é uma extensão da arquitetura *Inception* com a substituição dos módulos *Inception* tradicionais por convoluções separáveis em profundidade.

Na arquitetura *Inception*, a operação de convolução é realizada com diferentes tamanhos de filtro e as saídas são concatenadas. Já a arquitetura *Xception* adota um módulo de convolução separável em profundidade, que aplica primeiro uma convolução 1x1 para transformar o espaço de entrada (*cross-channel*), e depois uma convolução em profundidade para uma transformação espacial. Este procedimento é baseado na hipótese de que as correlações espaciais e *cross-channel* nas características de entrada podem ser mapeadas completamente separadas.

Essa arquitetura é composta por:

- Camada de entrada de 299x299x3;
- Camadas convolucionais iniciais com passo 2 para reduzir o tamanho das características de entrada;
- Módulos de convolução separáveis em profundidade repetidos 8 vezes;
- Camadas convolucionais finais para produzir o vetor de características;
- Camadas totalmente conectadas para a tarefa final de classificação.

Dentre as contribuições da arquitetura *Xception* têm-se a introdução de convoluções separáveis em profundidade em larga escala e a proposta de uma nova hipótese sobre a separabilidade das correlações de características. Além disso, apesar de ter aproximadamente o mesmo número de parâmetros, a arquitetura *Xception* demonstrou desempenho superior ao *Inception V3* no conjunto de dados *ImageNet*.

2.2.2.5 Arquiteturas ResNet

A *ResNet*, ou Rede Residual, foi introduzida por (HE *et al.*, 2016) e projetada para resolver o problema do desaparecimento do gradiente, que ocorre quando as redes neurais se tornam muito profundas. A ideia principal é introduzir "conexões residuais"(ou "conexões de atalho") que permitem que o gradiente seja diretamente retropropagado a camadas mais baixas.

Uma "conexão de atalho"na ResNet permite que a entrada de um conjunto de camadas seja adicionada à saída dessas camadas. Matematicamente, se $H(x)$ é a saída desejada de um conjunto de camadas e $F(x) = H(x) - x$ é a transformação realizada por essas camadas,

então a saída real dessas camadas é $F(x)+x$. Isso é chamado de "bloco residual"e é a unidade de construção da arquitetura *ResNet*.

As "conexões de atalho"permitem que a rede aprenda a transformação identidade quando $F(x)=0$. Por ser uma transformação de fácil aprendizado, facilita o treinamento de redes muito profundas, uma vez que as camadas podem aprender a identidade até que encontrem um bom conjunto de pesos. Além disso, a retropropagação do gradiente através da conexão de atalho permite que o sinal do gradiente chegue diretamente a camadas mais baixas sem atenuação.

Desde a introdução da *ResNet*, foram propostas várias variantes e extensões, como a *ResNeXt* (XIE *et al.*, 2017), que introduz "caminhos"ou "ramificações"dentro dos blocos residuais, e a *ResNet-D* (HE *et al.*, 2018), que modifica a *ResNet* para melhorar seu desempenho e eficiência.

A equação para a saída de um bloco residual pode ser formulada de acordo com a Equação 2.40, onde Y é a saída, x é a entrada, Wi são os pesos das camadas dentro do bloco, e F é a transformação aprendida pelo bloco..

$$Y = F(x, Wi) + x \tag{2.40}$$

2.2.2.6 Arquiteturas Inceptiom-ResNet

A arquitetura *Inception-ResNet*, como o nome sugere, é uma combinação das arquiteturas *ResNet* e *Inception*. Introduzida por (SZEGEDY *et al.*, 2017), essa arquitetura incorpora a ideia de conexões residuais da *ResNet* dentro dos módulos de *Inception*.

O conceito da arquiteturau *Inception-ResNet* é simples. Ela substitui os módulos *Inception* convencionais na *Inception* v3 e v4 por blocos de *Inception* modificados com uma conexão de atalho, similar à *ResNet*. Dessa forma, ela herda os benefícios das conexões residuais, como a capacidade de treinar redes muito profundas, e os benefícios dos módulos *Inception*, como a capacidade de aprender diferentes tamanhos de filtro em cada camada.

As equações que governam a *Inception-ResNet* são uma combinação das equações da *Inception* e da *ResNet*. Dessa forma, cada módulo *Inception* modificado pode ser considerado uma função F, e a conexão de atalho adiciona a entrada x formando a saída $Y = F(x, Wi) + x$.

2.2.2.7 Arquiteturas NASNet

Desenvolvida por Zoph *et al.* (2018), a arquitetura*NASNet* faz uso da Pesquisa em Arquitetura Neural (*Neural Architecture Search, Neural Architecture Search (NAS)*) (ZOPH; LE, 2017) para determinar de forma automatizada a arquitetura de uma rede neural. Ela é projetada para aprender a melhor estrutura com base nos dados de treinamento, ao invés de definir manualmente a estrutura e a disposição das camadas convolucionais, camadas totalmente conectadas e outras operações na rede.

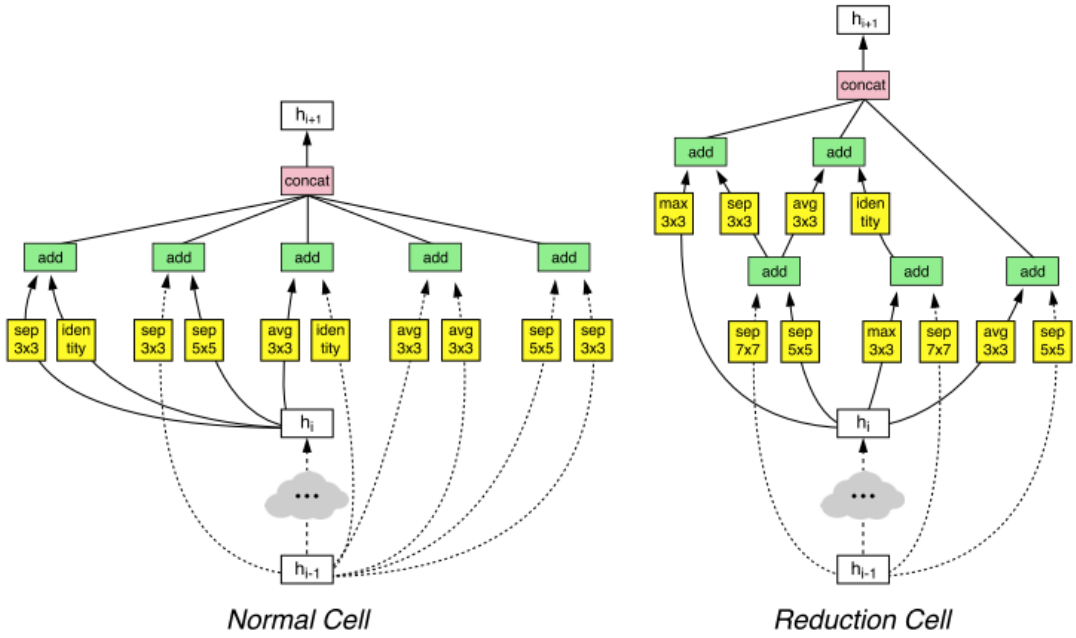
O método NAS faz uso de um controlador recorrente, que é treinado com um algoritmo de aprendizado por reforço. A tarefa do controlador é gerar uma descrição da arquitetura de uma rede neural, ou seja, uma *string* que especifica a sequência de operações a serem realizadas, como por exemplo convoluções ou ativações de ReLU, e como essas operações devem ser conectadas. A rede proposta é então treinada, e seu desempenho informa a próxima iteração do algoritmo de aprendizado por reforço.

A arquitetura NASNet é uma família de arquiteturas que foram descobertas através desse processo de aprendizado por reforço. Além disso, o método NAS é transferível, ou seja, a arquitetura aprendida em um conjunto de dados pode ser transferida para outro conjunto de dados com sucesso. As células convolucionais específicas descobertas pelo processo NAS para a NASNet incluem várias operações de convolução (como por exemplo, convoluções separáveis em profundidade de tamanho $5x5$ e $7x7$), combinações de convoluções e operações de pooling.

As principais contribuições da arquitetura NASNet são a introdução do conceito de Pesquisa em Arquitetura Neural, a demonstração de que as arquiteturas encontradas por esse processo podem superar as arquiteturas manuais de ponta e a prova de que essas arquiteturas podem ser transferidas com sucesso para outros conjuntos de dados. A Figura 4 apresenta uma ilustração dos módulos de precisão encontrados no espaço de pesquisa da *NASNet* dentro do *ImamgeNet*.

Na Figura 4, os blocos foram identificados com *Canadian Institute for Advanced Research, 10 classes* (CIFAR-10). A entrada (branca) é a estado oculto de ativações anteriores (ou imagem de entrada). A saída (rosa) é o resultado de uma operação de concatenação em todos os galhos. Cada célula convolucional é o resultado de blocos B. Um único bloco corresponde a duas operações primitivas (amarelo) e uma operação de combinação (verde).

Figura 4 – Arquitetura das melhores células convolucionais com B = 5.



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

2.2.2.8 Arquiteturas DenseNet

A *Densely Connected Convolutional Networks* (DenseNet) *(Densely Connected Convolutional Networks)* é uma arquitetura de rede neural convolucional introduzida por Huang *et al.* (2017). Ela é uma extensão da ideia das arquiteturas *ResNets*, no entanto, ao invés de adicionar entradas de camadas anteriores, a *DenseNet* concatena as entradas. Cada camada tem acesso direto às saídas de todas as camadas anteriores passando sua própria saída para todas as camadas subsequentes. Essa fato leva a uma conectividade mais densa entre as camadas.

Dentre as vantagens dessa arquitetura temos a redução do desbotamento do gradiente, o reforço da propagação de características e a promoção de recursos de aprendizagem mais eficazes. Uma contribuição importante do artigo *DenseNet* (HUANG *et al.*, 2017) é a ideia de "conexões densas". Ao contrário de outras redes convolucionais, que apenas transmitem informações diretamente de uma camada para a próxima, a DenseNet tem conexões de cada camada para todas as camadas subsequentes.

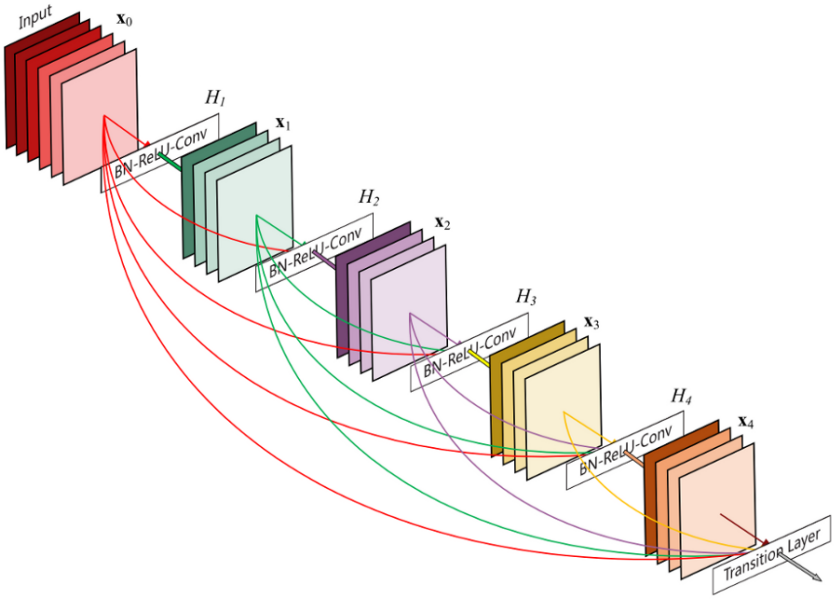
Considerando L como o número total de camadas na rede, a l-ésima camada recebe as características de todas as camadas anteriores, $0, 1, ..., l_1$, como entrada. Se denotarmos a operação não-linear executada na l-ésima camada como $H_l(.)$, a l-ésima camada em uma DenseNet é definida de acordo com a Equação 2.41, $[x_0, x_1, ..., x_{l-1}]$ denota a concatenação das

saídas da camada l.

A Figura 5 apresenta um bloco denso de 5 camadas com uma taxa de crescimento de k = 4. Cada camada recebe todos os mapas de recursos anteriores como entrada.

$$x_l = H_l([x_0, x_1, ..., x_{l-1}]) \tag{2.41}$$

Figura 5 – Exemplo de um bloco denso.



Fonte: (HUANG *et al.*, 2017).

2.2.3 Uso como extrator de características

Escolher um banco de dados para treinar uma Rede Neural Convolucional (CNN) de forma satisfatória pode se tornar uma tarefa bastante problemática dependendo da aplicação desejada (KARPATHY *et al.*, 2014; MOLCHANOV *et al.*, 2016). No entanto, esse impasse pode ser resolvido usando o conceito de *Transfer Learning* (KARPATHY *et al.*, 2014; MOLCHANOV *et al.*, 2016; ALEXANDRE, 2014). Ele garante que o conhecimento adquirido durante uma tarefa de solução de problemas possa ser reutilizado para resolver um problema semelhante (PAN; YANG, 2010; WEISS *et al.*, 2016). O uso do *Transfer Learning* não requer dados de treinamento em grande escala (PAN; YANG, 2010) e o tempo de treinamento para obter uma precisão equivalente pode ser reduzido (TANG *et al.*, 2019).

De forma concisa, para aproveitar o poder de uma CNN pré-treinada deve-se seguir dois passos: i) deve-se eliminar a última camada totalmente conectada do modelo e então ii) deve-

se redimensionar a entrada da camada descartada para um vetor unidimensional (KARPATHY *et al.*, 2014; MOLCHANOV *et al.*, 2016; ALEXANDRE, 2014). Dessa forma, o modelo pré-treinado é utilizado como um extrator de características e deixa de ser considerado um classificador.

Esta estratégia tem como vantagem necessitar de menos dados de treinamento e ser mais computacionalmente eficiente, uma vez que apenas as camadas finais (geralmente muito menores) precisam ser treinadas. No entanto, ela assume que as características extraídas pela rede pré-treinada são suficientemente boas para a nova tarefa, o que pode não ser verdade para alguns casos.

2.3 *Estimativa de profundidade monocular baseada em aprendizado profundo*

2.3.1 Estimativa de profundidade monocular

A estimativa de profundidade é uma tarefa tradicional de visão computacional que prevê a profundidade de uma ou mais imagens bidimensionais (2D). Ela estima a profundidade de cada pixel em uma imagem usando modelos treinados offline. Na percepção da máquina, o reconhecimento de alguns fatores funcionais, como a forma de uma cena a partir de uma imagem e a independência da imagem em relação à sua aparência é fundamental (GODARD *et al.*, 2017; LIU *et al.*, 2016; EIGEN *et al.*, 2014). Por isso, a estimativa de profundidade tem grande potencial para uso em aplicações distintas, incluindo a navegação e localização de robôs.

A tarefa de estimativa de profundidade por visão precisa de uma imagem RGB e uma imagem de profundidade como saída. A imagem de profundidade geralmente consiste em dados sobre a distância do objeto na imagem do ponto de vista da câmera (ZHOU *et al.*, 2017). A abordagem baseada em computador tem sido avaliada por vários investigadores em todo o mundo, e o problema tem sido um campo de pesquisa interessante.

Antes do advento do aprendizado profundo, os métodos tradicionais de estimativa de profundidade monocular se baseavam em técnicas geométricas, como estéreo correspondência, estrutura a partir do movimento e reconhecimento de objetos. Esses métodos utilizavam informações de múltiplas câmeras, movimento ou características visuais para inferir a profundidade em uma imagem (SCHARSTEIN *et al.*, 2014). No entanto, esses métodos eram limitados em termos de precisão e robustez em diversas condições de iluminação, texturas e oclusões.

Com o progresso dos modelos recentes de aprendizado profundo, a estimativa de

profundidade baseada em modelos aprendizado profundo conseguiu se demonstrar eficiênte em diversas aplicações (KHAN *et al.*, 2020; TOSI *et al.*, 2019; RAMAMONJISOA; LEPETIT, 2019). O recente aumento no uso de redes neurais convolucionais (CNN) e redes neurais convolucionais recorrentes (*Recurrent Neural Network* (RNN)) produziu uma melhoria considerável no desempenho dos procedimentos estimativa de profundidade monocular (KUZNIETSOV *et al.*, 2017; BAZRAFKAN *et al.*, 2018; FU *et al.*, 2018). As abordagens baseadas em redes neurauis convolucionais utilizam a capacidade das CNNs de aprender representações hierárquicas de características visuais para inferir a profundidade em uma única imagem. Diversas arquiteturas de CNNs têm sido propostas, incluindo estruturas com encoders e decoders, redes de reconstrução 3D e redes com atenção estruturada (EIGEN *et al.*, 2014; XU *et al.*, 2018; ALHASHIM; WONKA, 2018; GODARD *et al.*, 2019; RANFTL *et al.*, 2021a; LEE *et al.*, 2019; BHAT *et al.*, 2021; KIM *et al.*, 2022; OQUAB *et al.*, 2023).

A profundidade dos objetos em uma cena possui a ótima capacidade de estimação pela aplicação de abordagens ativas, como por exemplo as aplicações com sensores LiDAR e câmeras RGB-D. Neste tipo de abordagem a informação de profundidade é alcançada rapidamente (TROUVé *et al.*, 2013; RODRIGUES *et al.*, 2020). A câmera RGB-D é um tipo específico de dispositivo de detecção de profundidade que combina uma imagem RGB e sua imagem de profundidade correspondente (ULRICH *et al.*, 2020). Elas podem ser utilizadas em vários dispositivos, como smartphones e sistemas aéreos não tripulados, por conta do seu baixo custo e consumo de energia (KIM *et al.*, 2020). No entanto, elas têm alcance de profundidade limitado e sofrem com reflexos especulares e absorção de objetos. Por isso, muitas abordagens de completação (determinação do menor espaço métrico que contém um dado espaço) de profundidade foram propostas para reduzir a diferença entre mapas de profundidade esparsos e densos (DONG *et al.*, 2021).

Os modelos de estimativa de profundidade monoculares trabalham com uma sequência de imagens extraídas de uma única câmera, por isso não requerem imagens retificadas. Este fato, simplifica e facilita o acesso sendo uma das principais vantagens em relação aos modelos estéreos.

Os métodos de última geração geralmente se enquadram em uma de duas categorias: projetar uma rede complexa que seja poderosa o suficiente para regredir diretamente o mapa de profundidade ou dividir a entrada em compartimentos ou janelas para reduzir a complexidade computacional. Nesses cenários, os *benchmarks* mais populares são os conjuntos de dados

KITTI (GEIGER *et al.*, 2013) e NYUv2 e os modelos são normalmente avaliados usando RMSE ou erro relativo absoluto.

2.3.2 Métricas de avaliação

Eigen *et al.* (2014) recomenda, para avaliação da eficiência dos modelos de estimativa de profundidade, um procedimento de avaliação baseado em 05 métricas: diferença relativa absoluta (Abs-Rel), erro quadrado relativo (Sq-Rel), erro quadrático médio (RMSE), RMSE-log e acurácia, com um limite (δt). As métricas são formuladas usando as seguintes equações [9]:

$$Abs - Rel = \frac{1}{|D|} \sum_{pred \in D} |\delta t - pred| / \delta t \tag{2.42}$$

$$Sq - Rel = \frac{1}{|D|} \sum_{pred \in D} \|\delta t - pred\|^2 / \delta t \tag{2.43}$$

$$RSME = \sqrt{\frac{1}{|D|} \sum_{pred \in D} \|\delta t - pred\|^2} \tag{2.44}$$

$$RSME - Log = \sqrt{\frac{1}{|D|} \sum_{pred \in D} \|\log(\delta t) - \log(pred)\|^2} \tag{2.45}$$

$$\delta t = \frac{1}{|D|} \left| \left\{ pred \in D \mid \max\left(\frac{\delta t}{pred'}, \frac{pred}{\delta t} \right) < 1.25^t \right\} \right| \times 100\% \tag{2.46}$$

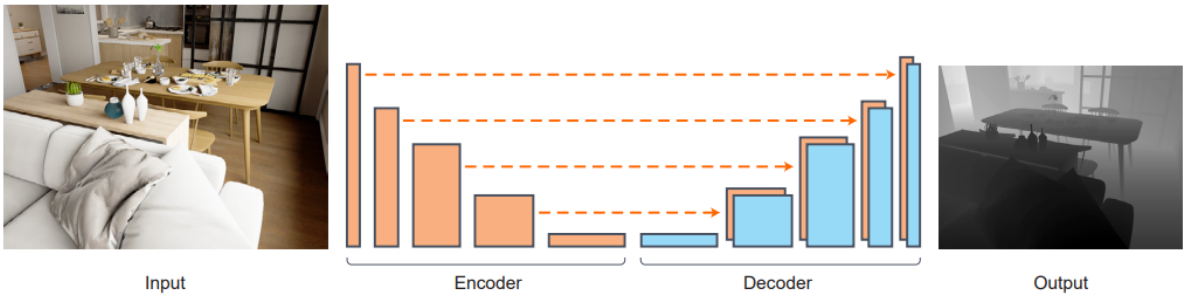
Onde, pred e o δt denotam a profundidade prevista e a verdade do solo, respectivamente. D representa o conjunto de todos os valores de profundidade previstos para uma única imagem, $|.|$ retorna o número de elementos em cada conjunto de entrada e δt representa o limite.

2.3.3 Visão geral dos métodos baseados em aprendizado profundo

Alhashim e Wonka (2018) apresentam uma rede neural convolucional para calcular um mapa de profundidade de alta resolução apartir de uma única imagem RGB com a ajuda de transferência de aprendizado. Seguindo uma arquitetura de codificador-decodificador padrão com conexões de salto, aproveitam os recursos extraídos usando redes pré-treinadas de alto desempenho ao inicializar o codificador, juntamente com estratégias de aumento e treinamento, gerando resultados mais precisos e obtendo mapas de profundidade detalhados e de alta precisão. A imagem RGB de entrada é codificada em um vetor de recursos usando a rede DenseNet-169 (HUANG *et al.*, 2017) pré-treinada no ImageNet (DENG *et al.*, 2009). Este vetor é então alimentado a uma série sucessiva de camadas de amostragem (LEHTINEN *et al.*, 2018), a fim de

construir o mapa de profundidade final na metade da resolução de entrada. Essas camadas de amostragem e as conexões de salto associadas formam o decodificador. A Figura 6 apresenta uma visão geral da arquitetura proposta por Alhashim e Wonka (2018).

Figura 6 – Visão geral da arquitetura proposta por Alhashim e Wonka (2018)



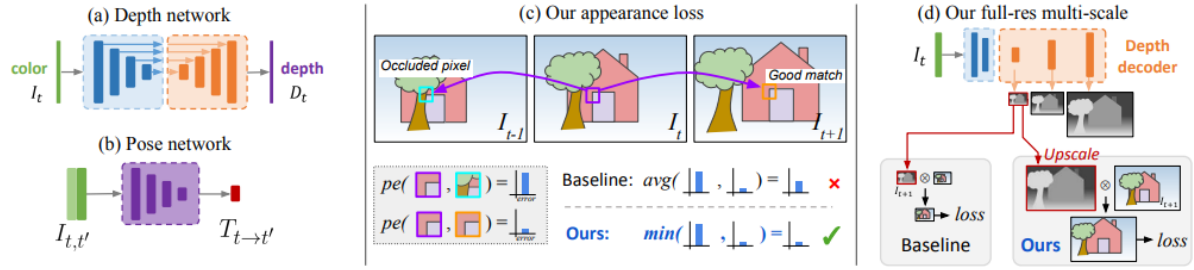
Fonte: Alhashim e Wonka (2018).

Já Godard *et al.* (2019) propõem um conjunto de melhorias em um método auto-supervisionado, de acordo com a arquitetura apresentada na Figura 7. Dentre as melhorias, a pequisa propõe (i) uma perda mínima de reprojeção, projetada para lidar com oclusões de forma robusta, (ii) um método de amostragem multiescala de resolução total que reduz artefatos visuais e (iii) uma perda de mascaramento automático para ignorar o treinamento de pixels que violam as suposições de movimento da câmera. Essas melhorias, em conjunto, resultam em mapas de profundidade melhorados quantitativa e qualitativamente em comparação aos métodos auto-supervisionados concorrentes. A Arquiterura proposta apresenta uma rede de profundidade,

A arquitetura porposta apresenta na Figura 7 (a) uma rede de profundidade, rede *Convolutional Networks for Biomedical Image Segmentation* (U-Net) (RONNEBERGER *et al.*, 2015) padrão, totalmente convolucional, para prever a profundidade. Já na Figura 7 (b) é apresentada uma rede de poses onde a pose entre um par de quadros é prevista com uma rede de poses separada. Na Figura 7 (c) é mostrada a reprojeção mínima por pixel. Quando as correspondências são boas, a perda de reprojeção deve ser baixa. No entanto, oclusões e desoclusões resultam em pixels da etapa de tempo atual não aparecendo nos quadros anterior e seguinte. A perda média da linha de base força a rede a corresponder aos pixels ocluídos, enquanto nossa perda mínima de reprojeção corresponde apenas a cada pixel na exibição em que é visível, levando a resultados mais nítidos. Por fim, a Figura 7 (d) traz a multiescala de resolução total onde è realizado um aumento das previsões de profundidade em camadas intermediárias e calculadas todas as perdas na resolução de entrada, reduzindo os artefatos de cópia de textura.

Para Ranftl *et al.* (2021a), o sucesso da estimativa de profundidade monocular depende de conjuntos de treinamento grandes e diversos. Por esse motivo desenvolveram um

Figura 7 – Visão geral da arquitetura proposta por Godard *et al.* (2019).

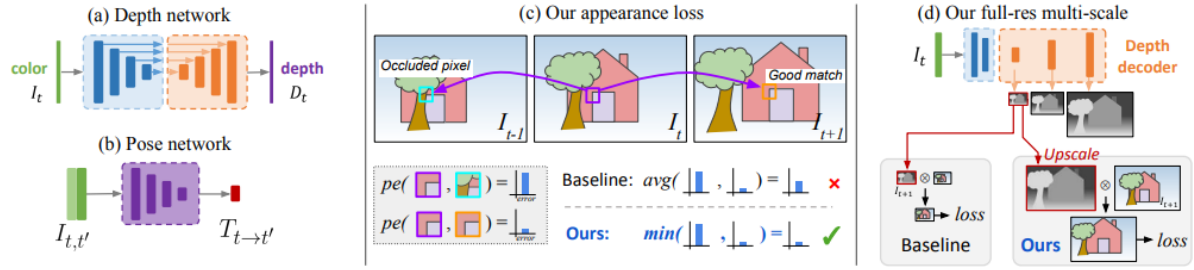


Fonte: Godard *et al.* (2019).

conjuto de ferramentas que permitem misturar vários conjuntos de dados durante o treinamento, mesmo que suas anotações sejam incompatíveis. Dessa forma, Ranftl *et al.* (2021a) propõem um objetivo de treinamento robusto que é invariante a mudanças na faixa e escala de profundidade. Enquanto isso, Lee *et al.* (2019) defende que nos esquemas de codificador-decodificador, a convolução repetida e as camadas de agrupamento espacial diminuem a resolução espacial das saídas de transição, e várias técnicas, como conexões de salto ou redes de deconvolução multicamadas, são adotadas para recuperar a resolução original para uma previsão densa efetiva. Por esse motivo propõem uma arquitetura de rede que utiliza novas camadas de orientação planar local localizadas em vários estágios na fase de decodificação, com a intenção de obter uma orientação mais eficaz de recursos densamente codificados para a previsão de profundidade desejada. A Figura 8 apresenta arquitetura proposta por Lee *et al.* (2019).

Como pode ser observado na Figura 8, a rede é composta por extrator de características densas (a rede base), extrator de informações contextuais (), camadas de orientação planar local e sua conexão densa para estimativa final de profundidade. As saídas das camadas de orientação planar local têm a resolução espacial total H permitindo atalhos dentro da fase de decodificação. São utilizadas conexões de salto da rede de base para conectar com saídas internas na fase de decodificação com resoluções espaciais correspondentes.

Figura 8 – Visão geral da arquitetura proposta por Lee *et al.* (2019).



Fonte: Lee *et al.* (2019).

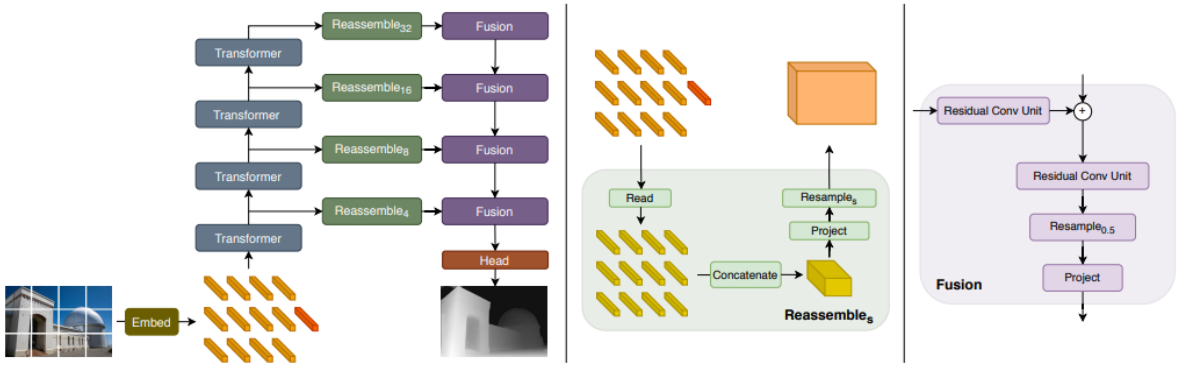
O trabalho de Ranftl *et al.* (2021a) apresenta uma arquitetura que aproveita os

transformadores de visão (DOSOVITSKIY *et al.*, 2021) no lugar de redes convolucionais como um *backbone* para tarefas de previsão densas, os transformadores de visão densa. A pesquisa reune vários estágios do transformador de visão em representações semelhantes a imagens em várias resoluções e os estágios são combinados progressivamente em previsões de resolução total usando um decodificador convolucional. O *backbone* do transformador processa representações em uma resolução constante e relativamente alta e possui um campo receptivo global em cada estágio. Essas propriedades permitem que o transformador de visão densa forneça previsões mais refinadas e globalmente coerentes quando comparadas a redes totalmente convolucionais. A Figura 9 apresenta uma visão geral da arquitetura proposta por Ranftl *et al.* (2021a).

Como pode ser obsevado na Figura 9, a imagem de entrada é transformada em *tokens* (laranja) extraindo *patches* não sobrepostos seguidos por uma projeção linear de sua representação achatada (DPT-Base e DPT-Large) ou aplicando um extrator de recursos -50 (HE *et al.*, 2016) (DPT-Hybrid). A incorporação da imagem é aumentada com uma incorporação posicional e um token de leitura independente do *patch* (vermelho) é adicionado. Os *tokens* são passados por vários estágios do transformador. Remontamos *tokens* de diferentes estágios em uma representação semelhante a uma imagem em várias resoluções (verde). Os módulos de fusão (roxo) fundem progressivamente e aumentam a resolução das representações para gerar uma previsão refinada.

A Figura 9 apresenta ao centro uma visão geral da operação *Reassembles* onde os *tokens* são montados em mapas de recursos com 1/s a resolução espacial da imagem de entrada. Já à direita, a Figura 9 mostra os blocos de fusão que combinam feições usando unidades convolucionais residuais (LIN *et al.*, 2016) e ampliam os mapas de feições.

Figura 9 – À esquerda uma visão geral da arquitetura proposta por Ranftl *et al.* (2021a).



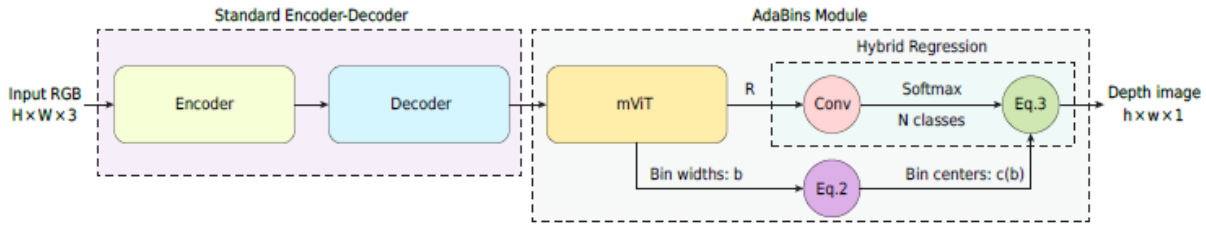
Fonte: Ranftl *et al.* (2021a).

A pesquisa de Bhat *et al.* (2021) aborda o problema de estimação de um mapa de profundidade densa de alta qualidade a partir de uma única imagem de entrada RGB. Para isso,

uma arquitetura de rede neural convolucional de codificador-decodificador de linha de base é utilizada e em seguida o processamento global de informações ajuda a melhorar a estimativa geral de profundidade. Dessa forma, Bhat *et al.* (2021) propôem um bloco de arquitetura, chamado de , baseado em transformador que divide a faixa de profundidade em *bins* cujo valor central é estimado de forma adaptativa por imagem. Os valores finais de profundidade são estimados como combinações lineares dos centros dos *bins*. A Figura 10 apresenta arquitetura proposta.

A arquitetura apresentada na 10 consiste em dois componentes principais: um bloco codificador-decodificador e o bloco estimador de largura de *bin* adaptativo chamado . A entrada da rede é uma imagem RGB de dimensões espaciais H e W, e a saída é uma imagem de profundidade h×w de canal único (por exemplo, metade da resolução espacial).

Figura 10 – Visão geral da arquitetura proposta por Bhat *et al.* (2021).



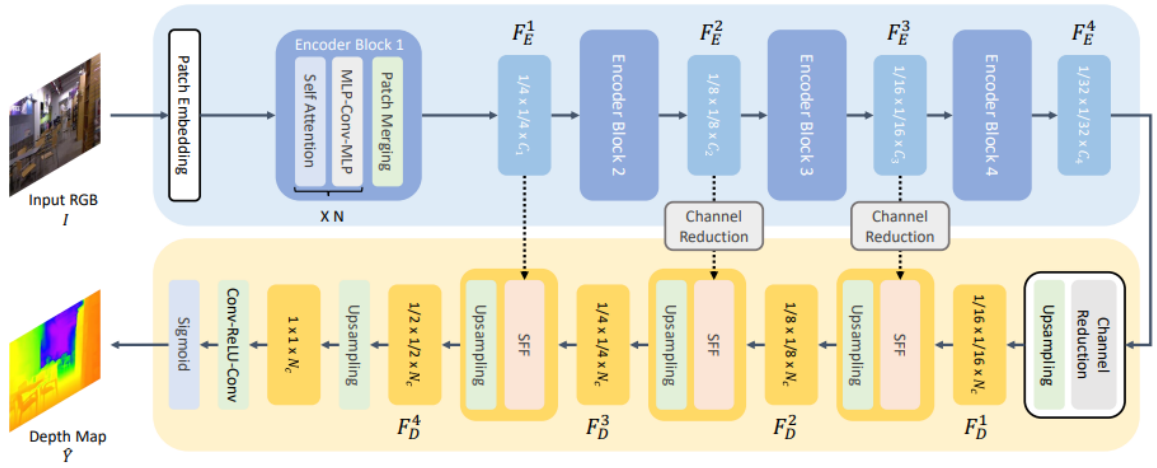
Fonte: Bhat *et al.* (2021).

Kim *et al.* (2022) propõem uma nova estrutura e estratégia de treinamento para estimativa de profundidade monocular visando melhorar ainda mais a precisão da previsão da rede. Dessa forma, implementam um codificador de transformador hierárquico para capturar e transmitir o contexto global e projetam um decodificador leve e poderoso para gerar um mapa de profundidade estimado considerando a conectividade local. A construção caminhos conectados entre recursos locais multiescala e o fluxo de decodificação global ao módulo de fusão de recursos seletivo proposto, permite que a rede passa integrar ambas as representações e recuperar detalhes finos. A arquitetura proposta é apresentada na Figura 11 e os principais componentes da arquitetura são o codificador, o decodificador e as conexões de salto com módulos de fusão de recursos.

2.3.4 Estimador de proundidade AdaBins

De acordo com o Bhat *et al.* (2021) a obordagem do estimador de profundidade pode ser vista como uma generalização da estimativa de profundidade por meio de uma rede de regressão ordinal proposta por Fu *et al.* (2018). Fu *et al.* (2018) observaram que uma melhoria de desempenho poderia ser alcançada se a tarefa de regressão de profundidade fosse transformada

Figura 11 – Visão geral da arquitetura proposta por Kim *et al.* (2022).



Fonte: Kim *et al.* (2022).

em uma tarefa de classificação. Então, eles propuseram dividir a faixa de profundidade em um número fixo de *bins* de largura predeterminada. Partindo desse princípio, o bloco inicialmente calcula *bins* adaptativos que mudam dinamicamente de acordo com as características da cena de entrada, $D = (dmin, dmax)$ em N *bins*. Em seguida utiliza uma abordagem de classificação para realizar uma discretização de valores de profundidade resultando em baixa qualidade visual com descontinuidades de profindidades nítidas óbivias. Os valores finais de profundidade são previstos como uma combinação linear dos centros dos *bins*, possibilitando a combimação das vantagens da classificação com as da regressão do mapa de profundidade.

2.3.4.1 Arquitetura

A arquitetura, apresentada na Figura 10, consiste em dois componentes principais: (i) um bloco codificador-decodificador construído em um codificador *EfficientNet B5* (TAN; LE, 2019) pré-treinado e um decodificador de aumento de amostragem de recurso padrão; (ii) o bloco estimador de largura de *bin* adaptativo.

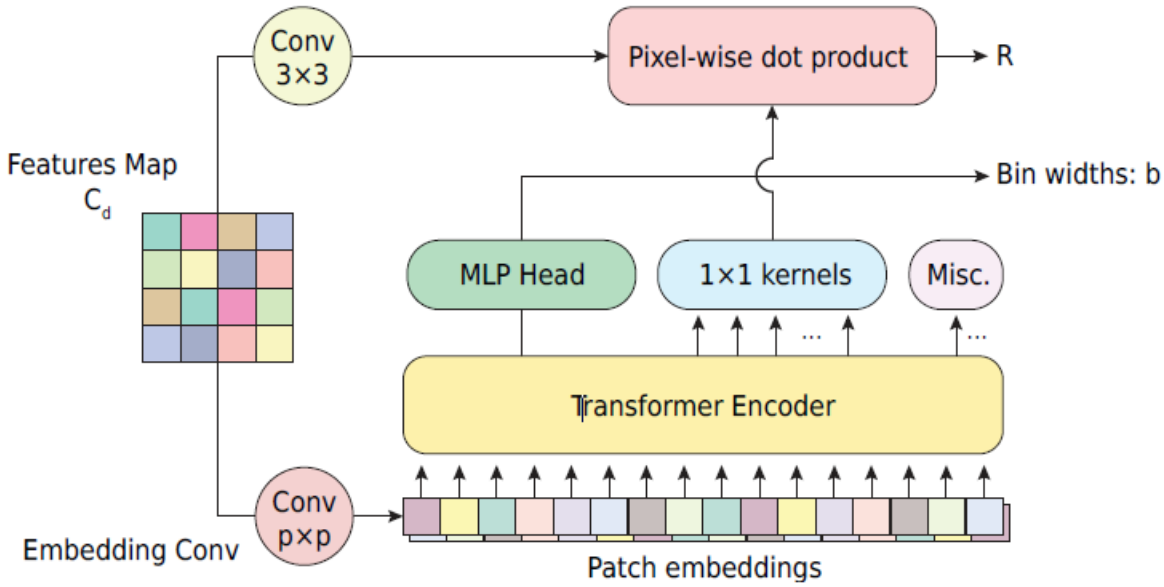
O primeiro componente é baseaddo na rede de regressão de profundidade simples de Alhashim e Wonka (2018) com a alteração do encoder DenseNet (HUANG *et al.*, 2017) para *EfficientNet B5* e a utiização de uma função de perda apropriada para a nova arquitetura. A saída do decodificador é um tensor $x_d \in \mathbb{R}^{h \times w \times C_d}$ e não uma única imagem de canal representando os valores finais de profundidade. O segundo componente é o módulo AdaBins proriamente dito, onde a entrada são os recursos decodificados de tamanho $h \times w \times C_d$ e a saída é um tensor de tamanho $h \times w \times 1$.

No módulo AdaBins o bloco de entrada é chamando de *mini-ViT* e possui duas

saídas: (i) um vetor b de larguras de *bin*, que define como o intervalo de profundidade D deve ser dividido para a imagem de entrada e (ii) mapas de atenção de alcance R de tamanho $h \times w \times C$, que contém informações úteis para computação de profundidade em nível de pixel.

A Figira 12 mostra uma visão geral do bloco *mini-ViT*. A entrada para o bloco é um mapa de recursos multicanal da imagem de entrada. O bloco inclui um codificador Transformer que é aplicado em incorporações de *patch* da entrada com o objetivo de aprender a estimar larguras de compartimentos b e um conjunto de kernels convolucionais necessários para calcular nossos mapas de alcance-atenção-R.

Figura 12 – Visão geral do bloco *mini-ViT*.



Fonte: Bhat *et al.* (2021).

Para obtenção do vetor de laguras de *bins* as características decodificadas passam por um bloco convolucional com tamanho de *kernel* $p \times p$, passo p e número de canais de saída E. O resultado dessa convolução é um tensor de tamanho $h/p \times w/p \times E$ que é então reformulado em um tensor espacialmente achatado $x_p \in \mathbb{R}^{S \times E}$, onde $S = \frac{hw}{p^2}$ serve como o comprimento efetivo da sequência para o transformador. Em seguida são adicionadas codificações aprendidas ao *patch embeddings* para depois enviá-las ao Transformador que gera uma sequência de incorporações de saída $x_0 \in \mathbb{R}^{S \times E}$. O bloco *MLP Head* utiliza uma ativação ReLU e produz um vetor N-dimensional b' que é normalizado de modo que sua soma seja 1. A equação 2.47 formula a obtenção do vetor de larguras de *bins*.

$$b_i = \frac{b'_i + \varepsilon}{\sum_{j=1}^{N} \left(b'_j + \varepsilon \right)}, \varepsilon = 10^{-3} \tag{2.47}$$

Os recursos decodificados representam uma informação de alta resolução e nível de pixel local, enquanto as incorporações de saída do transformador efetivamente contêm mais informações globais. Conforme pode ser observado na Figura 12 o segundo bloco de saídas do *Transformer Encoder* são utilizadas como um conjunto de kernels convolucionais 1×1 e são convoluídas com os recursos decodificado, seguindo uma camada convolucional 3×3, para obter os mapas R de alcance-atenção. O uso da incorporações de saída como kernels convolucionais permite que a rede integre informações globais adaptativas do transformador nas informações locais dos recursos decodificados. Assim, R e b são usados juntos para obter o mapa de profundidade final.

Os mapas R de alcance-atenção são passados através de uma camada convolucional 1×1 para obter canais N que são seguidos por uma ativação *Softmax*. Bhat *et al.* (2021) interpretam os N escores *Softmax* $p_k, k = 1, \cdots, N$, em cada pixel como probabilidades sobre N centros de profindidade do bin $c(b) := \{c(b_1), c(b_2), \cdots, c(b_N)\}$ que é calculado a partir do vetor de larguras de *bins* b de acordo com a Equação 2.48.

$$c(b_i) = d_{min} + (d_{max} - d_{min}) \left(b_i/2 + \sum_{j=1}^{i-1} b_j \right) \tag{2.48}$$

Por fim, o valor de profundidade $\tilde{d}$, em cada pixel, é calculado pela combinação linear das pontuações *Softmax* naquele pixel e os centros de profundidade $c(b)$ como formulado na Equação 2.49.

$$\tilde{d} = \sum_{k=1}^{N} c(b_k)\, p_k \tag{2.49}$$

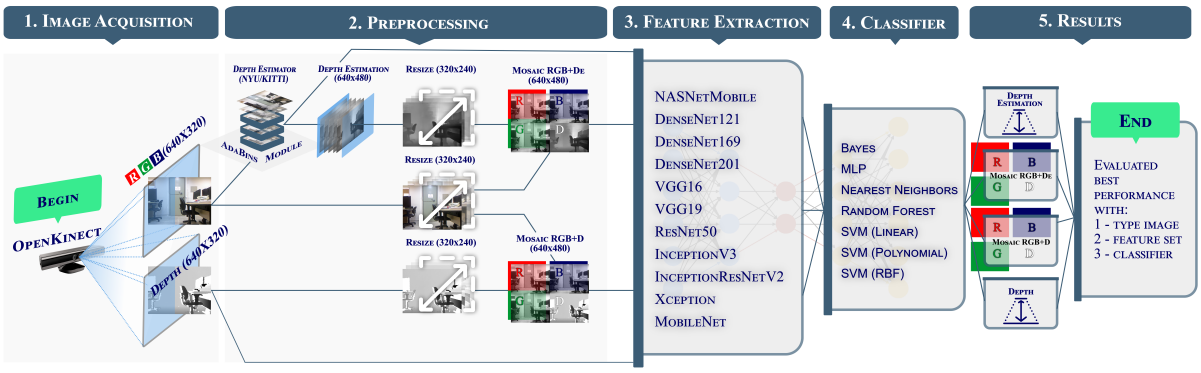
3 METODOLOGIA

Nesta Seção será descrita toda Metodologia seguida para a elaboração deste trabalho. Primeiro será explicada a forma de aquisição e estruturação do conjunto de imagens utilizadas, em seguida o pré-processamento escolhido será apresentado, assim como serão detalhadas a etapas de extração de características e classificação, e por fim, as métricas de avaliação utilizadas serão definidas.

A Figura 13 mostra os passos da metodologia aplicada nesta dissertação. Ela consiste em 05 (cinco) etapas: (1) Aquisição de imagens, (2) pré-processamento, (3) extração de características, (4) classificação e (5) resultados. A primeira etapa é a aquisição de imagens, onde um sensor Kinect, por meio do software libfreenect de código aberto, disponível na comunidade OpenKinect, captura as imagens RGB e de profundidade do ambiente (SARBOLANDI *et al.*, 2015). Na segunda etapa, pré-processamento, são estimados mapas de profundidade densos de alta qualidade para cada imagem RGB de entrada utilizando o estimador de profundidade (BHAT *et al.*, 2021) e todas as imagens (Kinect e) são preparadas para gerar os bancos de imagens de profundidade (lida e estimada) e RGB-D (Kinect e AdaBins).

Na terceira etapa, extração de características, são gerados os vetores de características que servirão de entrada para os classificadores, que realizam a tarefa de predição na quarta etapa. Por fim, na quinta etapa, são utilizadas métricas de avaliação para cada combinação de arquitetura e classificador, em todos os bancos de imagens analizados, concluindo a última etapa do processo. Assim, o robô finaliza sua atividade de localizaçõ, permitindo, portanto, que conclua sua tarefa de navegação.

Figura 13 – Fluxograma da metodologia adotada.



Fonte: Elaborada pelo autor.

A Seção 3.1, mostra como foi realizada a aquisição das imagens, como também o mapa topológico do ambiente de navegação. A Seção 3.2 detalha a estimação dos mapas de profundidade pelo bloco AdaBins (BHAT *et al.*, 2021), assim como o pré-processamento das imagens adquiridas e geradas para criação dos bancos de imagens que serão utilizados. A Seção 3.3 especifica como os estratores de características foram utilizados e a Seção 3.4 traz a etapa de classificação. Por último na Seção 3.5 são apresentadas as métricas para avaliação dos resultados de classificação.

3.1 Aquisição de dados

Para este trabalho foi utilizado um banco de imagens capturadas por um sensor Kinect instalado em um robô móvel constituído de uma base de metal, provido com dois motores, baterias, placas de circuito lógico e um sistema de engrenagens. A comunicação com o robô é realizada através de telemetria, possuindo um software próprio para a troca de dados com um microcontrolador. O sensor Kinect captura as imagens RGB e de profundidade do ambiente (SARBOLANDI *et al.*, 2015) por meio do software libfreenect versão 1 de código aberto, disponível na comunidade OpenKinect. A Tabela 1 apresenta mais especificações do veículo e sensor utilizado para para captura das imagens.

Tabela 1 – Especificações do robô móvel considerado.

Características do Robô Móvel	
Modelo	VEX EDR Kit
Motor	Servo EDR 393
Navegação	Interno
Fonte de Energia	6 baterias NiMH 1.2V e 2600mAh
Características do sensor Kinect	
Resolução do sensor RGB	640x480px, 30fps
Resolução do sensor de profundidade	640x480px, 30fps

A navegação do robô é executada em um ambiente do tipo interno, o segundo andar de um prédio do campus universitário. O ambiente interno foi escolhodo por ser um ambiente controlado e possuir características que favorecem o reconhecimento, auxiliando na localização e navegação do robô.

A Figura 14 apresenta o mapa topológico do ambiente, que é composto por seis salas, denominadas de R1 a R6. Cada sala é dividida em um ou dois nós, totalizando nove nós, rotulados com letras de A a I; cada nó representa a posição do robô na sala. As imagens foram capturadas em posições estratégicas, situadas a partir desses nós. Os nós são divididos em classes

e são rotuladas de C1 a C30. As classes são as possíveis direções que podem ser executadas pelo robô e se referem à orientação dentro de cada nó, representando as classes para a localização do robô em nossa abordagem. Se o robô identificar a classe, ele pode definir a direção que deve seguir, o nó e a sala em que se encontra.

Como exemplo, o robô pode encontrar-se na sala denominada R2 (nó C) e posicionado de frente para a porta de acesso a sala R1 (classe 8) ou em direção à sala denominada R3 (classe 11). O sistema de visão computacional proposto nesse trabalho tem a capacidade de identificar essas diferentes posições. As arestas do mapa topológico são simbolizadas pelos caminhos praticáveis pelo robô em sua navegação, como demonstrado na Figura 14.

Na 14 as salas são representadas pelos balões vermelhos. O balão verde representa o nó do mapa. As classes (C1 a C30) representam a direção do robô no momento da captura das imagens. Cada sala está interligada com a outra e o robô pode se mover entre elas executando os comandos programados em sua odometria.

A Tabela 2 ilustra algumas das rotas que o robô móvel pode seguir no ambiente. Os comandos que podem ser executados são: SF - Siga em frente; VD90 - Vire à direita 90°; VE90 - Vire à esquerda 90°; VD180 - Vire à direita 180°; VE180 - Vire à esquerda 180°. Por exemplo, para o robô executar a tarefa de locomoção da rota C20 a C9, cinco comandos distintos foram dados. O robô começa na posição (classe) C20 (nó F). Primeiramente é enviado a ele um comando para virar 90° para a direita, sequencialmente, são dados dois comandos "Siga em frente". Em seguida, outro comando é enviado ao robô para virar 90° para a esquerda, e por fim outro comando para seguir em frente, chegando assim ao seu destino, que corresponde a classe C9 (nó C).

Tabela 2 – Exemplo de rotas com base no mapa topológico e comandos disponíveis.

Rota	**Comandos**
C1 to C7	SF, VE90
C7 to C16	VE90, SF, SF, VD90, SF, VD90, SF, VE180
C16 to C20	VD180, SF, VE90
C20 to C9	VD90, SF, SF, VE90, SF
C9 to C25	SF, SF
C25 to C30	VD180, SF, VE90, SF, VD90
C30 to C6	VD90, SF, VE90, SF, TR90, SF, SF, VE180
C6 to C1	SF, VD180

Para este trabalho foram capturadas 50 (cinquenta) imagens com resolução de

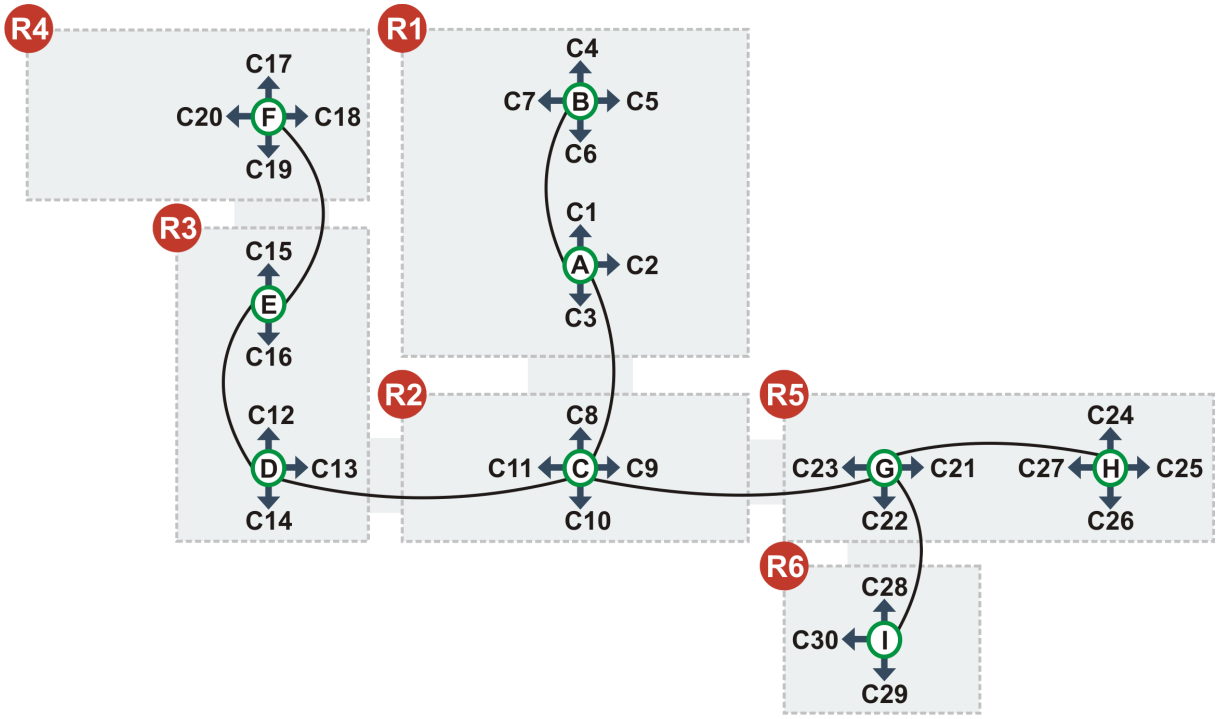


Figura 14 – Mapa topológico do ambiente, com nós e arestas.

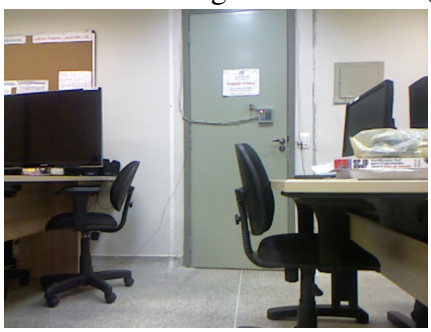
640x480 pixels para cada classe no ambiente escolhido. A aquisição de uma quantidade de imagens considerável para cada classe foi feita para que fosse estabelecido um funcionamento correto do sistema de navegação, abrangendo, assim, possíveis circunstâncias nas quais o robô esteja siatuado nas proximidades do nó, mas, não exatamente nele. Com isso, foram construidods 02 (dois) bancos de imagens, um com imagens RGB e outro com imagens de profundidade, totalizando 1500 imagens por banco de imagens. A Figura 15(a) mostra um exemplo de imagem RGB e Figura 15(b) um exemplo de imagem de profundidade, ambas, coletadas pelos sensores do Kinect. Essas imagens são referentes à classe 3, a qual está situada no nó A, na sala R1.

3.2 Pré-processamento

Como uma das contribuições deste trabalho tem-se a adição, na etapa de pré processamento, do estimador de profundidade AdaBins (BHAT *et al.*, 2021), com a finalidade de estimar a profundidade a partir das imagens RGB capturadas, gerando novos bancos de imagens para realização de comparações de desempenho posteriormente. O estimador AdaBins é apontado na literatura como uma rede capaz de estimar profundidades de forma precisa e com alta qualidade quando comparado a outros modelos desenvolvidos para estimativa de profundidade monocular com base no conjunto de dados NYU DEPTH V2 (MASOUMIAN *et al.*, 2022).

Inicialmente, são estimados mapas de profundidade densos de alta qualidade para

Figura 15 – Amostras de imagens da classe 3 (nó A).



(a) Imagem RGB.

(b) Imagem de profundidade.

(c) Imagem de profundidade estimada.

Fonte: Autor.

cada imagem RGB de entrada utilizando o estimador de profundidade AdaBins (BHAT *et al.*, 2021). O estimador de profundidade AdaBins foi previamente pré-treinado com o popular conjunto de dados de ambientes internos NYU DEPTH V2 que fornece imagens e mapas de profundidade, com limite superior de 10 metros, para diferentes cenas internas capturadas em uma resolução de 640×480 pixels (SILBERMAN *et al.*, 2012). A Figura 15(c) mostra um exemplo de imagem de profundidade estimada.

Em seguida, as imagens capturadas pelo sensor Kinect (RGB e profundidade) e as imagens de profundidade geradas pelo estimador AdaBins são dimensionadas para o tamanho de 320 x 240 pixels. As imagens RGB tem seus canais de cores separados e concatenados com as imagens de profundidade (lidas e estimadas) para gerar os conjuntos de dados de imagens em mosaico RGB-D. Nesses novos conjuntos de dados, as imagens são dispostas lado a lados formando uma nova imagem de tamanho 640x480 pixels.

A Figura 16 mostra exemplos de amostras do conjunto de dados RGB-D (Kinect) de cada classe e em qual sala física elas estão. Cada classe contém 50 imagens distintas. As salas são representadas pelos balões vermelhos e as classes (C1 a C30) representam cada direção do robô no momento da captura das imagens.

Além das imagens RGB-D, são feitas cópias das imagens originais de profundidade

e imagens de profundidade estimada para que seja criado os conjuntos de dados de profundidade. Ao final da etapa de pré-processameto, tem-se 04 (quatro) conjuntos de imagens, são eles: Profundidade (Kinect), RGB-D (Kinect), Profundidade estimada (AdaBins) e RGB-D (AdaBins). Todos os bancos de imagens possuem 50 amostras por classe, totalizando 1500 amostras para cada conjunto de dados.

Figura 16 – Exemplos de amostras do conjunto de dados RGB-D (Kinect) para cada classe.

3.3 Extração de características

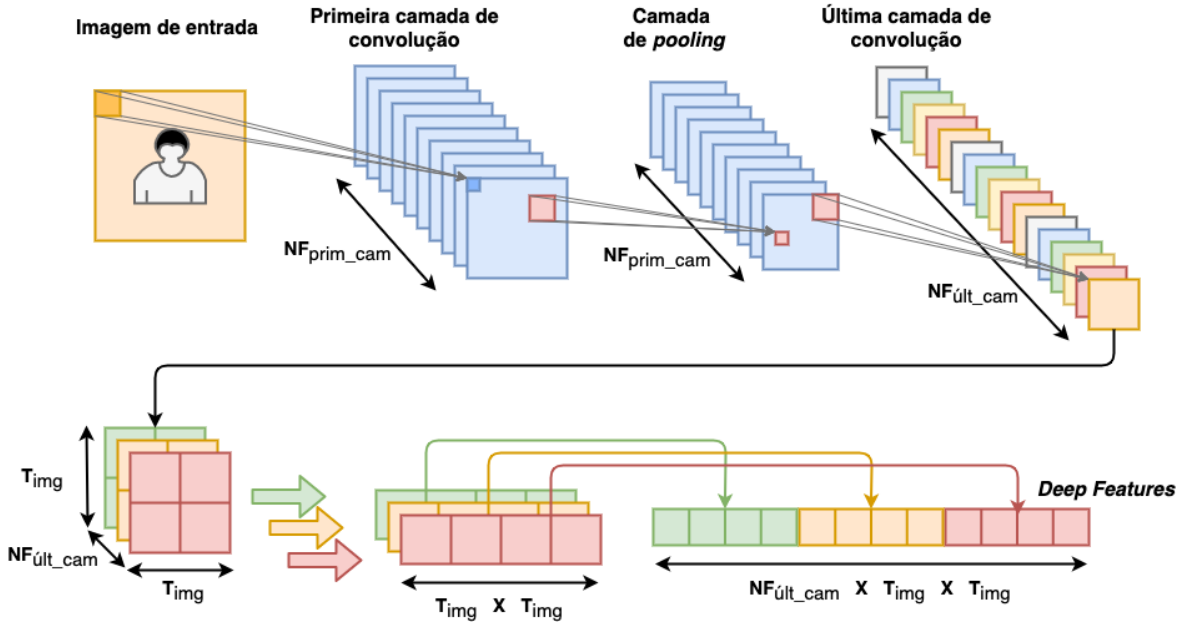
Na etapa de Extração de Características, são gerados os vetores de características que servirão de entrada para as técnicas de Aprendizado de Máquina posteriormente para a realização da tarefa de classificação.

Neste trabalho, utilizou-se a CNN com conceito de *Transfer Learning*, que utiliza o poder descritivo de modelos pré-treinados como ponto de partida no processo de classificação de imagens. Dessa forma, a última camada convolucional, que é a camada totalmente conectada, é removida e o redimensionamento da entrada dessa camada removida é realizado em um vetor unidimensional (PAN; YANG, 2010; TAN *et al.*, 2018). Feito isso, o modelo pré-treinado passa a

ser utilizado como um extrator de características, não se comportando mais com um classificador. Essa técnica tem sido bastante utilizada em diversos trabalhos na literatura (GRUBINGER *et al.*, 2017; MAZO *et al.*, 2018; SUH *et al.*, 2018; LIN *et al.*, 2018; WANG *et al.*, 2019; CAI *et al.*, 2020; KLUS *et al.*, 2021; MORAWSKA *et al.*, 2022).

Inicialmente, a imagem a ser processada, é submetida a uma sequência de transformações não lineares que são definidas de acordo com a arquitetura utilizada, convertendo a imagem em um conjunto de pequenas matrizes. Em seguida, é realizado o redimensionamento de cada uma dessas matrizes para um vetor unidimensional, onde o conjunto desses vetores é concatenado, formando então um único vetor que pode ser interpretado como os atributos extraídos pela CNN. Esse processo é exemplificado na Figura 17 (SILVA, 2019), onde NF_{prim_cam} simboliza a quantidade de filtros aplicados nas primeiras camadas, NF_{lt_cam} representa a quantidade de filtros aplicados na ultima camada e T_{img} é o tamanho da imagem original após as transformações de todas as camadas da CNN.

Figura 17 – Ilustração das duas primeiras etapas de uma CNN.



Fonte: (SILVA, 2019).

Ao final, os vetores de atributos seriam submetidos a uma ou mais camadas totalmente conectadas utilizadas para treinar a rede no banco de dados de referência. No entanto, as últimas camadas conectadas são removidas do modelo. Dessa forma, quando uma imagem é submetida a essa nova arquitetura, se tem um vetor de informações que será utilizado por camadas totalmente conectadas para compor as proobabilidades de classe.

As seguintes arquiteturas CNN foram consideradas para a etapa de extração de carac-

terísticas: VGG16, VGG19, MobileNet, ResNet50, InceptionV3, Xception, InceptionResNetV2, DenseNet121, DenseNet169, DenseNet201 e NASNetMobile. Todas elas foram implementadas, inicializadas e pré-treinadas com o banco de dados *ImageNet* conforme especificado em seus respectivos artigos. Feito isso foram utilizadas como extratores de características nas abordagens utilizadas neste trabalho. A tabela 8 apresenta a quantidade de atributos extraídos por cada arquitetura e tempos de extração que serão abordados e analisados posteriormente.

3.4 Classificação

Neste trabalho, foram utilizados classificadores com princípios diversos com o intuito de se abranger diferentes aspectos na tarefa de classificação. Nesse sentido, foraom utilizados os seguintes classificadores: Bayes, kNN, *Random Forest*, MLP e SVM.

O classificador Naive Bayes é baseado em probabilidade estatística, e durante o processo de classificação operou com a função de densidade de probabilidade gaussiana. Já o classificador é uma técnica baseada em distância e teve seu hiperparâmetro k escolhido por meio de uma pesquisa em grade (*grid search*), testando os valores ímpares de 1 a 11.

O classificador *Random Forest Random Forest* apresentada características de um classificador de *bagging*. A busca aleatória a partir do *Random Forest* foi usada para encontrar: o número de atributos para a melhor divisão (1,2,3, ... ou 10); a profundidade máxima da árvore a ser utilizada (6 ou nenhuma), o número mínimo de amostras necessárias para dividir um nó interno (1,2,3, ... ou 10), o número mínimo de amostras por folha a serem empregadas (1,2,3, ... ou 10), se amostras de autoinicialização devem ser usadas na construção de árvores ou não, e qual função deve ser usada para medir a qualidade de uma divisão (gini ou entropia). Além disso, o processo de treinamento do *Random Forest* utilizou 3000 estimadores.

O classificador MLP consiste de uma rede neural artificial e durante o processo de classificação realizou seu treinamento usando o método Levenberg-Marquardt e neurônios variando de 2 a 1000 na camada oculta. Já o classificador SVM é fundamentado pela Teoria do Aprendizado e utilizou kernels Linear, Polonomial e RBF, e hiperparâmetros C e γ com os valores 2^{-5}, 2^{-4}, 2^{-3}, ...,2^{15} and 2^{-15}, 2^{-14}, ...,2^{3}, respectivamente. Os hiperparâmetros do MLP e da SVM foram determinados através de validação cruzada (*cross validation*) com 10-folds.

A Validação cruzada é um método que engloba treinamentos e testes de um modelo de Aprendizagem de Máquina em diferentes arranjos de um mesmo conjunto de dados. Existem

02 (dois) tipos mais comuns: *hold out* e k-*fold*. O *hold out* divide o conjunto de amostras em 02 (dois) subconjuntos, um para a etapa de treinamento e outro para etapa de validação. Já o k-*fold* efetua a divisão das amostras em *k* subgrupos. Dessa forma, são executadas *k* iterações para a etapa de treinamento e um subconjunto é dedicado para a etapa de teste em cada uma das iterações, enquanto os restantes formam o conjunto utilizado na etapa de treino. Em outras palavra, são aplicados k-*folds* para a fase de treinamento e os outros *folds* são utilizados na fase de teste.

Neste trabalho, após a etapa de extração de características, foi aplicada validação cruzada por *hold out*, onde cada base de dados formada pelos atributos resultantes teve seus padrões divididos em 02 (dois) conjuntos, de forma que 80% das amostras foram utilizadas na fase de treinamento e 20% na fase de teste, por 10 iterações. Para cada repetição do *hold out*, foi considerada a validação cruzada com 10 *folds* e busca em grade (*grid search*) (HAYKIN, 2010).

3.5 Métricas de avaliação

Afim de avaliar o desempenho das redes utilizadas neste trabalho, duas métricas foram escolhidas: Acurácia e F1-Score. A avaliação do F1-Score é interessante porque as métricas *Precision* e *Recall* são essenciais para o sistema, uma vez que falsos positivos e falsos negativos não são desejáveis. A análise conjunta das métricas utilizadas pode demosntrar o quão acertivo foram os classificadores. É importante destacar que a classe corresponde à posição e à direção do robô no ambiente, de acordo com o mapa topológico.

Essas métricas são calculadas a partir das frequências de classificação de cada classe para um extrator expecífico. Dessa forma, para calcular cada métrica de avaliação, deve-se conhecer a matriz de confusão para cada método de classificação. A Tabela 3 apresenta a matriz de confusão de ordem 30 que retrata as localizações do robô preditas de forma certa ou errada por um método de classificação.

Tabela 3 – Matriz de confusão para um problema de classificação multiclasse.

Classe Real \ **Classe Prevista**	**1**	**2**	**3**	**...**	**30**
1	VP	FN	FN	FN	FN
2	FP	VN	FN	FN	FN
3	FP	FN	VP	FN	FN
⋮	⋮	⋮	⋮	⋮	⋮
30	FP	FN	FN	FN	VN

Fonte: Autor.

Existem quatro tipos de frequências, o Verdadeiro Positivo (VP) que corresponde à previsão correta da classe para uma amostra rotulada, classificando uma amostra com a posição e direção do robô no ambiente de forma correta. Já o Verdadeiro Negativo (VN) que representa a previsão correta para a classe que a amostra não é rotulada, corresponde a reconhecer que outra localização foi corretamente realizada pelo classificador, por exemplo. Enquanto que o Falso Positivo (FP) é a frequência de previsões incorretas de uma classe para a qual a amostra está rotulada, como reconhecer uma outra localização no lugar da localização avaliada. Por fim, o Falso Negativo (FN) é a previsão incorreta para a classe que a amostra não pertence, como reconhecer a localização em análise incorretamente como outra localização, por exemplo. Na Tabela 3 a classe 3 foi destacada para um melhor entendimento das frequências de classificação.

A Acurácia representa a quantidade de acertos dividido pela quantidade de predições. É calculada por meio da Equação 3.1.

$$Acuracia = \frac{\text{n}^\circ\text{. total de acerto}}{\text{n}^\circ\text{. total de amostras no conjunto de dados}} \tag{3.1}$$

O F1-Score caracteriza a relação entre as métricas Sensibilidade (*Recall*) e Precisão, de forma mais específica, corresponde à média harmônica entre Sensibilidade e Precisão. Essa métrica é calculada de acordo com a Equação 3.2.

$$F1 - Score = \frac{2 \times Precisão \times Sensibilidade}{Precisão + Sensibilidade} \tag{3.2}$$

Onde, a Sensibilidade é a fração de verdadeiros positivos e Precisão é parcela de verdadeiros positivos relativa a todas as predições positivas. Elas são calculas pelas Equações 3.3 e 3.4, respectivamente.

$$Sensibilidade = \frac{VP}{VP + FN} \tag{3.3}$$

$$Precisão = \frac{VP}{VP + FP} \tag{3.4}$$

4 RESULTADOS

Os resultados deste trabalho são apresentados em três etapas. Na primeira etapa, Seção 4.1, todas as combinações de arquiteturas CNN com os classificadores considerados tiveram suas métricas de avaliação comparadas a partir dos dois conjuntos de imagens, profundidade e RGB-D, provenientes do ambiente real adquiridas através do sensor Kinect. Na segunda etapa, Seção 4.2, a mesma análise é realizada nos dois conjuntos de imagens, profundidade estimada e RGB-D, provenientes do estimador de profundidade AdaBins (BHAT *et al.*, 2021) a partir das imagens RGB capturadas pelo sensor Kinect. Por fim, Seção 4.3, as melhores combinações das duas análises são comparadas e têm seus tempos de processamento analisados.

Todos os experimentos foram realizados em um computador com processador Intel Core i7 7th Gen, 16 GB de RAM, placa de vídeo *GeForce* GTX 1050 Ti 4GB com GPU e sistema operacional Windows 10 Professional 64 bits. Para implementação dos algoritmos e obtenção dos resultados, foram utilizados os *frameworks Scikit-learn*, *Pytorch*, *Tensorflow* e *Keras* através da linguagem *Python*.

Em todas as análises, as imagens foram divididas aleatoriamente em 10 grupos, sendo 20% para teste e o restante para treinamento, em cada uma das 10 iterações e os melhores valores são destacados na cor verde.

4.1 Análise de desempenho da classificação com sensor kinect

Nesta seção, discutimos os resultados obtidos combinando a extração de atributos utilizando CNNs e a classificação com métodos de aprendizado de máquina para os conjuntos de dados gerados a partir do sensor Kinect, são eles: Profundidade e RGB-D.

As tabelas 4 e 5 apresentam os valores médios e desvios padrão de Acurácia e F1-Score, respectivamente, obtidos pelas várias arquiteturas de redes neurais convolucionais (CNN) combinadas com os diferentes classificadores para a tarefa de classificação das imagens de profundidade e RGB-D capturadas a partir do sensor Kinect.

Observando a tabela 4, pode-se verificar que, para as imagens de profundidade, a arquitetura MobileNet apresentou a melhor acurácia média, com valor de 99,675%, além de ter obtido o melhor F1-Score em conjunto com o classificador kNN, com valor de 99,690%, como pode ser verificado na tabela 5. Por outro lado, a pior combinação de arquitetura e classificador foi obtida com a arquitetura VGG16 em conjunto com o classificador SVM (Poly.), apresentando

Tabela 4 – Acurácia alcançada para as imagens de profundidade e RGB-D obtidas a partir do sensor Kinect.

Depth							
Classifier / **Architecture**	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	86.132±0.827	98.301±0.859	97.203±0.596	95.687±1.046	98.035±0.816	5.353±2.762	98.168±0.854
VGG19	85.733±0.806	97.425±0.806	96.806±0.680	96.396±0.573	97.868±0.748	22.21±1.067	98.132±0.943
MobileNet	92.534±1.011	99.675±0.211	96.856±0.843	98.069±0.748	99.214±0.452	98.549±0.490	99.341±0.452
InceptionV3	86.361±1.558	94.064±0.772	94.528±0.980	96.463±0.909	96.655±0.646	10.891±3.324	96.655±0.490
Xception	92.751±1.796	98.097±0.340	96.371±1.011	97.507±0.929	98.457±0.611	26.841±1.989	98.391±0.542
ResNet50	77.482±1.781	90.325±0.389	93.52±0.249	82.589±1.204	94.01±0.957	12.987±1.638	93.668±0.742
InceptionResNetV2	88.228±1.408	95.836±0.909	94.676±1.087	95.984±0.499	96.926±1.062	15.929±2.974	96.992±1.062
NASNetMobile	83.934±1.660	92.719±1.514	93.265±1.211	92.178±1.185	96.737±0.772	95.538±1.681	96.473±0.748
DenseNet121	80.368±1.083	97.673±0.712	97.473±0.686	97.597±0.542	98.178±0.558	19.08±2.692	98.244±0.327
DenseNet169	86.462±1.668	98.267±0.742	97.99±0.827	97.858±0.611	98.525±0.267	9.835±2.911	98.658±0.389
DenseNet201	86.924±0.646	98.264±0.471	98.85±0.581	97.742±0.806	98.476±0.827	11.853±0.859	98.74±0.827
RGB-D							
Classifier / **Architecture**	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	90.943±0.951	99.358±0.561	98.784±0.538	97.441±1.865	99.629±0.486	4.597±1.191	99.629±0.486
VGG19	92.069±1.639	99.211±0.635	98.782±0.803	98.634±0.957	99.324±0.712	15.505±3.476	99.324±0.712
MobileNet	96.789±0.508	99.873±0.253	98.431±0.854	99.193±0.768	99.746±0.506	99.746±0.506	99.873±0.253
InceptionV3	96.207±1.205	97.726±0.795	98.487±0.878	99.041±0.608	98.899±0.981	9.569±5.327	98.899±0.981
Xception	95.942±1.438	99.352±0.588	98.270±1.007	98.623±0.933	99.316±0.630	17.474±8.369	99.316±0.630
ResNet50	83.280±1.943	92.362±2.005	96.098±2.067	85.882±19.318	96.661±1.481	8.385±4.033	96.779±1.583
InceptionResNetV2	93.400±2.494	97.996±0.449	97.684±1.413	98.484±1.213	98.635±0.893	12.566±2.087	98.635±0.893
NASNetMobile	90.628±0.886	96.283±1.325	96.030±0.944	94.456±2.283	97.973±0.531	96.620±1.230	97.973±0.531
DenseNet121	87.624±1.673	99.463±0.270	99.327±0.608	99.183±0.828	99.437±0.848	14.330±7.396	99.437±0.848
DenseNet169	95.574±0.738	99.594±0.332	99.581±0.572	99.449±0.535	99.722±0.340	6.967±3.109	99.722±0.340
DenseNet201	93.721±1.490	99.456±0.517	99.580±0.557	99.061±0.529	99.470±0.513	8.321±7.803	99.737±0.322

Fonte: Autor

Tabela 5 – F1-Score alcançado para as imagens de profundidade e RGB-D obtidas a partir do sensor Kinect.

Depth							
Classifier / **Architecture**	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	82.269±0.997	98.218±0.867	97.147±0.594	95.452±1.007	97.996±0.784	0.976±2.282	98.127±0.830
VGG19	83.098±0.986	97.409±0.787	96.745±0.802	96.407±0.690	97.822±0.773	13.252±1.614	98.104±0.948
MobileNet	92.665±1.019	99.69±0.211	96.847±0.937	97.991±0.795	99.227±0.472	98.523±0.571	99.344±0.472
InceptionV3	85.773±1.630	93.862±0.792	94.305±1.066	96.361±0.908	96.526±0.669	4.78±4.266	96.529±0.505
Xception	92.002±1.814	98.046±0.354	96.209±1.038	97.424±0.925	98.385±0.614	14.692±2.158	98.318±0.544
ResNet50	74.349±1.878	89.939±0.405	93.157±0.282	81.738±1.332	93.803±1.011	3.86±1.712	93.503±0.614
InceptionResNetV2	87.609±1.612	95.764±0.907	94.44±1.118	95.844±0.488	96.823±1.016	13.428±3.271	96.886±1.016
NASNetMobile	82.777±1.636	92.354±1.523	92.894±1.228	91.773±1.214	96.634±0.764	95.055±1.503	96.363±0.738
DenseNet121	76.827±1.413	97.711±0.710	97.398±0.704	97.538±0.525	98.098±0.553	11.731±2.736	98.17±0.327
DenseNet169	84.766±1.907	98.251±0.743	98.005±0.811	97.825±0.582	98.488±0.243	3.21±2.725	98.624±0.373
DenseNet201	85.114±0.599	98.246±0.491	98.866±0.590	97.781±0.789	98.481±0.849	4.517±0.641	98.748±0.849
RGB-D							
Classifier / **Architecture**	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	88.137±0.916	99.275±0.644	98.726±0.584	97.165±2.004	99.579±0.568	0.807±0.552	99.579±0.568
VGG19	90.894±1.725	99.190±0.659	98.818±0.793	98.675±0.953	99.306±0.737	8.866±2.863	99.306±0.737
MobileNet	96.984±0.473	99.888±0.223	98.496±0.771	99.154±0.812	99.771±0.457	99.771±0.457	99.888±0.223
InceptionV3	96.058±1.340	97.590±0.781	98.352±0.943	98.936±0.624	98.788±0.978	3.963±3.251	98.788±0.978
Xception	95.458±1.759	99.314±0.601	98.154±1.105	98.533±1.119	99.246±0.685	9.524±7.171	99.246±0.685
ResNet50	81.546±2.297	91.985±2.173	95.841±2.206	85.215±20.573	96.528±1.570	2.386±2.276	96.658±1.649
InceptionResNetV2	92.842±2.711	97.912±0.445	97.479±1.605	98.337±1.386	98.501±0.960	10.386±1.784	98.501±0.960
NASNetMobile	89.585±1.361	95.914±1.528	95.666±1.101	94.023±2.538	97.857±0.534	96.160±1.495	97.834±0.574
DenseNet121	84.637±1.725	99.479±0.264	99.268±0.651	99.123±0.922	99.355±0.987	8.362±6.364	99.355±0.987
DenseNet169	94.123±0.881	99.582±0.341	99.584±0.558	99.392±0.599	99.682±0.390	2.197±2.264	99.682±0.390
DenseNet201	92.227±2.189	99.461±0.503	99.605±0.522	99.093±0.500	99.497±0.496	3.040±4.841	99.766±0.285

Fonte: Autor

acurácia média de apenas 5,353%. Nesse caso, é possível observar uma grande diferença entre a acurácia, tabela 4, e o F1-Score, tabela 5, indicando que o modelo apresentou dificuldades em classificar uma ou mais classes.

Partindo para a análise das imagens RGB-D, pode-se observar na tabela 4, que a combinação da arquitetura MobileNet com os classificadores kNN e SVM (RBF) apresentaram as melhores acurácias médias, atingindo um valor máximo de 99,873% e um desvio padrão mínimo de 0,253%. O mesmo também ocorre para o F1-Score, como pode ser observado na tabela 5, atingindo um valor máximo de 99,888% com desvio padrão mínimo de 0,223. Por outro lado, a combinação da arquitetura VGG16 com o classificador SVM (Poly.) apresentou a pior acurácia, com um valor máximo de apenas 4,597% e um desvio padrão de 1,191%.

Ao comparar as melhores e piores combinações de arquitetura e classificador, nos dois casos, observa-se que a diferença nos valores de acurácia pode ser explicada por diferenças nas características das arquiteturas e dos classificadores em relação aos dados de entrada. O kNN é um classificador de aprendizagem baseado em instância que pode ser bem adequado para as imagens que possuem uma distribuição de características bem definida e de fácil separação no espaço de características. Além disso, a arquitetura MobileNet é projetada para ter uma alta eficiência computacional e uma baixa complexidade de modelo, o que pode ser especialmente útil em casos onde os recursos de hardware são limitados. Já o SVM (Poly.) é um classificador mais complexo, adequado para dados que possuem uma distribuição mais complexa, mas pode ser muito sensível a parâmetros de ajuste, o que pode ter ocasionado o resultado ruim observado.

Por fim, observando a tabela 4, verifica-se que as acurácias alcançadas em quase todas as combinações para as imagens RGB-D são maiores do que as alcançadas para as imagens de profundidade, indicando que as imagens RGB-D fornecem mais informações e, portanto, tornam a tarefa de classificação mais fácil. Percebe-se também que o classificadores kNN, RF, SVM (Linear) e SVM (RBF) resultaram em bons desempenhos em conjunto com as diferentes arquiteturas de CNN, alcançando uma acurácia mínima de 90,325% para as imagens de profundidade e 92,362% para as imagens RGB-D. Ao correlacionar a acurácia com o F1-Score, de forma geral, observa-se que os valores do F1-Score acompanham a acurácia. Isso indica que a acurácia é um indicador confiável da capacidade do classificador em identificar corretamente as classes.

4.2 Análise de desempenhdo da classificação com estimador de profundidade

Nesta seção, discutimos os resultados obtidos combinando a extração de atributos utilizando CNNs e a classificação com métodos de aprendizado de máquina para os conjuntos de dados gerados pelo estimador de profundidade AdaBins (BHAT *et al.*, 2021) a partir das imagens RGB lidas pelo sensor Kinect.

Tabela 6 – Acurácia alcançada para as imagens de profundidade estimada e RGB-D obtidas a partir do estimador AdaBins.

Depth							
Classifier / Architecture	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	88.155±0.827	98.809±0.859	97.496±0.596	97.828±1.046	98.015±0.816	11.452±2.762	98.287±0.854
VGG19	78.4±0.806	97.567±0.806	97.377±0.680	96.45±0.573	98.03±0.748	10.094±1.067	98.295±0.943
MobileNet	93.175±1.011	99.672±0.211	98.148±0.843	98.201±0.748	98.943±0.452	98.422±0.490	99.212±0.452
InceptionV3	87.353±1.558	95.088±0.772	94.542±0.980	94.609±0.909	96.661±0.646	38.563±3.324	96.661±0.490
Xception	93.856±1.796	97.922±0.340	97.408±1.011	97.932±0.929	98.456±0.611	88.21±1.989	98.251±0.542
ResNet50	72.938±1.781	95.633±0.389	95.527±0.249	92.535±1.204	96.107±0.957	7.281±1.638	95.64±0.742
InceptionResNetV2	89.334±1.408	96.413±0.909	94.613±1.087	94.08±0.499	96.986±1.062	12.381±2.974	97.052±1.062
NASNetMobile	87.149±1.660	93.74±1.514	94.919±1.211	95.09±1.185	97.066±0.772	95.844±1.681	97.065±0.748
DenseNet121	88.406±1.083	97.922±0.712	97.872±0.686	97.453±0.542	98.316±0.558	5.649±2.692	98.382±0.327
DenseNet169	87.928±1.668	98.284±0.742	97.364±0.827	98.012±0.611	98.288±0.267	12.554±2.911	98.405±0.389
DenseNet201	90.389±0.646	98.662±0.471	99±0.581	98.138±0.806	98.739±0.827	8.146±0.859	98.739±0.827
RGB-D							
Classifier / Architecture	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	93.079±0.791	99.872±0.254	99.081±0.781	99.621±0.507	99.609±0.320	9.834±2.580	99.749±0.307
VGG19	84.194±1.348	99.356±0.698	99.365±0.559	98.689±1.121	99.488±0.477	7.047±3.511	99.488±0.477
MobileNet	97.460±1.288	99.870±0.258	99.744±0.313	99.326±0.434	99.473±0.480	99.617±0.507	99.744±0.313
InceptionV3	97.312±0.666	98.790±0.293	98.501±0.785	97.138±2.547	98.905±0.549	33.881±3.253	98.905±0.549
Xception	97.085±0.553	99.174±1.046	99.328±0.449	99.052±1.020	99.315±0.777	89.977±1.706	99.174±1.046
ResNet50	78.396±1.072	97.789±1.228	98.160±0.925	96.224±2.091	98.817±0.452	4.701±3.216	98.817±0.452
InceptionResNetV2	94.571±0.944	98.586±1.050	97.619±1.008	96.530±4.396	98.696±0.765	9.767±5.754	98.696±0.765
NASNetMobile	94.099±0.814	97.343±0.772	97.733±0.288	97.440±1.140	98.306±0.707	96.823±1.211	98.574±0.743
DenseNet121	96.387±1.115	99.716±0.567	99.734±0.326	99.037±1.191	99.577±0.565	4.243±1.138	99.577±0.565
DenseNet169	97.195±0.972	99.611±0.525	98.945±1.156	99.606±0.528	99.482±0.492	8.893±4.378	99.466±0.496
DenseNet201	97.457±1.792	99.859±0.281	99.731±0.329	99.463±0.658	99.736±0.324	5.719±3.119	99.736±0.324

Fonte: Autor

As tabelas 6 e 7 apresentam os valores médios e desvios padrão de Acurácia e F1-Score, respectivamente, obtidos pelas várias arquiteturas de redes neurais convolucionais (CNN) combinadas com os diferentes classificadores para a tarefa de classificação das imagens de profundidade estimada e RGB-D geradas a partir do estimador de profundidade AdaBins.

Na tabela 6, para as imagens de profundidade estimadas, é possível observar que a melhor acurácia foi obtida pela combinação entre a arquitetura MobileNet e o classificador kNN, com média de 99,672% e desvio padrão de 0,211%, indicando um alto desempenho na tarefa de classificação. Além disso, a tabela 7 apresenta um F1-Score também é muito alto, 99,667% com desvio padrão de 0,211%, sugerindo que a precisão e o recall da classificação também são muito altos. Por outro lado, na 6, a pior combinação de acurácia foi obtida com a arquitetura DenseNet121 e o classificador SVM (Poly.), com média de 5,649% e desvio padrão de 2,692%,

Tabela 7 – F1-Score alcançado para as imagens de profundidade estimada e RGB-D obtidas a partir do estimador AdaBins.

Depth							
Architecture \ **Classifier**	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	85.225±0.997	98.811±0.867	97.424±0.594	97.797±1.007	98.001±0.784	4.113±2.282	98.264±0.830
VGG19	75.232±0.986	97.548±0.787	97.285±0.802	96.398±0.690	98.003±0.773	2.976±1.614	98.285±0.948
MobileNet	93.141±1.019	99.667±0.211	98.083±0.937	98.149±0.795	98.895±0.472	98.36±0.571	99.186±0.472
InceptionV3	86.853±1.630	94.991±0.792	94.42±1.066	94.631±0.908	96.591±0.669	35.794±4.266	96.594±0.505
Xception	93.443±1.814	97.945±0.354	97.346±1.038	97.833±0.925	98.475±0.614	86.337±2.158	98.284±0.544
ResNet50	69.718±1.878	95.5±0.405	95.365±0.282	92.223±1.332	95.977±1.011	2.561±1.712	95.542±0.614
InceptionResNetV2	89.016±1.612	96.214±0.907	94.356±1.118	93.774±0.488	96.891±1.016	6.228±3.271	96.967±1.016
NASNetMobile	86.732±1.636	93.581±1.523	94.71±1.228	95.056±1.214	96.91±0.764	92.018±1.503	96.961±0.738
DenseNet121	87.263±1.413	97.85±0.710	97.81±0.704	97.342±0.525	98.262±0.553	1.393±2.736	98.334±0.327
DenseNet169	87.086±1.907	98.293±0.743	97.42±0.811	98.024±0.582	98.282±0.243	4.595±2.725	98.419±0.373
DenseNet201	90.033±0.599	98.654±0.491	99.009±0.590	98.162±0.789	98.699±0.849	2.422±0.641	98.699±0.849
RGB-D							
Architecture \ **Classifier**	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	91.304±0.707	99.874±0.250	99.007±0.851	99.552±0.600	99.584±0.340	3.402±2.388	99.718±0.344
VGG19	82.290±1.735	99.331±0.734	99.369±0.542	98.666±1.160	99.489±0.465	1.991±2.600	99.489±0.465
MobileNet	97.482±1.283	99.865±0.269	99.753±0.304	99.314±0.398	99.437±0.502	99.606±0.520	99.730±0.329
InceptionV3	97.267±0.687	98.764±0.346	98.472±0.781	97.160±2.464	98.855±0.594	29.675±2.176	98.855±0.594
Xception	96.953±0.565	99.212±0.988	99.314±0.477	98.946±1.197	99.337±0.752	88.378±1.668	99.212±0.988
ResNet50	76.467±1.291	97.673±1.242	98.112±0.910	96.146±1.886	98.765±0.491	1.583±2.765	98.765±0.491
InceptionResNetV2	94.334±1.032	98.373±1.268	97.393±1.214	96.213±4.736	98.570±0.939	4.817±3.820	98.583±0.949
NASNetMobile	93.866±0.651	97.188±0.852	97.536±0.428	97.387±1.163	98.136±0.781	96.773±1.236	98.441±0.797
DenseNet121	96.133±1.170	99.621±0.756	99.688±0.381	98.924±1.320	99.521±0.672	0.993±0.928	99.521±0.672
DenseNet169	96.698±1.121	99.625±0.503	98.989±1.141	99.595±0.538	99.474±0.501	3.145±2.347	99.474±0.501
DenseNet201	97.557±1.633	99.874±0.250	99.749±0.307	99.479±0.636	99.717±0.347	1.630±2.365	99.717±0.347

Fonte: Autor

mostrando que essa combinação teve um desempenho muito inferior em relação às outras.

Para as imagens RGB-D, geradas a partir das imagens RGB lidas pelo sensor Kinect e as imagens de profundidade estimadas pelo estimador AdaBins, pode-se verificar na tabela 6 que a melhor combinação foi obitida com o arquitetura VGG16 e o classificador kNN, com acurácia média de 99,872% e desvio padrão de 0,254%. Por outro lado, a pior combinação foi a da rede DenseNet121 com o classificador SVM (Poly.), que obteve a acurácia média mais baixa de 4,243% e desvio padrão de 1,138%.

Comparando os classificadores, nos dois casos analisados, nota-se que o classificador SVM com kernel RBF geralmente apresenta os melhores resultados em termos de acurácia e F1-Score. Esse resultado pode ser explicado pelo fato de que esse classificador é capaz de modelar de forma mais eficiente as relações complexas presentes nos dados de entrada. Outro fato observado é que a arquitetura de CNN MobileNet, apresentou bons resultados, acima de 90% para todos os classificadores.

Por fim, verifica-se que nas tabelas 6 e 7, de maneira geral, a acurácia é maior para as imagens RGB-D do que para as imagens de profundidade estimadas e os valores de acurácia e F1-Score estão positivamente correlacionados nos dois casos analisados. Percebe-se também que com exeção dos classificadores Bayes e SVM com kernel polinomial, todos os outro resultaram em bons desempenhos em conjunto com as diferentes arquiteturas de CNN, alcançando uma

acurácia mínima de 93,740% para as imagens de profundidade estimadas e 96,530% para as imagens RGB-D.

4.3 Comparação entre as abordagens de classificação

Observando os resultados expostos nas seções 4.1 e 4.2 percebe-se que as acurácias variam amplamente entre as diferentes combinações de arquitetura e classificador para cada uma das variações analisadas. Nas duas abordagens, os melholhes desempenhos formam conseguidos utilizando-se as imagens RGB-D. Para o Kinect, tabela 4, as combinações com melhor desempenho foram as dos classificadores kNN e SVM com kernel RBF combinados com a arquitetura MobileNet, com 98,873% de acurácia. Já para as imagens geradas apartir do estimador de profundidade AdaBins, tabela 6, a combinação mais precisa foi com o classificador kNN e a arquitetura VGG16, com 99,872% de acurácia.

Comparando os resultados das seções 4.1 e 4.2 constata-se que as imagens em mosaico RGB-D tiveram um melhor desempenho utilizando a abordagem de estimativa de profundidade quando comparadas à mesma abordagem utilizando o sensor Kinect. Além disso, mesmo não tendo atingido o melhor desempenho nas tabelas 6 e 7 as arquiteturas CNNs combinadas com os classificadores SVM (RBF) e SVM (Linear) atingiram altos percentuais de acurácia e F1-Score, com um valores mínimos de 98,306% e 98,136% respectivamente, para as imagens RGB-D. De forma geral, a acurácia para o conjunto de dados obtidos a partir do estimador de profundidade foi maior do que para o conjunto de dados obtidos a partir do sensor Kinect, sugerindo que os dados gerados apartir do estimador de profundidade podem ser mais discriminativos que os dados do sensor Kinect.

A tabela 8 mostra os números de atributos e tempos de extração de cada arquitetura CNN utilizada. O tempo de extração é uma métrica importante para avaliar o desempenho dos algoritmos de processamento de imagem e é medido em milissegundos (ms). Observa-se que a arquitetura MobileNet apresenta o melhor desempenho em termos de tempo de extração, com uma média de 7,999 ms para a captura de informações das imagens RGB-D utilizando a profundidade lida pelo sensor Kinect e 7,929 ms para a captura de informações das imagens RGB-D utilizando a profundidade estimada pelo estimador AdaBins. Por outro lado, a arquitetura InceptionResNetV2 teve o tempo mais longo de 77,337 ms e 78,142 ms, respectivamente. A diferença de tempo entre o melhor e o pior caso é significativa, sendo de aproximadamente 10 (dez) vezes menor, e pode ser atribuído às diferenças de complexibilidade das arquiteturas e ao

número de atributos extraídos.

Tabela 8 – Número de atributos e tempo de extração retornado para as imagens em mosaico (RGB-D)

CNN Architecture	Number of attributes	Extraction Time (ms)	
		RGB-D (Kinect)	RGB-D (AdaBins)
VGG16	512	20.455±0.518	20.432±0.529
VGG19	512	25.480±0.509	25.509±0.507
MobileNet	1024	7.999±0.845	7.929±0.453
InceptionV3	2048	34.283±0.807	34.741±1.126
Xception	2048	32.220±0.564	32.324±0.618
ResNet50	2048	25.516±0.653	25.182±0.634
InceptionResNetV2	1536	77.337±2.429	78.142±1.947
NASNetMobile	1056	39.283±6.872	41.667±6.892
DenseNet121	1024	35.026±2.109	36.560±1.857
DenseNet169	1664	52.190±2.082	53.778±2.730
DenseNet201	1920	69.671±2.453	69.347±1.925

Fonte: Autor

A tabela 9 apresenta os tempos de treinamento de cada classificador, considerando as bases de dados RGB-D, com profundidade lida (Kinect) e estimada (AdaBins). Para as imagens RGB-D (Kinect), o classificador kNN obteve o menor tempo de treinamento, 0,021s, quando combinado com a arquitetura VGG19. Já para as imagens RGB-D (AdaBins), o classificador Bayes obteve 0,015s para a mesma arquitetura, VGG19. No entanto, como analizado nas seções 4.1 e 4.2, apesar de terem conseguido uma alta taxa de acurácia (99,211% e 84,194%), essas combinações não apresentaram os melhores desempenhos de acurácia como nostrado nas tabelas 4 e 6.

Como explorado anteriormente nas seções 4.1 e 4.2, para as imagens RGB-D (Kinect) o melhor desempenho foi alcançado pela combinação da arquitetura MobileNet com os classificadores kNN e SVM (RBF) e, conforme a tabela 9, possuem tempos de treinamento de 0,068s e 3,207s, respectivamente. Já para as imagens RGB-D (AdaBins) o melhor desempenho foi alcançado pela combinação da arquitetura VGG16 com o classificador kNN e possui tempo de treinamento de 0,022s.

De ante dos resultados expostos e analisados, pode-se concluir que o uso do estimador de profundidade a partir de visão monocular é uma opção viável para a tarefa de localização e navagação em robês móveis para ambientes internos. Podendo ser utilizado, por exemplo, quando as limitações de espaço, consumo ou investimento inviabilizem o uso de sensores de profundidade.

Tabela 9 – Tempo de treino (s) para os bancos de imagens gerados (profundidade Kinect e AdaBins.

RGB-D (Profundidade Kinect)							
Classifier / **Architecture**	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	0.053±0.039	0.024±0.002	13.385±2.610	41.999±16.105	1.247±0.069	2.378±0.086	2.333±0.083
VGG19	0.028±0.003	0.021±0.002	10.990±1.437	37.558±11.337	1.201±0.063	2.323±0.086	1.304±0.049
MobileNet	0.066±0.025	0.068±0.015	7.778±1.639	123.807±49.173	2.991±0.048	2.848±0.047	3.207±0.054
InceptionV3	0.090±0.046	0.138±0.031	47.526±1.812	132.585±51.594	6.136±0.099	8.853±0.272	6.816±0.133
Xception	0.076±0.015	0.118±0.004	68.684±0.651	155.593±68.737	5.114±0.106	8.131±0.232	5.364±0.151
ResNet50	0.099±0.035	0.122±0.006	32.609±2.503	336.856±103.278	3.991±0.092	8.983±0.237	4.641±0.131
InceptionResNetV2	0.070±0.019	0.136±0.082	30.142±2.177	87.726±35.667	4.137±0.135	6.617±0.175	4.326±0.140
NASNetMobile	0.027±0.005	0.046±0.003	27.517±1.985	148.012±36.063	3.074±0.056	4.645±0.116	4.068±0.076
DenseNet121	0.038±0.000	0.047±0.001	52.723±1.890	114.085±43.417	2.603±0.055	4.447±0.129	3.030±0.056
DenseNet169	0.046±0.004	0.097±0.015	17.915±2.120	125.904±50.174	4.372±0.085	7.198±0.152	5.839±0.101
DenseNet201	0.047±0.010	0.126±0.023	36.427±2.267	143.728±56.310	4.807±0.098	8.489±0.214	5.010±0.091
RGB-D (Profundidade AdaBins)							
Classifier / **Architecture**	Bayes	kNN	RF	MLP	SVM (Linear)	SVM (Poly.)	SVM (RBF)
VGG16	0.073±0.050	0.022±0.002	8.817±2.574	23.161±5.961	1.116±0.085	2.374±0.072	1.281±0.047
VGG19	0.015±0.004	0.022±0.002	21.908±1.861	54.603±15.268	1.123±0.061	2.382±0.096	1.227±0.052
MobileNet	0.038±0.008	0.048±0.004	17.814±2.006	63.895±23.073	2.787±0.039	2.794±0.041	2.970±0.059
InceptionV3	0.089±0.030	0.141±0.051	29.249±2.729	200.724±82.824	6.152±0.138	8.939±0.236	6.409±0.128
Xception	0.139±0.035	0.157±0.075	6.925±1.690	59.063±22.269	4.570±0.098	9.002±0.224	7.065±0.143
ResNet50	0.247±0.248	0.140±0.030	40.043±21.898	219.070±69.247	3.839±0.131	8.842±0.256	5.244±0.104
InceptionResNetV2	0.078±0.028	0.090±0.007	34.564±2.257	163.053±62.614	3.941±0.048	5.733±0.122	4.453±0.062
NASNetMobile	0.040±0.010	0.057±0.024	8.457±2.022	98.642±36.013	2.589±0.029	2.780±0.057	2.720±0.061
DenseNet121	0.031±0.004	0.050±0.010	34.108±2.253	123.530±47.456	2.440±0.036	4.484±0.150	4.496±0.117
DenseNet169	0.092±0.031	0.098±0.015	6.679±1.631	38.345±12.533	4.079±0.059	7.303±0.151	4.164±0.063
DenseNet201	0.072±0.023	0.119±0.034	6.072±1.468	88.568±32.040	4.498±0.076	8.276±0.236	4.744±0.069

Fonte: Autor

5 CONCLUSÃO

Na seção 5.1 são apresentadas as conclusões obtidas apartir das análises feitas no capítulo 4. Na seção 5.2 são sugeridos caminhos para desenvolvimento de trabalhos futuros.

5.1 Conclusão

Esta dissertação apresenta uma abordagem para localização e navegação de robôs móveis em mapas topológicos de ambientes internos. O experimeto consiste em um sistema de visão computacional baseado em visão monocular e estimativa de profundidade, geradas a partir das imagens RGB capturadas. As imagens foram agrupadas, gerando uma única imagem em mosaico RGB-D. Em seguida, aplicou-se o conceito de Transfer Learning, utilizando Redes Neurais Convolucionais (Convolutional Neural Network, CNN) como extrator de características. Ao final, foi realizada uma avaliação com as combinações das arquiteturas CNN com classificadores estabelecidos na literatura, dentre eles, Naive Bayes, Perceptron Múltiplas Camadas, k-Vizinhos Mais Próximos, Floresta Aleatória e Máquina de Vetores de Suporte. Os resultados da abordagem proposta são compaprados com resultados gerados a partir de imagens de profundidade obtidas por um sensor Kinect.

A estimativa de profundidade isoladamente ou aplicada na criação de mosaicos RGB-D, provou ser um método eficiente para a tarefa de localização e navegação de robôs móveis em mapas topológicos de ambientes internos. O desempenho dos classificadores utilizando apenas a estimativa de profundidades alcançou 99,6% em Acurácia e F1-Score e 99,8% quando utilizada em mosaico RGB-D. Os tempos de processamento realizados também foram adequados para um sistema de visão computacional, com valores de 7,929ms e 0,022s, para os tempos de extração e treinamento da combinação com melhor desempenho (classificador kNN e arquitetura VGG19) utilizando imagens RGB-D (profundidade estimada).

5.2 Trabalhos Futuros

Para trabalhos futuros, ambientes externos e outros ambientes internos serão testados com esta abordagem. Além disso, podemos avaliar outros métodos de estimativa de profundidade como o MiDaS, presente em Ranftl *et al.* (2021b), o M4Depth, presente em Fonder *et al.* (2021), ou FastDepth, presente em Wofk *et al.* (2019), bem como avaliar esses métodos em abordagens SLAM como em Zhang *et al.* (2022). Podemos aplicar outras técnicas de aprendizado de

máquina como o Optimum-Path Forest (OPF), apresentado em Nunes *et al.* (2014), e ainda utilizar outras arquiteturas CNN, como ResNeXt-50 (XIE *et al.*, 2016) e CapsNet (SABOUR *et al.*, 2017).

Outro aspecto importante envolve avaliar métodos de classificação mais simples aplicando diretamente as imagens brutas, e neste contexto podemos citar os Vizinhos Mais Próximos (NN), o Classificador de Distância Mínima (MDC) e o Classificador Linear e Quadrático (CQ e LMQ).

REFERÊNCIAS

ALEXANDRE, L. A. 3d object recognition using convolutional neural networks with transfer learning between input channels. In: **IAS**. [*S. l.*: *s. n.*], 2014.

ALHASHIM, I.; WONKA, P. High quality monocular depth estimation via transfer learning. **arXiv e-prints**, abs/1812.11941, 2018. Disponível em: https://arxiv.org/abs/1812.11941.

BAZRAFKAN, S.; JAVIDNIA, H.; LEMLEY, J.; CORCORAN, P. Semiparallel deep neural network hybrid architecture: first application on depth from monocular camera. **Journal of Electronic Imaging**, SPIE, v. 27, n. 4, p. 043041, 2018. Disponível em: https://doi.org/10.1117/1.JEI.27.4.043041.

BENGIO, Y.; GOODFELLOW, I. J.; COURVILLE, A. Deep learning. **Nature**, Citeseer, v. 521, n. 7553, p. 436–444, 2015.

BHAT, S.; ALHASHIM, I.; WONKA, P. Adabins: Depth estimation using adaptive bins. **2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 4008–4017, 2021.

BREIMAN, L. Random forests. **Machine Learning**, v. 45, p. 5–32, 2004.

CAI, E.; BAIREDDY, S.; YANG, C.; CRAWFORD, M.; DELP, E. J. Deep transfer learning for plant center localization. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops**. [*S. l.*: *s. n.*], 2020.

CANZIANI, A.; PASZKE, A.; CULURCIELLO, E. **An Analysis of Deep Neural Network Models for Practical Applications**. 2017.

CHIN, W. H.; LOO, C. K.; SEERA, M.; KUBOTA, N.; TODA, Y. Multi-channel bayesian adaptive resonance associate memory for on-line topological map building. **Appl. Soft Comput.**, v. 38, p. 269–280, 2016.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 1800–1807, 2017.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: **CVPR**. [*S. l.*: *s. n.*], 2009.

DIOP, M.; ONG, L.-Y.; LIM, T. S.; HUN, L. C. A computer vision-aided motion sensing algorithm for mobile robot's indoor navigation. **2016 IEEE 14th International Workshop on Advanced Motion Control (AMC)**, p. 400–405, 2016.

DONG, X.; GARRATT, M. A.; ANAVATTI, S. G.; ABBASS, H. A. **Towards Real-Time Monocular Depth Estimation for Robotics: A Survey**. 2021.

DOSOVITSKIY, A.; BEYER, L.; KOLESNIKOV, A.; WEISSENBORN, D.; ZHAI, X.; UNTERTHINER, T.; DEHGHANI, M.; MINDERER, M.; HEIGOLD, G.; GELLY, S.; USZKOREIT, J.; HOULSBY, N. **An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale**. 2021.

DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern Classification**. 2. ed. New York: Wiley, 2001.

EIGEN, D.; PUHRSCH, C.; FERGUS, R. **Depth Map Prediction from a Single Image using a Multi-Scale Deep Network**. 2014.

ELARABY, A. F.; HAMDY, A.; REHAN, M. M. A kinect-based 3d object detection and recognition system with enhanced depth estimation algorithm. **2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)**, p. 247–252, 2018.

FARIA, J. M.; MOREIRA, A. H. J. Implementation of an autonomous ros-based mobile robot with ai depth estimation. In: **IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society**. [*S. l.*: *s. n.*], 2021. p. 1–6.

FONDER, M.; ERNST, D.; DROOGENBROECK, M. V. M4depth: A motion-based approach for monocular depth estimation on video sequences. **CoRR**, abs/2105.09847, 2021.

FU, H.; GONG, M.; WANG, C.; BATMANGHELICH, K.; TAO, D. Deep ordinal regression network for monocular depth estimation. **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**, p. 2002–2011, 2018.

GEIGER, A.; LENZ, P.; STILLER, C.; URTASUN, R. Vision meets robotics: The kitti dataset. **The International Journal of Robotics Research**, v. 32, n. 11, p. 1231–1237, 2013. Disponível em: https://doi.org/10.1177/0278364913491297.

GODARD, C.; AODHA, O. M.; BROSTOW, G. J. **Unsupervised Monocular Depth Estimation with Left-Right Consistency**. 2017.

GODARD, C.; Mac Aodha, O.; FIRMAN, M.; BROSTOW, G. J. Digging into self-supervised monocular depth prediction. October 2019.

GRUBINGER, T.; CHASPARIS, G. C.; NATSCHLäGER, T. Generalized online transfer learning for climate control in residential buildings. **Energy and Buildings**, v. 139, p. 63 – 71, 2017. ISSN 0378-7788.

GU, J.; WANG, Z.; KUEN, J.; MA, L.; SHAHROUDY, A.; SHUAI, B.; LIU, T.; WANG, X.; WANG, G. Recent advances in convolutional neural networks. **CoRR**, abs/1512.07108, 2015.

HAYKIN, S. Neural networks and learning machines. In: . [*S. l.*: *s. n.*], 2010.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 770–778, 2016.

HE, T.; ZHANG, Z.; ZHANG, H.; ZHANG, Z.; XIE, J.; LI, M. **Bag of Tricks for Image Classification with Convolutional Neural Networks**. 2018.

HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **ArXiv**, abs/1704.04861, 2017.

HUANG, G.; LIU, Z.; WEINBERGER, K. Q. Densely connected convolutional networks. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 2261–2269, 2017.

Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. **ArXiv e-prints**, fev. 2015.

JIANG, R.; YANG, S.; GE, S. S.; WANG, H.; LEE, T. heng. Geometric map-assisted localization for mobile robots based on uniform-gaussian distribution. **IEEE Robotics and Automation Letters**, v. 2, p. 789–795, 2017.

JIN, J.; DUNDAR, A.; CULURCIELLO, E. Flattened convolutional neural networks for feedforward acceleration. **CoRR**, abs/1412.5474, 2014. Disponível em: http://arxiv.org/abs/1412.5474.

JIN, Y.; YU, L.; CHEN, Z.; FEI, S. A mono slam method based on depth estimation by densenet-cnn. **IEEE Sensors Journal**, v. 22, n. 3, p. 2447–2455, 2022.

KARPATHY, A.; TODERICI, G.; SHETTY, S.; LEUNG, T.; SUKTHANKAR, R.; FEI-FEI, L. Large-scale video classification with convolutional neural networks. **2014 IEEE Conference on Computer Vision and Pattern Recognition**, p. 1725–1732, 2014.

KHAN, F.; SALAHUDDIN, S.; JAVIDNIA, H. Deep learning-based monocular depth estimation methods—a state-of-the-art review. **Sensors**, v. 20, n. 8, 2020. ISSN 1424-8220. Disponível em: https://www.mdpi.com/1424-8220/20/8/2272.

KIM, D.; GA, W.; AHN, P.; JOO, D.; CHUN, S.; KIM, J. Global-local path networks for monocular depth estimation with vertical cutdepth. **arXiv preprint arXiv:2201.07436**, 2022.

KIM, H. M.; KIM, M. S.; LEE, G. J.; JANG, H. J.; SONG, Y. M. Miniaturized 3d depth sensing-based smartphone light field camera. **Sensors**, v. 20, n. 7, 2020. ISSN 1424-8220. Disponível em: https://www.mdpi.com/1424-8220/20/7/2129.

KLUS, R.; KLUS, L.; TALVITIE, J.; PIHLAJASALO, J.; TORRES-SOSPEDRA, J.; VALKAMA, M. Transfer learning for convolutional indoor positioning systems. In: **2021 International Conference on Indoor Positioning and Indoor Navigation (IPIN)**. [*S. l.*: *s. n.*], 2021. p. 1–8.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. **Communications of the ACM**, v. 60, p. 84 – 90, 2012.

KULKARNI, M.; JUNARE, P.; DESHMUKH, M.; REGE, P. P. Visual slam combined with object detection for autonomous indoor navigation using kinect v2 and ros. In: **2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA)**. [*S. l.*: *s. n.*], 2021. p. 478–482.

KUMAR, R. H.; VANJARE, A. M.; OMKAR, S. N. Autonomous drone navigation using monocular camera and light weight embedded system. In: **2023 International Conference for Advancement in Technology (ICONAT)**. [*S. l.*: *s. n.*], 2023. p. 1–6.

KURDI, M. M.; DADYKIN, A. K.; ELZEIN, I. Navigation of mobile robot with cooperation of quadcopter. **2017 Ninth International Conference on Advanced Computational Intelligence (ICACI)**, p. 30–36, 2017.

KUZNIETSOV, Y.; STüCKLER, J.; LEIBE, B. **Semi-Supervised Deep Learning for Monocular Depth Map Prediction**. 2017.

LECUN, Y. **Generalization and network design strategies**. [*S. l.*]: Elsevier, 1989.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 5 2015. ISSN 0028-0836.

LECUN, Y.; BOSER, B. E.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W. E.; JACKEL, L. D. Handwritten digit recognition with a back-propagation network. In: **NIPS**. [*S. l.*: *s. n.*], 1989.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, IEEE, v. 86, n. 11, p. 2278–2324, 1998.

LEE, J. H.; HAN, M.-K.; KO, D. W.; SUH, I. H. From big to small: Multi-scale local planar guidance for monocular depth estimation. **ArXiv**, abs/1907.10326, 2019.

LEHTINEN, J.; MUNKBERG, J.; HASSELGREN, J.; LAINE, S.; KARRAS, T.; AITTALA, M.; AILA, T. Noise2Noise: Learning image restoration without clean data. In: DY, J.; KRAUSE, A. (Ed.). **Proceedings of the 35th International Conference on Machine Learning**. PMLR, 2018. (Proceedings of Machine Learning Research, v. 80), p. 2965–2974. Disponível em: https://proceedings.mlr.press/v80/lehtinen18a.html.

LI, Y.; SHI, C. Localization and navigation for indoor mobile robot based on ros. **2018 Chinese Automation Congress (CAC)**, p. 1135–1139, 2018.

LIN, G.; MILAN, A.; SHEN, C.; REID, I. **RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation**. 2016.

LIN, Q.; HONG, J.; LIU, Z.; LI, B.; WANG, J. Investigation into the topology optimization for conductive heat transfer based on deep learning approach. **International Communications in Heat and Mass Transfer**, v. 97, p. 103 – 109, 2018. ISSN 0735-1933.

LIN, Y.; JEON, Y. Random forests and adaptive nearest neighbors. **Journal of the American Statistical Association**, Taylor & Francis, v. 101, n. 474, p. 578–590, 2006.

LIU, F.; SHEN, C.; LIN, G.; REID, I. Learning depth from single monocular images using deep convolutional neural fields. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 38, n. 10, p. 2024–2039, 2016.

LOO, S. Y.; MASHOHOR, S.; TANG, S. H.; ZHANG, H. Deeprelativefusion: Dense monocular slam using single-image relative depth prediction. In: **2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [*S. l.*: *s. n.*], 2021. p. 6641–6648.

MA, F.; KARAMAN, S. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. **2018 IEEE International Conference on Robotics and Automation (ICRA)**, p. 1–8, 2018.

MASOUMIAN, A.; RASHWAN, H. A.; CRISTIANO, J.; ASIF, M. S.; PUIG, D. Monocular depth estimation using deep learning: A review. **Sensors**, v. 22, n. 14, 2022. ISSN 1424-8220. Disponível em: https://www.mdpi.com/1424-8220/22/14/5353.

MAZO, C.; BERNAL, J.; TRUJILLO, M.; ALEGRE, E. Transfer learning for classification of cardiovascular tissues in histological images. **Computer Methods and Programs in Biomedicine**, v. 165, p. 69 – 76, 2018. ISSN 0169-2607.

MENDES, R. de Q.; RIBEIRO, E. G.; ROSA, N. dos S.; GRASSI, V. On deep learning techniques to boost monocular depth estimation for autonomous navigation. **Robotics Auton. Syst.**, v. 136, p. 103701, 2021.

MOLCHANOV, P.; TYREE, S.; KARRAS, T.; AILA, T.; KAUTZ, J. Pruning convolutional neural networks for resource efficient transfer learning. **ArXiv**, abs/1611.06440, 2016.

MORAWSKA, B.; LIPINSKI, P.; LICHY, K.; ADAMKIEWICZ, K. Transfer learning-based uwb indoor localization using mht-mdc and clusterization-based sparse fingerprinting. **Journal of Computational Science**, v. 61, p. 101654, 2022. ISSN 1877-7503. Disponível em: https://www.sciencedirect.com/science/article/pii/S1877750322000692.

NAIR, V.; HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In: FüRNKRANZ, J.; JOACHIMS, T. (Ed.). **ICML**. Omnipress, 2010. p. 807–814. Disponível em: http://dblp.uni-trier.de/db/conf/icml/icml2010.html#NairH10.

NUNES, T.; COELHO, A.; LIMA, C.; PAPA, J.; ALBUQUERQUE, V. Eeg signal classification for epilepsy diagnosis via optimum path forest - a systematic assessment. **Neurocomputing**, p. –, 07 2014.

OQUAB, M.; DARCET, T.; MOUTAKANNI, T.; VO, H. V.; SZAFRANIEC, M.; KHALIDOV, V.; FERNANDEZ, P.; HAZIZA, D.; MASSA, F.; EL-NOUBY, A.; HOWES, R.; HUANG, P.-Y.; XU, H.; SHARMA, V.; LI, S.-W.; GALUBA, W.; RABBAT, M.; ASSRAN, M.; BALLAS, N.; SYNNAEVE, G.; MISRA, I.; JEGOU, H.; MAIRAL, J.; LABATUT, P.; JOULIN, A.; BOJANOWSKI, P. **DINOv2: Learning Robust Visual Features without Supervision**. 2023.

PAN, S. J.; YANG, Q. A survey on transfer learning. **IEEE Transactions on Knowledge and Data Engineering**, v. 22, p. 1345–1359, 2010.

PAPA, L.; ALATI, E.; RUSSO, P.; AMERINI, I. Speed: Separable pyramidal pooling encoder-decoder for real-time monocular depth estimation on low-resource settings. **IEEE Access**, v. 10, p. 44881–44890, 2022.

PATIL, V.; LINIGER, A.; DAI, D.; GOOL, L. V. Improving depth estimation using map-based depth priors. **IEEE Robotics and Automation Letters**, v. 7, n. 2, p. 3640–3647, 2022.

RAMAMONJISOA, M.; LEPETIT, V. **SharpNet: Fast and Accurate Recovery of Occluding Contours in Monocular Depth Estimation**. 2019.

RANFTL, R.; BOCHKOVSKIY, A.; KOLTUN, V. Vision transformers for dense prediction. **ICCV**, 2021.

RANFTL, R.; BOCHKOVSKIY, A.; KOLTUN, V. **Vision Transformers for Dense Prediction**. 2021.

ROCCHI, A.; WANG, Z.; PAN, Y.-J. A practical vision-aided multi-robot autonomous navigation using convolutional neural network. In: **2023 IEEE 6th International Conference on Industrial Cyber-Physical Systems (ICPS)**. [*S. l.*: *s. n.*], 2023. p. 1–6.

RODRIGUES, R. T.; MIRALDO, P.; DIMAROGONAS, D. V.; AGUIAR, A. P. **Active Depth Estimation: Stability Analysis and its Applications**. 2020.

RONNEBERGER, O.; FISCHER, P.; BROX, T. **U-Net: Convolutional Networks for Biomedical Image Segmentation**. 2015.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. **International Journal of Computer Vision (IJCV)**, v. 115, n. 3, p. 211–252, 2015.

SABOUR, S.; FROSST, N.; HINTON, G. E. Dynamic routing between capsules. **CoRR**, abs/1710.09829, 2017.

SARBOLANDI, H.; LEFLOCH, D.; KOLB, A. Kinect range sensing: Structured-light versus time-of-flight kinect. **CoRR**, abs/1505.05459, 2015. Disponível em: http://arxiv.org/abs/1505.05459.

SCHARSTEIN, D.; HIRSCHMÜLLER, H.; KITAJIMA, Y.; KRATHWOHL, G.; NEŠIĆ, N.; WANG, X.; WESTLING, P. High-resolution stereo datasets with subpixel-accurate ground truth. In: JIANG, X.; HORNEGGER, J.; KOCH, R. (Ed.). **Pattern Recognition**. Cham: Springer International Publishing, 2014. p. 31–42. ISBN 978-3-319-11752-2.

SILBERMAN, N.; HOIEM, D.; KOHLI, P.; FERGUS, R. Indoor segmentation and support inference from rgbd images. In: **ECCV**. [*S. l.*: *s. n.*], 2012.

SILVA, S. P. P. da. **Localização e Navegação de Robôs Móveis em Mapas Topológicos utilizando Redes Neurais Convolucionais baseada em Transfer Learning**. Dissertação (Mestrado) – INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO CEARÁ, PRÓ-REITORIA DE PESQUISA, PÓS-GRADUAÇÃO E INOVAÇÃO, PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO, Fortaleza/CE, 2019.

SILVA, S. P. P. da; ALMEIDA, J. S.; OHATA, E. F.; RODRIGUES, J. J.; ALBUQUERQUE, V. H. C. de; FILHO, P. P. R. Monocular vision aided depth map from rgb images to estimate of localization and support to navigation of mobile robots. **IEEE Sensors Journal**, v. 20, p. 12040–12048, 2020.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **CoRR**, abs/1409.1556, 2014. Disponível em: http://arxiv.org/abs/1409.1556.

SONG, Z.; JIANG, G. yi; HUANG, C. A survey on indoor positioning technologies. In: . [*S. l.*: *s. n.*], 2011.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. **The Journal of Machine Learning Research**, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

SUH, H. K.; IJSSELMUIDEN, J.; HOFSTEE, J. W.; HENTEN, E. J. van. Transfer learning for the classification of sugar beet and volunteer potato under field conditions. **Biosystems Engineering**, v. 174, p. 50 – 65, 2018. ISSN 1537-5110.

SZEGEDY, C.; IOFFE, S.; VANHOUCKE, V.; ALEMI, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In: **AAAI**. [*S. l.*: *s. n.*], 2017. p. 4278–4284.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S. E.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. **2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 1–9, 2015.

SZEGEDY, C.; VANHOUCKE, V.; IOFFE, S.; SHLENS, J.; WOJNA, Z. Rethinking the inception architecture for computer vision. **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 2818–2826, 2016.

TAHA, A. K.; CANSEVER, G. Robot localization in rgb-d images using pca and cnn. In: **2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)**. [*S. l.*: *s. n.*], 2021. p. 684–687.

TAN, C.; SUN, F.; KONG, T.; ZHANG, W.; YANG, C.; LIU, C. A survey on deep transfer learning. In: **Artificial Neural Networks and Machine Learning - ICANN 2018 - 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part III**. [*S. l.*: *s. n.*], 2018. p. 270–279.

TAN, M.; LE, Q. EfficientNet: Rethinking model scaling for convolutional neural networks. In: CHAUDHURI, K.; SALAKHUTDINOV, R. (Ed.). **Proceedings of the 36th International Conference on Machine Learning**. PMLR, 2019. (Proceedings of Machine Learning Research, v. 97), p. 6105–6114. Disponível em: https://proceedings.mlr.press/v97/tan19a.html.

TANG, H.; SCAIFE, A. M. M.; LEAHY, J. P. Transfer learning for radio galaxy classification. **Monthly Notices of the Royal Astronomical Society**, v. 488, p. 3358–3375, 2019.

THEODORIDIS, S.; KOUTROUMBAS, K. D. Pattern recognition, fourth edition. In: . [*S. l.*: *s. n.*], 2008.

TOSI, F.; ALEOTTI, F.; POGGI, M.; MATTOCCIA, S. **Learning monocular depth estimation infusing traditional stereo knowledge**. 2019.

TROUVé, P.; CHAMPAGNAT, F.; BESNERAIS, G. L.; SABATER, J.; AVIGNON, T.; IDIER, J. Passive depth estimation using chromatic aberration and a depth from defocus approach. **Appl. Opt.**, Optica Publishing Group, v. 52, n. 29, p. 7152–7164, Oct 2013. Disponível em: https://opg.optica.org/ao/abstract.cfm?URI=ao-52-29-7152.

ULRICH, L.; VEZZETTI, E.; MOOS, S.; MARCOLIN, F. Analysis of rgb-d camera technologies for supporting different facial usage scenarios. **Multimedia Tools and Applications**, v. 79, p. 29375–29378, Oct 2020. Disponível em: https://doi.org/10.1007/s11042-020-09479-0.

VAPNIK, V. N. The nature of statistical learning theory. In: **Statistics for Engineering and Information Science**. [*S. l.*: *s. n.*], 2000.

WANG, C.; DONG, S.; ZHAO, X.; PAPANASTASIOU, G.; ZHANG, H.; YANG, G. Saliencygan: Deep learning semisupervised salient object detection in the fog of iot. **IEEE Transactions on Industrial Informatics**, v. 16, p. 2667–2676, 2020.

WANG, Q.; DU, P.; YANG, J.; WANG, G.; LEI, J.; HOU, C. Transferred deep learning based waveform recognition for cognitive passive radar. **Signal Processing**, v. 155, p. 259 – 267, 2019. ISSN 0165-1684.

WEISS, K. R.; KHOSHGOFTAAR, T. M.; WANG, D. A survey of transfer learning. **Journal of Big Data**, v. 3, p. 1–40, 2016.

WOFK, D.; MA, F.; YANG, T.; KARAMAN, S.; SZE, V. Fastdepth: Fast monocular depth estimation on embedded systems. **CoRR**, abs/1903.03273, 2019.

XIE, S.; GIRSHICK, R. B.; DOLLÁR, P.; TU, Z.; HE, K. Aggregated residual transformations for deep neural networks. **CoRR**, abs/1611.05431, 2016.

XIE, S.; GIRSHICK, R. B.; DOLLÁR, P.; TU, Z.; HE, K. Aggregated residual transformations for deep neural networks. **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, p. 5987–5995, 2017.

XU, D.; WANG, W.; TANG, H.; LIU, H.; SEBE, N.; RICCI, E. Structured attention guided convolutional neural fields for monocular depth estimation. **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**, p. 3917–3925, 2018.

YOKOYAMA, K.; MORIOKA, K. Autonomous mobile robot with simple navigation system based on deep reinforcement learning and a monocular camera. In: **2020 IEEE/SICE International Symposium on System Integration (SII)**. [*S. l.*: *s. n.*], 2020. p. 525–530.

ZHANG, D.; HAN, J.; LI, C.; WANG, J.; LI, X. Detection of co-salient objects by looking deep and wide. **International Journal of Computer Vision**, v. 120, p. 215–232, 2016.

ZHANG, Y.; WU, T.; ZHAO, H.; DU, S.; YU, L.; WANG, X. Pseudo depth maps for rgb-d slam. In: **2022 International Conference on High Performance Big Data and Intelligent Systems (HDIS)**. [*S. l.*: *s. n.*], 2022. p. 168–172.

ZHOU, T.; BROWN, M.; SNAVELY, N.; LOWE, D. G. Unsupervised learning of depth and ego-motion from video. In: **2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [*S. l.*: *s. n.*], 2017. p. 6612–6619.

ZOPH, B.; LE, Q. V. Neural architecture search with reinforcement learning. **ArXiv**, abs/1611.01578, 2017.

ZOPH, B.; VASUDEVAN, V.; SHLENS, J.; LE, Q. V. **Learning Transferable Architectures for Scalable Image Recognition**. 2018.