UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Vinícius Barbosa Henrique

Extração de edificações por *Deep Learning* e combinação de dados LiDAR e imagens ópticas

Presidente Prudente

2024

Vinícius Barbosa Henrique

Extração de edificações por *Deep Learning* e combinação de dados LiDAR e imagens ópticas

Dissertação apresentada ao Programa de Pós Graduação em Ciências Cartográficas (PPGCC) na Faculdade de Ciências e Tecnologia (FCT) da Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), como parte dos requisitos para a obtenção do título de Mestre em Ciências Cartográficas.

Orientador: Prof. Dr. Maurício Galo

Coorientador: Prof. Dr. Milton Hirokazu Shimabukuro

Presidente Prudente

2024

H519e

Henrique, Vinícius Barbosa
Extração de edificações por Deep Learning e combinação de dados LiDAR e imagens ópticas / Vinícius Barbosa Henrique. -- Presidente Prudente, 2024
95 p. : il., tabs., fotos, mapas

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Faculdade de Ciências e Tecnologia, Presidente Prudente
Orientador: Maurício Galo
Coorientador: Milton Hirokazu Shimabukuro

1. Aprendizado profundo. 2. Extração de edificações. 3. LiDAR. 4. Modelos de cores. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Ciências e Tecnologia, Presidente Prudente. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

IMPACTO[1] ESPERADO DESTA PESQUISA

O tema central desta Dissertação é a extração de edificações urbanas por meio da combinação de modelos digitais de superfície normalizados e imagens ópticas usando aprendizado profundo. Essa abordagem aprimora a qualidade da identificação de edificações, permitindo a atualização eficiente de informações urbanas, úteis para o planejamento e gestão das cidades, detecção de expansões e uso do solo.

POTENTIAL IMPACT OF THIS RESEARCH

The central theme of this Dissertation is the extraction of urban buildings through the combination of normalized digital surface models and optical images using deep learning. This approach enhances the quality in building identification, allowing the efficient updating of urban information for city planning and management, including the detection of urban expansion and land use.

[1] Informação inserida de acordo com a Portaria Unesp nº 117, de 21 de dezembro de 2022.

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Presidente Prudente

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Extração de edificações por *Deep Learning* e combinação de dados LiDAR e imagens ópticas

AUTOR: VINÍCIUS BARBOSA HENRIQUE
ORIENTADOR: MAURICIO GALO
COORIENTADOR: MILTON HIROKAZU SHIMABUKURO

Aprovado como parte das exigências para obtenção do Título de Mestre em Ciências Cartográficas, área: Aquisição, Análise e Representação de Informações Espaciais pela Comissão Examinadora:

Prof. Dr. MAURICIO GALO (Participaçao Presencial)
Departamento de Cartografia / Faculdade de Ciencias e Tecnologia de Presidente Prudente

Prof. Dr. HIDEO ARAKI (Participaçao Virtual)
Departamento de Geomática / Universidade Federal do Paraná

Prof. Dr. NILTON NOBUHIRO IMAI (Participaçao Presencial)
Departamento de Cartografia / Faculdade de Ciencias e Tecnologia de Presidente Prudente

Presidente Prudente, 06 de janeiro de 2023

Dedico este trabalho ao Prof. Dr. Amilton Amorim *(in memoriam)*.
Obrigado por ter me orientado no início desta jornada.

AGRADECIMENTOS

Agradeço aos meus orientadores, Prof. Dr. Maurício Galo e Prof. Dr. Milton Hirokazu Shimabukuro, pela paciência, sugestões e principalmente por acreditarem e terem me motivado.

Agradeço ao Programa de Pós Graduação em Ciências Cartográficas (PPGCC) e a UNESP pela oportunidade de desenvolver este projeto, e aos meus amigos que fiz durante este período em Presidente Prudente.

Agradeço aos meus familiares pelo apoio e amor, que me ajudaram a finalizar esta pesquisa.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

A extração de edificações a partir de imagens ópticas constitui um importante passo para o planejamento urbano e desenvolvimento territorial para as cidades, e os processos automáticos, como o *machine learning* e *deep learning*, beneficiam indiretamente os gestores públicos no contexto de processos decisórios. O *deep learning* é uma subárea do aprendizado de máquina, que se concentra no treinamento de redes neurais profundas com enfoque no aprendizado com base em dados com alta variabilidade, como luminosidade, posição, textura e outros, e difere de outras técnicas por utilizar redes neurais na aprendizagem e envolver o uso de múltiplas camadas de neurônios artificiais para realizar tarefas complexas de processamento de dados. Há diversas redes disponíveis na literatura para os mais variados fins, incluído o contexto de segmentação de edificações em área urbana, ou redes inicialmente designadas para uma finalidade que apresenta compatibilidade em aprendizado de edificações (como a *U-Net* desenvolvida para segmentação em imagens da área médica). Independente da rede considerada, é comum variações com implementações de módulos e novos processos, como ocorre na *ResUNet-a* baseada na *U-Net*, com o objetivo de aprimorar sua segmentação. A melhoria nos resultados dessas redes também pode ser produzida pela entrada de informações adicionais, modificando os dados usados no treinamento para melhorar a identificação de um objeto, como o uso de dados LiDAR (*Light Detection And Ranging*) e diferentes modelos de cores, por exemplo. A presente pesquisa se propõe a avaliar o efeito da modificação dos dados de entrada nas redes de *deep learning*, com uso de dados LiDAR e composição de cores, na arquitetura *ResUNet-a*, e testar a hipótese de aprimoramento da segmentação. Para avaliar a hipótese foi realizado um experimento prático visando testar o uso da rede *ResUNet-a*, sendo possível atingir 96.0% de correspondência na segmentação semântica de edificações pela métrica F1-Score com um modelo de parâmetros treinado em 60 épocas durante o período de 4 dias. A partir disso, a metodologia foi proposta para avaliar o desempenho utilizando dois *datasets*: *HInDSM* (composto por informação de *matiz*, intensidade e *MDSn* - modelo digital de superfície normalizado) e imagem RGB, usado para efeito de comparação. O resultado pelo *HInDSM* alcançou 96,601% de F1-Score, correspondendo a um acréscimo de 1,89% se comparado ao uso apenas da imagem RGB. Além disso, o modelo apresenta melhorias qualitativas na segmentação de edificações, e conclui-se que o uso de *MDSn* combinado com os componentes matiz (*hue*) e intensidade, do modelo de cor HSI trazem melhorias na acurácia e desempenho na identificação de edificações em meio urbano.

Palavras-chave: Aprendizado Profundo. Extração de edificações. LiDAR. Modelo de Cores.

ABSTRACT

The extraction of buildings from optical images is an important step for urban planning and territorial development for cities, and automated processes, such as machine learning and deep learning, indirectly benefit public decision-makers. Deep learning is a subfield of machine learning that focuses on training deep neural networks with an emphasis on data-driven learning with high variability, such as brightness, position, texture, and others. It differs from other techniques in using neural networks in learning and involving the use of multiple layers of artificial neurons to perform complex data processing tasks. There are several networks available in the literature for various purposes, including the context of building segmentation in urban areas, or networks initially designed for a purpose that is compatible with building learning (such as the *U-Net* developed for segmentation in medical images). Regardless of the network considered, variations with module implementations and new processes are common, as seen in the *ResUNet-a* based on *U-Net*, aiming to improve its segmentation. Improved results in these networks can also be achieved by introducing additional information, modifying the training data to enhance object identification, such as the use of Light Detection and Ranging (LiDAR) data and different color models, for example. This research aims to evaluate the effect of modifying input data in deep learning networks, using LiDAR data and color composition in the *ResUNet-a* architecture, and test the hypothesis of segmentation improvement. To evaluate the hypothesis, a practical experiment was conducted to test the use of the *ResUNet-a* network, achieving a 96.0% correspondence in semantic building segmentation by the *F1-Score* metric with a parameter model trained in 60 epochs over a period of 4 days. Subsequently, the methodology was proposed to evaluate performance using two datasets: *HInDSM* (comprising hue, intensity, and normalized digital surface model (*nDSM*) information) and *RGB* images, used for comparison. The result for *HInDSM* reached a 96.601% *F1-Score*, representing an increase of 1.89% compared to using only *RGB* images. Additionally, the model presents qualitative improvements in building segmentation, and it is concluded that the use of *nDSM* combined with hue and intensity components from the *HSI* color model brings improvements in accuracy and performance in the identification of buildings in urban environments.

Keywords: Deep learning. Building Extraction. LiDAR. Color Models.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CLS	- Classificador
CUDA	- *Compute Unified Device Architecture*
cuDNN	- *CUDA Deep Neural Network library*
DL	- *Deep Learning*
DSM	- *Digital Surface Model*
DTM	- *Digital Terrain Model*
FN	- *False Negative*
FP	- *False Positive*
FPN	- *Feature Pyramid Network*
GIS	- *Geographic Information System*
GPU	- *Graphics Processing Unit*
GSD	- *Ground Sample Distance*
HInDSM	- Composição hue, intensidade e MDSn
HSV	- *Hue, Saturation e Value*
IHS	- *Intesidade, Hue e Saturação*
IoU	- *Intersection Over Union*
LASER	- *Light Amplification by Stimulated Emission of Radiation*
LiDAR	- *Light Detection And Ranging*
MCC	- *Matthews Correlation Coefficient*
MDS	- Modelo Digital de Superficie
MDSn	- Modelo Digital de Superficie Normalizado
MDT	- Modelo Digital de Terreno
mIoU	- *Mean Intersection Over Union*
nDSM	- *Normalized Digital Surface Model*

OA	- *Overall acurracy*
RBF	- *Radial Basis Function*
REG	- Regressão
ReLu	- *Rectified Linear Unit*
RGB	- *Red, Green e Blue*
RPN	- *Region Proposal Networks*
TN	- *True Negative*
TP	- *True Positive*
VRAM	- *Video Random Access Memory*
Wcov	- *Weighted By Coverage*

CÓDIGOS

SUMÁRIO

1 INTRODUÇÃO

A extração automática de edificações em meio urbano não é uma necessidade recente, e tem se tornado cada vez mais essencial no cotidiano administrativo das cidades, como no planejamento urbano e desenvolvimento territorial. Para as cidades, a representação detalhada da complexidade territorial é um tópico relevante e de grande importância por fomentar a resolução de questões de direito ao solo, permitir implantação de políticas públicas direcionadas e, dessa forma, melhorar as condições de vida do cidadão. Todavia, segundo Sohn e Dowman (2007) extrações automáticas não atingem 100% de sucesso por algumas razões, como, por exemplo: a complexidade nas cenas, oclusões e dependência dos sensores. Como consequência, pesquisas sobre esse assunto são importantes meios para melhorar essas segmentações e reduzir a necessidade de extrações manuais.

Há uma gama de metodologias para a extração automática de feições a partir de dados obtidos por sensores remotos, com o uso de algoritmos de aprendizagem de máquina ou de classificadores lineares, que podem ser aplicados a dados distintos. Duas informações importantes nesse sentido são imagens de sensores ópticos, por permitirem ampla visão da superfície terrestre, e dados LiDAR advindos de sistemas de varredura *LASER* (*Light Amplification by Stimulated Emission of Radiation*), sendo esse amplamente utilizado por ser uma fonte de aquisição de nuvens de pontos com coordenadas tridimensionais em alta densidade. Apesar da alta resolução desses produtos, a segmentação manual em imagens de sensores ópticos para extração de objetos urbanos possui complexidades relacionadas à extração perfeita e exaustiva por um operador humano, e o desenvolvimento computacional em *hardwares* e linguagens de programação envolvidos nesses processos não acompanharam o aumento da oferta de imagens de alta resolução, contexto já observado por (WILKINSON, 2005). O ritmo de mudanças também é um fator de relevância no espaço urbano, devido às cidades estarem em constantes processos de alterações, requerendo procedimentos rápidos de modo a permitir constantes atualizações. Ao se tratar da extração em dados LiDAR, tem-se que essa também pode ser conduzida por algoritmos computacionais ou manualmente, filtrando os objetos de interesse na nuvem de pontos ou aplicando algoritmos que identificam uma geometria específica, e segmenta esses dados.

Yuan (2018) diz que em 2006 o interesse em pesquisas envolvendo redes neurais de aprendizagem profunda, ou *deep learning*, aumenta com investigações voltadas para a criação de algoritmos capazes de aprenderem uma determinada função, como reconhecimento facial ou diagnósticos em saúde, com processos de funcionamento semelhante ao do neurônio humano. Segundo Yuan (2018), o uso de *deep learning* surge, nesse aspecto da extração de informações, como uma alternativa aos classificadores lineares, por permitir a inclusão da aprendizagem profunda em imagens ópticas e de dados LASER, transferindo o foco da extração dos objetos urbanos para a eficácia das redes neurais em aprender, detectar e segmentar, e introduzindo o

conceito de generalização ilimitada como habilidade em distinguir objetos em sua diversidade de cores e formas.

Há uma variedade de redes que podem ser enquadradas na segmentação semântica de edificações, como a *U-Net* e *ResUNet-a* que utilizam poucas imagens para treino, com enfoque na definição das bordas dos objetos. Portanto, a aplicação de algoritmos de *deep learning* para a extração de edificações em áreas urbanas torna-se promissora devido a esse ambiente possuir características com alta variabilidade entre si, como a diversidade de coberturas de edificações, com a presença de materiais distintos, em diferentes formas e condições físicas. Apesar dessa possibilidade, os dados usados para segmentação semântica em ambiente urbano ainda são majoritariamente imagens aéreas com composição de cores em RGB (*Red, Green e Blue*), por ser um dado com maior disponibilidade. Como as redes neurais permitem a inserção de dados distintos, o aprendizado com outras composições ou a adição de outras informações podem impactar nos resultados apresentados pelas redes. Desse modo, este trabalho tem por objetivo avaliar o uso combinado de dados LiDAR e imagens ópticas no modelo de cor HSI (Hue, Saturação e Intensidade) para o aprimoramento dos resultados das redes neurais, com enfoque na extração de edificações em ambiente urbano.

1.1 Hipótese

Este trabalho avaliou o uso combinado de imagens ópticas com diferentes modelos de cor e modelos de elevação de superfície normalizados, advindos de dados *LiDAR*, para a extração de edificações com uso da rede neural convolucional *ResUNet-a*. Estima-se que a rede neural usada permita a identificação e segmentação semântica desses objetos urbanos pelo uso combinado desses dados como camadas de entrada na rede. Desse modo, a hipótese colocada é: “A acurácia e desempenho na segmentação semântica de edificações em meio urbano, por técnicas de *deep learning*, são aprimorados pela combinação de imagens ópticas e dados *LiDAR*”.

1.2 Objetivos

1.2.1 Objetivo geral

Avaliar a extração de edificações por redes neurais convolucionais, inserindo a combinação de modelos digitais de superfície normalizados advindos de dados *LiDAR* e imagens ópticas representadas nos modelos de cores IHS e RGB.

1.2.2 Objetivos específicos

Como objetivos específicos tem-se que:

- Construir, treinar e gerar um modelo por rede neural convolucional a partir da combinação de dados *LiDAR* e imagens ópticas em diferentes modelos de cor, e aplicá-lo para segmentação semântica de edificações em área urbanizada;
- Comparar a segmentação semântica obtida, com a resultante de imagens sem transformação (*RGB*) e dados *LiDAR*.

2 *DEEP LEARNING* E REDES NEURAIS CONVOLUCIONAIS

2.1 *Deep learning*

O aprendizado de máquina, ou *machine learning* é essencial para o entendimento claro do conceito sobre *deep learning*. Segundo Goodfellow, Bengio e Courville (2016, p. 96), o aprendizado de máquina é "uma forma de estatística aplicada com maior ênfase no uso de computadores, para estimar estatisticamente funções complicadas, e uma menor ênfase na comprovação do intervalo de confiança ao redor dessas funções". Desse modo, o aprendizado de máquina consegue reconhecer padrões e alcança um aprendizado contínuo em predições, com a habilidade de promover ajustes aprimorando seus resultados.

Há duas categorias ditas como principais para a temática: aprendizado supervisionado e não-supervisionado. O aprendizado não-supervisionado é tratado quando há a necessidade de processamento de um conjunto de dados sem o uso explícito de rótulos pré-definidos, portanto ocorre quando não se tem um modelo que se comporte como instrutor — ou seja, que instrua o processamento — para indicar durante o aprendizado. O supervisionado se contrapõe ao anterior por associar um rótulo para cada propriedade que se quer aprender (GOODFELLOW; BENGIO; COURVILLE, 2016). Há possibilidades de outras abordagens, como um aprendizado semi-supervisionado, quando houver o interesse em rotular um ou mais alvos, mantendo os demais sem rotulação. A condição mais comum de aprendizado de máquina, segundo LeCun, Bengio e Hinton (2015), é o aprendizado supervisionado. Nesse modo, as categorias recebem valores probabilísticos como saída do processamento, e o maior valor indica a categoria prevista pela rede, conforme o aprendizado na etapa de treinamento.

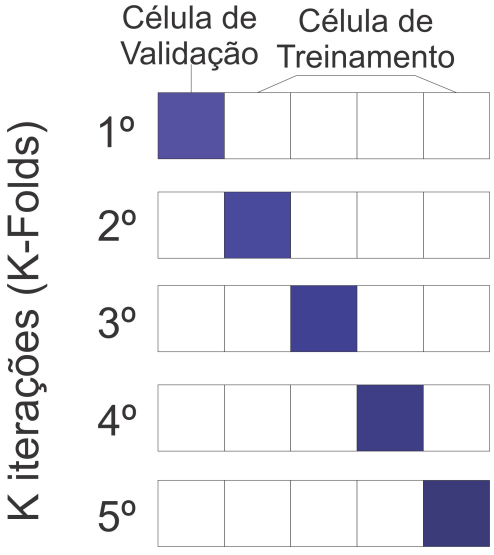
O aprendizado em um sistema de *deep learning* envolve um processo iterativo de treinamento, em que parâmetros devem ser ajustados. Em um sistema comum de *deep learning*, a quantidade de parâmetros pode ser de milhões. De todo modo, os algoritmos de *machine learning* devem conseguir generalizar, ou seja, manter um ótimo desempenho de predições em dados novos, previamente não observados pela rede e, para avaliar essa capacidade, deve-se computar o erro de generalização, e aplicar estratégias para minimizar o erro (GOODFELLOW; BENGIO; COURVILLE, 2016). A avaliação pode ser feita em um conjunto de dados de teste, coletados separadamente do conjunto de dados de treinamento, para a seleção garantir que o conjunto de dados difira, seja novo e desconhecido pela rede.

Pode-se avaliar o desempenho de um algoritmo, segundo Goodfellow, Bengio e Courville (2016), pelo valor do erro obtido em seu treinamento, e pela diferença entre o erro do treinamento e o erro de teste, sendo esses erros definidos em: *erro de teste* é o erro da generalização obtido no conjunto de dados novos e não vistos; o *erro de treinamento* é estimado selecionando amostras do conjunto de dados de treinamento visando avaliar sua capacidade generalizar no conjunto

conhecido. Deve-se buscar sempre os menores valores nos erros elencados, evitando o *overfitting* e o *underfitting*. *Overfitting* ocorre quando um modelo de aprendizado de máquina é treinado excessivamente a partir dos dados de treinamento, a ponto de começar a capturar ruídos e flutuações aleatórias em vez dos padrões e relações subjacentes. *Underfitting* é o oposto de *overfitting* e ocorre quando um modelo de aprendizado de máquina é muito simples, ou não é treinado adequadamente a partir dos dados de treinamento, resultando no fato que o modelo não se ajusta bem aos dados de treinamento e tem um desempenho reduzido tanto nos dados de treinamento quanto nos demais dados (não vistos pela rede) (GOODFELLOW; BENGIO; COURVILLE, 2016).

Há também metodologias para a validação dos dados, sendo os mais utilizados segundo Kohavi (1995) o *holdout*, *k-fold* e *leave-one-out*. O método *holdout* é feito dividindo o *dataset*, podendo ser em igual proporção, sendo uma parte para treinamento e outra para teste. O método de *k-fold* é conhecido como validação cruzada, por dividir o *dataset* em k subconjuntos de mesma dimensão, sendo um subconjunto aplicado na validação e os subconjuntos restantes usados no cálculo da acurácia do modelo, como indicado na Figura 2.1. O método de *leave-one-out* é derivado do método de *k-fold*, sendo o k é igual a N, ou seja, o número total de dados, e seu cálculo é feito N vezes para cada dado, com alto custo computacional.

Figura 2.1 – Metodologia de avaliação *k-fold*.



Fonte: Transfer (2021).

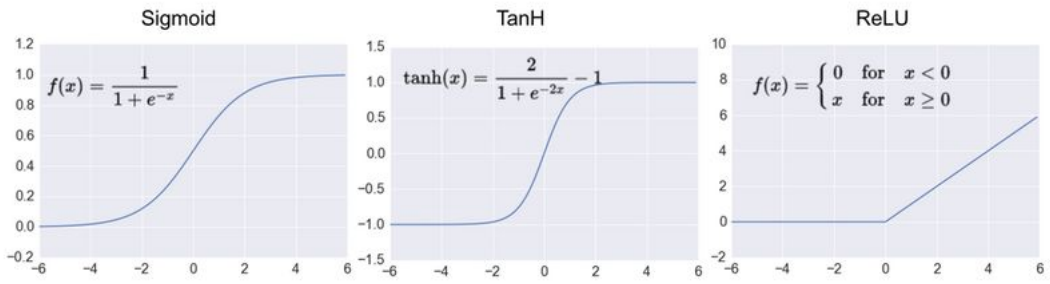
A função de ativação tem como princípio definir se a informação recebida pelo neurônio é relevante para ser transmitida, sendo as principais funções apresentadas por Goodfellow, Bengio e Courville (2016) as seguintes *ReLu (rectified linear unit), Sigmoid, Softmax, TanH* (tangente hiperbólica), *RBF (Radial Basis Function), Softplus, Hard tanh, Absolute value rectification e Maxout*. Dentre essas, a função de ativação mais usada atualmente entre as redes neurais convolucionais é a ReLu devido ao desempenho computacional, e na Figura 2.2 pode-se

observar a representação gráfica de algumas das funções citadas (GOODFELLOW; BENGIO; COURVILLE, 2016).

A função custo é uma função que calcula a diferença entre a saída desejada e a saída obtida, usada posteriormente para ajustar os pesos de uma rede. O objetivo é sempre minimizar essa diferença, aproximando a saída obtida da saída desejada. Dessa forma, pode-se considerar, por exemplo, que uma tarefa de reconhecimento de sons de animais pode ser efetuada com alto desempenho na identificação por um determinado algoritmo, mas esse mesmo algoritmo não tem o mesmo desempenho ao reconhecer sons de automóveis.

É importante destacar que as redes neurais profundas têm desempenhos diferentes em conjunto de dados distintos, por isso não é possível estimar a melhor arquitetura (número e tipos de camada, funções de ativação e função custo) que sirvam para todas as aplicações (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figura 2.2 – Representação gráfica de algumas funções de ativação, no qual x é o valor de entrada para o neurônio.



Fonte: Moujahid (2016).

O uso da função de ativação *ReLu* em camadas ocultas aprimorou o desempenho das redes em comparação com a função *Sigmoid* (GOODFELLOW; BENGIO; COURVILLE, 2016). É uma função que retorna os mesmos valores dos inseridos diretamente, ou 0 quando a entrada for 0 ou menor que 0, sendo uma função não-linear por sempre ter a saída zerada para entradas negativas.

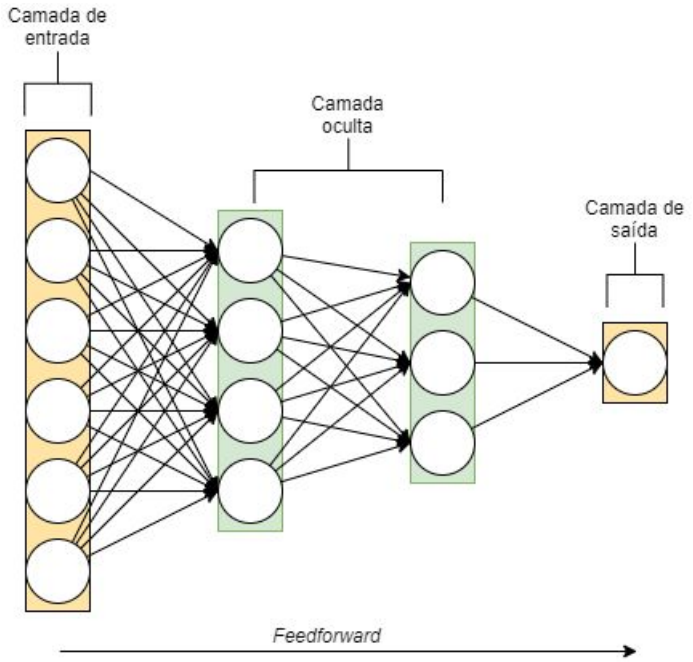
A necessidade de um classificador que seja sensível a informações relevantes, e insensível a informações irrelevantes (como as variações de luminosidade, posição e fundos em uma fotografia), não é recente. Desde 1960, segundo LeCun, Bengio e Hinton (2015), sabe-se que os classificadores lineares (como a regressão linear) conseguem atuar apenas em regiões simples e pequenas, não sendo eficientes em identificar variações nos dados. Há a possibilidade de uso de ferramentas genéricas não-lineares, mas não sendo essas efetivas na generalização fora da abrangência dos dados de treino. Dessa forma, para solucionar esses problemas, essas ferramentas podem ser substituídas por boas arquiteturas com capacidade de aprendizado, como as existentes em *deep learning*.

Em 2006 ressurge o interesse em um campo de pesquisas da área de aprendizagem de

máquina, o *deep learning*, e nos últimos anos trouxe avanços no processamento de informações em larga escala em diversas áreas do conhecimento. Segundo Deng e Yu (2014), pode-se definir o *deep learning* como "um campo do aprendizado de máquina baseado em aprender níveis de representação, sendo os níveis subsequentes definidos a partir dos níveis inferiores, e os níveis inferiores podem auxiliar a definir os níveis superiores", com os níveis sendo as camadas de abstração nas diferentes arquiteturas. A aplicação do *Deep learning* é diversa, abrangendo desde reconhecimento de voz, até a detecção e classificação de objetos, ou pixel em imagens. Além disso, a capacidade de resolução de problemas tem-se tornado cada vez maior com o passar dos anos, devido ao avanço de redes neurais profundas (LECUN; BENGIO; HINTON, 2015).

O uso de *deep learning* para classificação se diferencia dos outros classificadores (como os lineares) por possuir redes neurais para aprendizagem. Este fato implica em que, objetos poderiam ser classificados mesmo que houvesse variabilidade entre si, como forma ou luminosidade, porque a rede aprende a diferenciar além do que ocorre em outros classificadores. Seu diferencial é ter neurônios artificiais em redes, que avançam de uma camada a outra (*feedforward* - Figura 2.3) com a informação dos pesos da camada inferior enviada para uma função não-linear, como a ReLU que têm por característica o rápido aprendizado em arquiteturas com muitas camadas (LECUN; BENGIO; HINTON, 2015). Devido ao fato dessas arquiteturas apresentarem consideráveis quantidades de camadas, ou seja, de profundidade, é dado o nome de *deep learning*.

Figura 2.3 – *Feedforward* e camadas em uma rede neural.



Fonte: adaptado de Goodfellow, Bengio e Courville (2016).

2.2 Redes Neurais Convolucionais

O funcionamento do neurônio humano é a inspiração para as redes neurais artificiais, projetadas para simular a forma rápida e paralela de funcionamento do cérebro humano, similar ao modo que o cérebro processa informações. As redes neurais convolucionais são assim chamadas por possuírem camadas de convolução. A aplicação da convolução é baseada na leitura de dados, segundo Goodfellow, Bengio e Courville (2016), em grade, assim como a composição das imagens matriciais. Dessa forma, ao usar a convolução em redes neurais, inclui-se na rede a operação matemática de convolução.

A maior motivação, descrita por Goodfellow, Bengio e Courville (2016), para aplicação da convolução é devido a três melhorias proporcionadas: interações esparsas, compartilhamento de parâmetros e representação equivariante. As interações esparsas ocorrem pela redução do *kernel* da rede na entrada dos dados, permitindo que uma imagem que possua milhões de pixels possa ser armazenada em poucos parâmetros, reduzindo sua ocupação a uma centena de pixels. Tratando do compartilhamento de parâmetros, temos que ele implica no aproveitamento de parâmetros pelo modelo usando-a diversas vezes, sendo que em uma rede neural sem convolução esses parâmetros são aplicados apenas uma vez. A representação equivariante significa que, ao entrar com dados diferentes, a resposta da rede será alterada no mesmo sentido, ou seja, apresentará uma saída diferente proporcional à entrada.

Outra camada comumente encontrada em redes convolucionais são as de *pooling*, normalmente posicionadas após as camadas de convolução. Essa camada é responsável por reduzir as informações em uma região, a depender da dimensão do *pooling* (GOODFELLOW; BENGIO; COURVILLE, 2016). Em uma aplicação de *max pooling* (quando o valor máximo da região é considerado) com dimensão 2×2, ao inserir dado com dimensão 36×36 teremos, após o *pooling*, a saída de um dado com dimensão 18×18. Em uma região com valores $[17, 19; 24, 19]$ o *max pooling* 2×2 escolheria o valor 24 para representar aquela região (Figura 2.4). Dessa forma, as camadas posteriores receberão menos parâmetros do que as camadas anteriores receberam.

Figura 2.4 – Exemplo de aplicação de camada de *max pooling*.

Fonte: adaptado de Intel (2017)

2.3 Arquiteturas para segmentação de edificações

Uma arquitetura, no contexto de *deep learning*, pode ser entendida como um conjunto de instruções, considerando a codificação, organizado hierarquicamente em camadas, com funções de ativação e custo. Há inúmeras arquiteturas para os mais variados fins, para diversas categorias de dados de entrada. Para o caso de segmentação semântica de imagens ópticas, um levantamento feito nas revistas *ISPRS Journal of Photogrammetry and Remote Sensing* e *Remote Sensing*, e no portal *IEEE Xplorer* e em indexadores como Portal de Periódicos Capes e Google Acadêmico, selecionando como tema da busca as redes neurais com enfoque em segmentação de edificações em ambiente urbano, é mostrado um alto número de publicações recentes sobre o assunto. Apesar de ser possível encontrar arquiteturas testadas exclusivamente para edificações, há estudos com redes aplicadas em outros fins testadas para segmentação semântica de edificações.

As redes aqui elencadas Tabela 2.1 foram avaliadas no quesito segmentação de edificações, selecionando as métricas expostas pelos autores. Nesse sentido, identificaram-se métricas comumente usadas, entre elas: *mIoU* (*mean intersection over union*) e *IoU* (*intersection over union*), *Wcov* (*weighted by coverage)*, *F1-Score*, *AO* (*overall accuracy*), *Recall*, *Precision* e *MCC* (*matthews correlation coefficient*). Por não ser factível compatibilizar todas as métricas e compará-las sem a construção de uma matriz de confusão, esses dados não foram considerados decisivos na escolha das arquiteturas. A disponibilidade das arquiteturas em repositórios, como o *GitHub*, também foi um fator para classificação dos artigos, já que algumas redes possuíam módulos complexos sem o acesso público dos algoritmos. Sete artigos não informaram o repositório para *download* do *software*.

A categoria de publicação também foi considerada, já que muitas arquiteturas são publicadas nos formatos *preprint* e em conferências, e ainda não foram publicadas em periódicos revisados. A razão para isso é a necessidade de publicação rápida pelos autores para garantir a originalidade, por ser uma área do conhecimento com desenvolvimento rápido de pesquisas e avanços. De todo modo, as publicações em *preprint* foram desconsideradas na seleção dos artigos devido ao alto risco em não se terem a verificação por pares. Dos doze selecionados, seis foram publicados em conferência. A opção pela remoção dos artigos de conferência causaria um impacto alto nessa revisão, pela presença de artigos como a ResNet, que possui mais de 47 mil citações (informação obtida no banco de dados SCOPUS) e foi publicado em *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

Considerando os artigos presentes na Tabela 2.1, duas redes se destacam para análises: o primeiro selecionado trata-se da ResUNet-a (DIAKOGIANNIS et al., 2020), considerando sua disponibilidade, ano de publicação, estar publicado em periódico revisado, utilizar MDSn (modelo digital de superfície normalizado) na composição dos dados e possuir alto índice em segmentação de edificações entre as redes levantadas. A outra rede é *U-Net* (RONNEBERGER; FISCHER; BROX, 2015), por ser uma rede usada como base para diversas arquiteturas, e com alto desempenho em segmentação de edificações. As duas arquiteturas mencionadas serão

explicitadas nas próximas seções.

Tabela 2.1 – Redes selecionadas para a revisão.

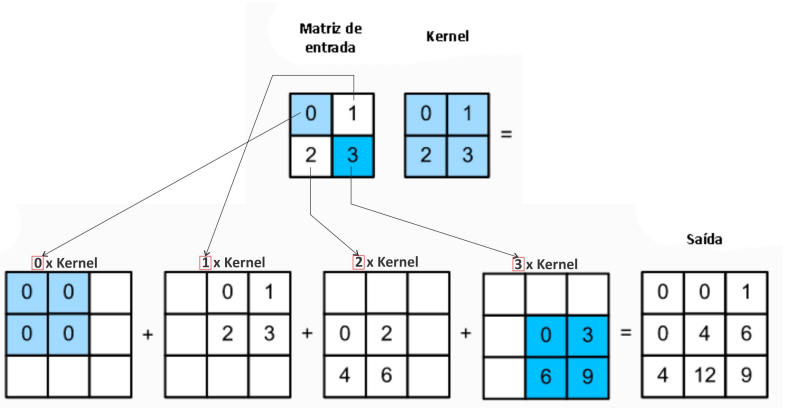
Denominação da rede	Tipo de Publicação	Tipo de Segmentação Testado	Tipo de dado	Disponibilidade	Citações (SCOPUS/Jul 2021)
DARNet (2019)	Conferência	Edificações	Imagens RGB	Não citado	18
DeepLabv3+ (2018)	Conferência	Edificações	Imagens RGB	Github	552
EaNet (2020)	Periódico	Paisagem urbana geral	Imagens RGB	Github	3
EuNet (2019)	Periódico	Edificações	Imagens RGB	Não citado	9
GRRNet (2019)	Periódico	Edificações	NIR+RGB-MDSn	Github	52
MA-FCN (2020)	Periódico	Edificações	Imagens RGB	Não citado	13
Polygon-RNN (2017)	Conferência	Outros	Imagens RGB	Não citado	101
PolyMapper (2019)	Conferência	Edificações	Imagens RGB	Não citado	19
Resnet (2016)	Conferência	Outros	Imagens RGB	Github	43164
ResUNet-a (2020)	Periódico	Paisagem urbana geral	RGB+NIR+MDSn	Github	50
SegNET (2017)	Conferência	Paisagem urbana geral	Imagens RGB	Não citado	4135
SRI-NET (2019)	Periódico	Edificações	Imagens RGB	Não citado	35
Mask R-CNN (2017)	Conferência	Paisagem urbana geral	Imagens RGB	Github	5984
U-NET (2015)	Conferência	Imagens médicas	Imagens RGB	Github	15814

Na Tabela 2.1, as referências para os artigos são: DARNet (CHENG et al., 2019), DeepLabv3+ (CHEN et al., 2018), EaNet (ZHENG et al., 2020), EuNet (KANG et al., 2019), GRRNet (HUANG et al., 2019), MA-FCN (WEI; JI; LU, 2020), Polygon-RNN (CASTREJON et al., 2017), PolyMapper (LI; WEGNER; LUCCHI, 2019), ResNet (HE et al., 2016), ResUNet-a (DIAKOGIANNIS et al., 2020), SegNET (BADRINARAYANAN; KENDALL; CIPOLLA, 2017), SRI-NET (SHI; LI; ZHU, 2019), Mask R-CNN (HE et al., 2017) e U-NET (RONNEBERGER; FISCHER; BROX, 2015).

2.3.1 Arquitetura U-Net

A arquitetura U-Net é uma rede neural completamente convolucional que trabalha com poucas imagens de treino e proporciona alta precisão em segmentação semântica. Essa rede foi criada em 2015 para uso na área médica com pouca ou reduzida quantidade de imagens de treino. Sua execução é baseada em camadas consecutivas com operações de *max pooling* e de *up-convolution* — ou seja, em sentido inverso de uma camada de convolução — importante para aumentar a resolução de saída (RONNEBERGER; FISCHER; BROX, 2015). O processo de *up-convolution*, ou convolução transposta, ocorre pela multiplicação da matriz de entrada por um *kernel*, visando reamostrar uma camada para uma dimensão maior que a anterior, como indicado na Figura 2.5.

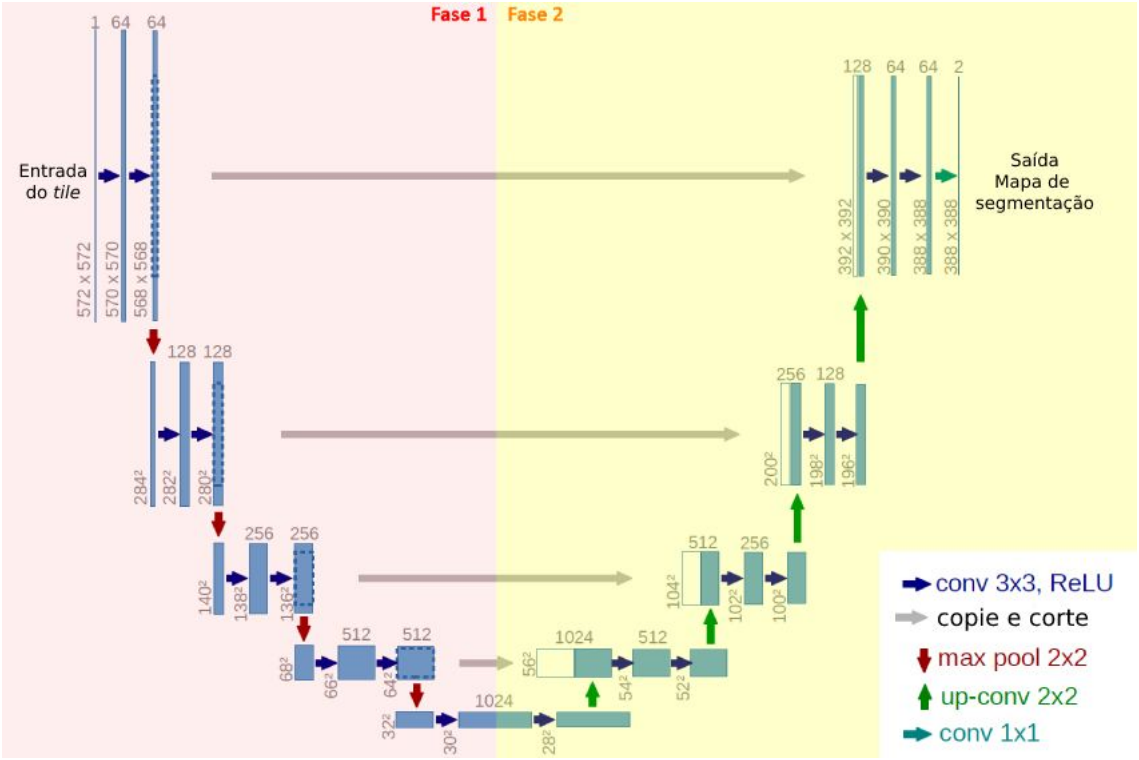
Figura 2.5 – Operação de *up-convolution*.



Fonte: adaptado de Mishra (2020).

Ronneberger, Fischer e Brox (2015) explicam que seu funcionamento pode ser resumido em duas fases — como apresentado na Figura 2.6 — uma de redução e uma camada de ativação *ReLU* seguida por *max pooling*, e outra de expansão. Uma fase de redução (Fase 1) ocorre com a aplicação de duas convoluções 3×3, seguidas por uma camada ReLU e uma camada de *max pooling* 2×2, com quatro repetições, e ao final de cada redução, o número de canais é dobrado. Na fase de expansão — Fase 2 — o processo ocorre em sentido inverso, com uma camada de *up-convolution* 2 x 2 que dobra os canais, seguida por duas camadas convolucionais 3 x 3 e uma camada ReLU. Nesse ponto, é necessário um recorte na camada de redução correspondente ao encontrado na Fase 1 com o mesmo número de canais, realizado antes das camadas convolucionais, necessário para uma concatenação do mapa de características. Ao final, a rede aplica uma última camada convolucional 1 x 1 e, no total, a rede possui 23 camadas convolucionais.

Figura 2.6 – Arquitetura U-Net.



Fonte: Ronneberger, Fischer e Brox (2015).

Em seus estudos, Ronneberger, Fischer e Brox (2015) concluíram haver um ótimo desempenho da arquitetura *U-Net* para aplicações biomédicas com baixo número de imagens de treino. Sobre as imagens de satélite com alta resolução, Freudenberg et al. (2019) demonstram que o rendimento na detecção de palmeiras foi alto, com precisão entre 89% e 92%. Estudos com redes baseadas nessa arquitetura, como a apresentada por Wang, Zhang e Dai (2020) com um aprimoramento da arquitetura, resultaram em uma precisão próxima a 84%, 2% superior à *U-Net* sem os aprimoramentos.

Segundo Ahmed, Mahbub e Rahman (2020), os modelos atuais mais sofisticados de extração de edificações, estradas e telhados são baseados na arquitetura *U-Net*. Observam-se também demonstrações do uso dessa arquitetura nos desafios promovidos pela iniciativa *SpaceNet*, com a *U-Net* sempre entre as mais rápidas e eficientes, como o visto no desafio de extração de rede viária *Automated Road Network Extraction and Route Travel Time Estimation from Satellite Imagery* — SN5 (ETTEN, 2020).

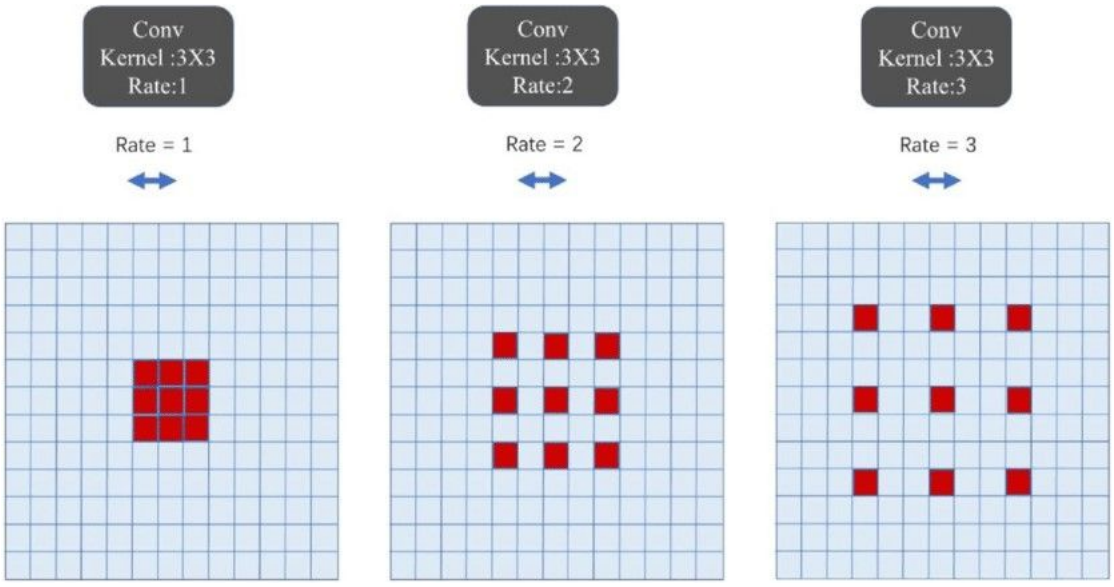
2.3.2 Arquitetura *ResUNet-a*

A *ResUNet-a* é uma rede totalmente convolucional para a segmentação semântica, que combina diversos módulos avançados de *deep learning* em uma única arquitetura, com a introdução de uma nova função de custo específica para a segmentação semântica de classes desbalanceadas, ou seja, classes com recorrências variadas. Sua arquitetura é baseada em camadas

de blocos residuais modificados (*ResBlock-a*), em uma forma de *encoder-decoder* semelhante à presente na *U-Net* (DIAKOGIANNIS et al., 2020). De fato, a *ResUNet-a* possui o mesmo princípio de funcionamento da *U-Net*, vista na Subseção 2.3.1.

A *ResUNet-a* combina diversos módulos e princípios recorrentes na segmentação semântica de imagens ópticas, como elencado por Diakogiannis et al. (2020), sendo eles: paradigma de *encoder-decoder* como na *U-Net*, para garantir uma transição gradual das imagens para as máscaras de segmentação; blocos residuais modificados para resolver os problemas de explosão/desaparecimento de gradientes; módulo *Atrous Convolutions* para aplicação de múltiplas convoluções dilatadas como apresentado na Figura 2.7, cujo objetivo é garantir um melhor desempenho na identificação das correlações em diferentes pontos de uma mesma imagem; a camada *pooling piramidal*, para ter informações de contexto em segundo plano e melhorar o desempenho; e por último, a rede possui quatro tarefas complementares enquanto está aprendendo: máscara de segmentação, identificação de bordas comuns entre as máscaras, transformação da distância, e a última é a reconstrução da imagem no espaço de cores IHS (intensidade, hue e saturação).

Figura 2.7 – *Atrous Convolutions*, em que *Rate* é a taxa de dilatação da convolução.



Fonte: Jiao et al. (2019).

A rede é apresentada em duas arquiteturas diferentes, *ResUNet-a d6* e *ResUNet-a d7*, diferenciando-se na profundidade (uma com seis camadas, e outra com sete camadas). Na prática, a última camada no modelo d7 possui o dobro de filtros que o modelo d6 e, na comparação feita pelos autores, conclui-se que o modelo d6 possui melhor desempenho na tarefa de segmentação individual de edificações, vegetação baixa e árvore, ao comparar com o d7, mas possui menor desempenho quando avaliado na tarefa de segmentação para todas as classes (DIAKOGIANNIS et al., 2020).

O autor também propõe o uso do coeficiente de *Tanimoto* (Equação 2.1) com a complementação $T(1-p_i, 1-l_i)$ como função custo para a rede, onde $p \equiv \{p_i\}, p_i \in [0,1]$ representa o vetor de probabilidades para o i-ésimo pixel, e $l \equiv \{l_i\}$ corresponde ao vetor com os rótulos verdadeiros, onde $l_i \in [0,1]$ para vetores binários. Essa função custo convergiu rapidamente para resultados satisfatórios quando comparada com as funções custo *Tanimoto* sem complementação. A função foi generalizada para aplicações com múltiplas classes (Equação 2.2), onde w_j são os pesos por classes J, p_{iJ} é a probabilidade do pixel i pertencer à classe J, e l_{iJ} é o rótulo do pixel i associado à classe J.

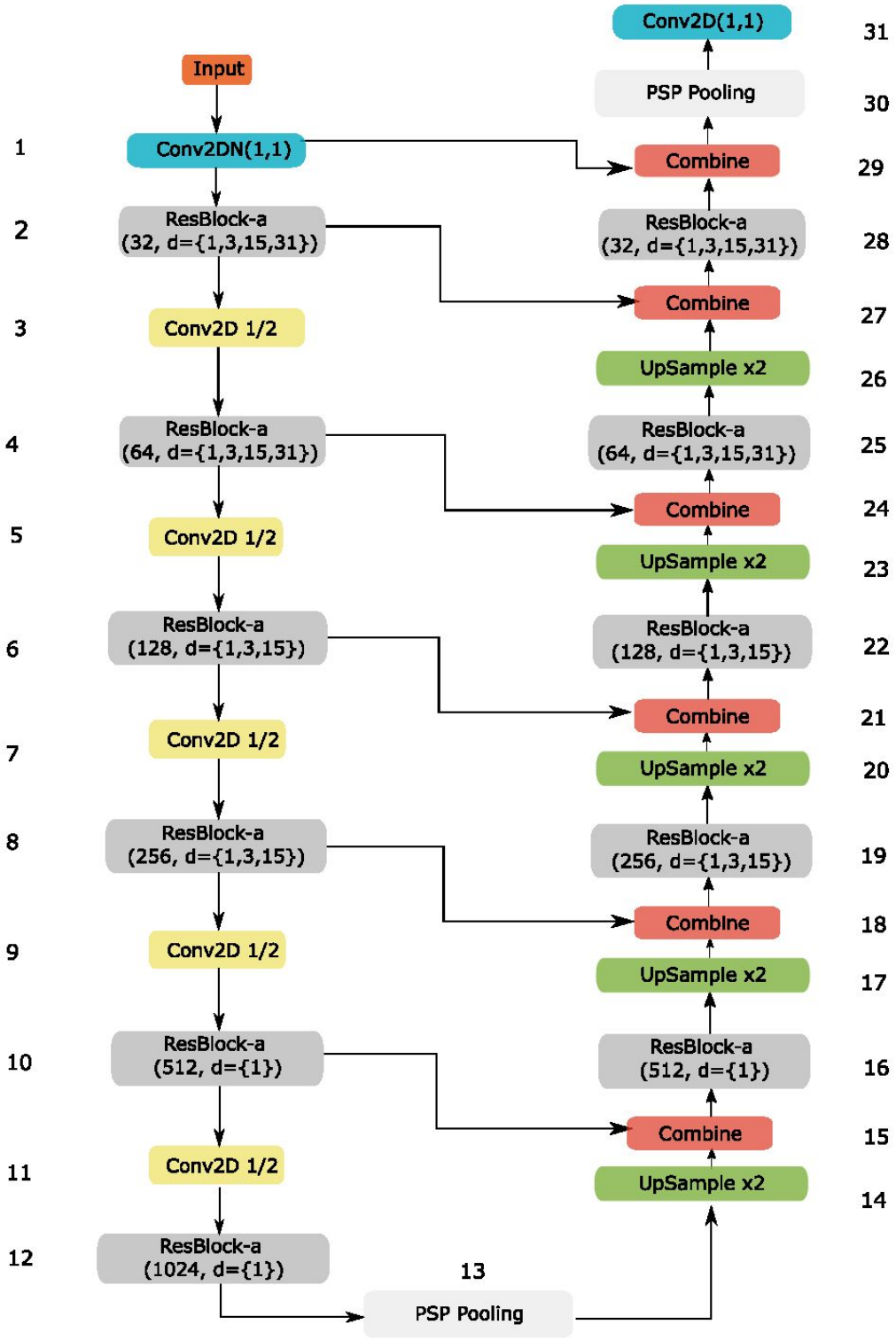
$$\widetilde{T}(p_i, l_i) = \frac{T(p_i, l_i) + T(1-p_i, 1-l_i)}{2} \tag{2.1}$$

$$T(p_i, l_i) = \frac{\sum_{J=1}^{N_{class}} w_J \sum_{i=1}^{N_{pixel}} p_{iJ} l_{iJ}}{\sum_{J=1}^{N_{class}} w_j \sum_{i=1}^{N_{pixel}} (p_{iJ}^2 + l_{iJ}^2 - p_{iJ} l_{iJ})} \tag{2.2}$$

A *ResUNet-a d6* é apresentada na Figura 2.8, sendo composta por uma fase de *encoder* na esquerda e, na face direita, o *decoder*. A cada convolução, é introduzido um bloco residual *ResBlock-a* (Figura 2.9), com a quantidade de filtros iniciada em 32 e incrementada até 1024, em que cada bloco possui múltiplas taxas de dilatação - ou múltiplas camadas de *atrous convolutions*. No bloco residual, cada *tile* avança sobre uma camada de normalização, uma camada de função de ativação *ReLU* e uma convolução bidimensional, seguido por mais uma camada de normalização, função de ativação e convolução. Entre o *encoder-decoder* e, no final, há uma camada de *PSP Pooling*, em que o dado de entrada é dividido em quatro partes iguais, aplicando uma camada de *max-pooling*. A estratégia é extrair um mapa de características em porções diferentes da camada de entrada (nas proporções 1/1, 1/2, 1/4 e 1/8) que auxilie na segmentação, baseando-se em uma informação global (DIAKOGIANNIS et al., 2020). Isso é concatenado ao final e aplicada mais uma convolução.

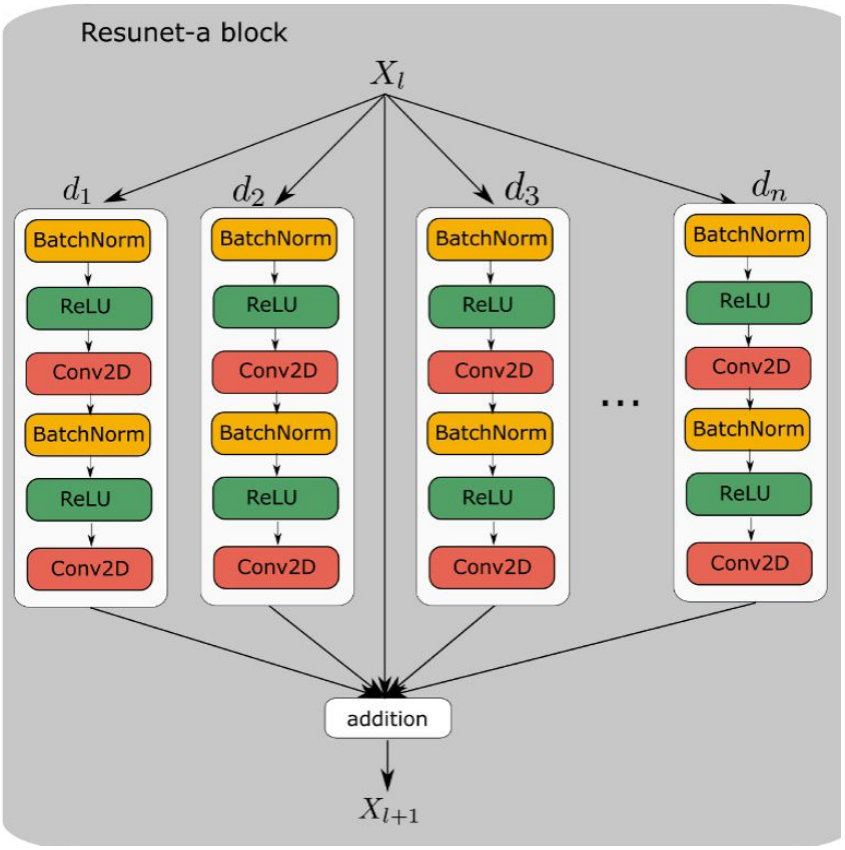
Os testes da rede no *dataset* de *Potsdam*, que podem ser obtidos a partir de ISPRS-WG-III/4 (2018), demonstraram que os dois modelos (*d6* e *d7*) apresentam *overall accuracy* (OA) acima de 90%, com *F1-Score* mínimo de 91,6%. O melhor desempenho apresentado entre as seis classes analisadas (superfície impermeável, carros, edificações, vegetação rasteira, *background* e árvores) é na segmentação de edificações, com *F1-Score* de 97,1%.

Figura 2.8 – Arquitetura ResUNet-a.

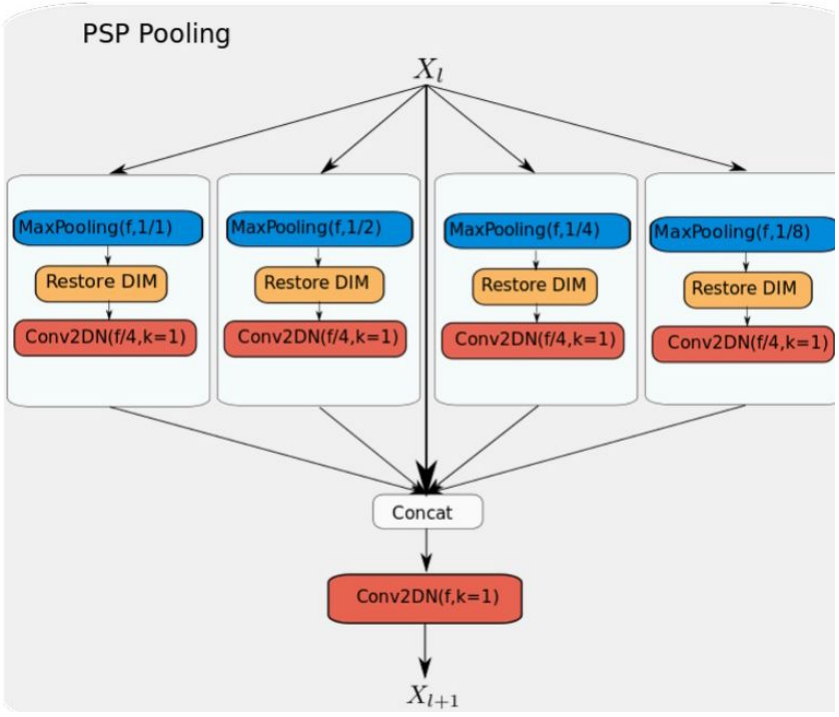


Fonte: Diakogiannis et al. (2020).

Figura 2.9 – Bloco residual da ResUNet-a.



(a) Bloco residual.



(b) PSPooling da ResUNet-a.

Fonte: Diakogiannis et al. (2020).

3 LASER E SISTEMAS LIDAR

Os sistemas de varredura e o perfilamento a LASER conseguem coletar uma grande quantidade de dados tridimensionais com alta precisão, sendo uma excelente fonte de dados (SHAN; TOTH, 2008). Sua origem remonta à década de 60, com aplicações terrestres iniciais no meio militar, e foi posteriormente adaptado para plataformas aéreas. Segundo Wehr e Lohr (1999, p. 69), "ao compararmos com a técnica de radar de micro-ondas, o LASER é vantajoso por ter pulsos com alta energia e curto intervalo de tempo, e a sua luz tem comprimento de onda comparativamente curto, sendo bem colimada usando pequenas aberturas".

Um sistema LiDAR é composto, segundo El-Sheimy, Valeo e Habib (2005), por três tecnologias: GNSS, sistema de navegação inercial e LASER. Ele é entendido como um sensor ativo de sensoriamento remoto, independente de luz solar e passível de obtenção 24 horas por dia. O LiDAR trouxe um grande impacto na construção de modelos digitais de terreno, permitindo a reconstrução tridimensional de todos os objetos presentes no terreno. Em comparação com outras fontes de obtenção de dados, Beraldin, Blais e Lohr (2010) mencionam que o LASER possui luz bem direcionada, monocromática, brilhante e coerente, e essas características facilitam sua distinção da luz solar.

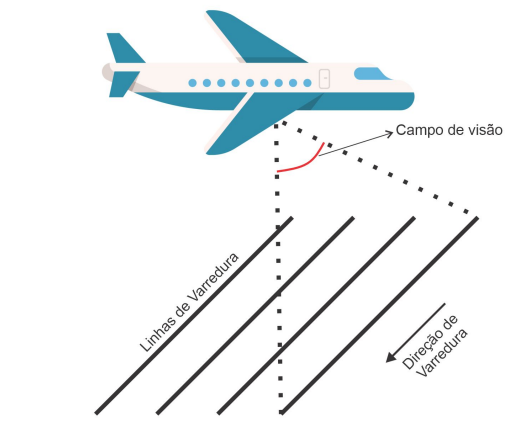
3.1 Princípios

Pode-se definir o LASER como um elemento de medição remota, que extrai informações em formato de nuvem de pontos, sendo cada ponto relacionado com uma incerteza (BERALDIN; BLAIS; LOHR, 2010). Segundo Shan e Toth (2008), quando embarcado em uma plataforma aérea, o LASER consegue descrever a paisagem a partir da emissão de pulsos de radiação direcionados para a superfície terrestre, que sofrem reflexão e retornam à unidade LASER. Pelo intervalo de tempo entre a emissão e o retorno, é encontrada a distância ao objeto, e sua precisão está diretamente relacionada à medida do tempo, sendo praticável devido à precisão conhecida da velocidade da luz, sendo a maior fonte de erro relacionada à medição do tempo.

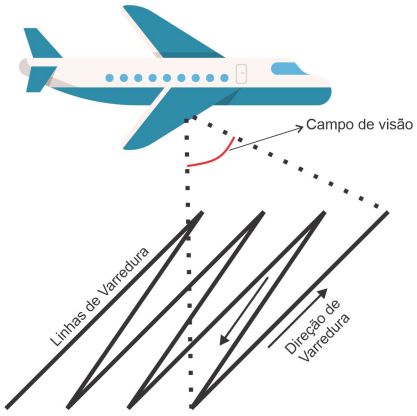
O LASER também pode ser emitido por um feixe contínuo de radiação, e o intervalo de tempo é medido pela comparação entre o padrão de ondas sinusoidais e a diferença de fase entre eles, sendo esses os dois princípios de medição descritos por Wehr e Lohr (1999): *Time-of-flight ranging* e princípio de medida por diferença de fase entre o sinal transmitido e recebido. De todo modo, existem algumas técnicas envolvidas na varredura com sistemas LiDAR, com vantagens e desvantagens para diferentes aplicações. El-Sheimy, Valeo e Habib (2005) citam quatro métodos: espelho de rotação constante (Figura 3.10a) — que produz linhas paralelas no solo — com principal desvantagem de não haver observações a cada rotação do espelho; oscilação de espelho (Figura 3.10b) — que produz linhas em *zig-zag* — e sua desvantagem é devida à variação

da velocidade e aceleração que provoca distorções no espaçamento dos pontos; conjunto de fibra-óptica (Figura 3.10c) — parte do princípio da transmissão do pulso em um ângulo fixo — e possui a desvantagem de ter um campo de visão (*footprint*) pequeno; e o último é o escâner elíptico (Figura 3.10d) — que utiliza dois espelhos para rotacionar o LASER de forma elíptica — sendo as principais desvantagens o aumento da complexidade por usar dois espelhos e por dobrar as incertezas da localização do ponto.

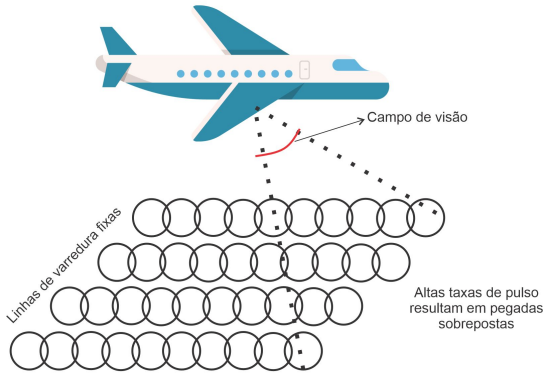
Figura 3.10 – Técnicas de varredura LiDAR.



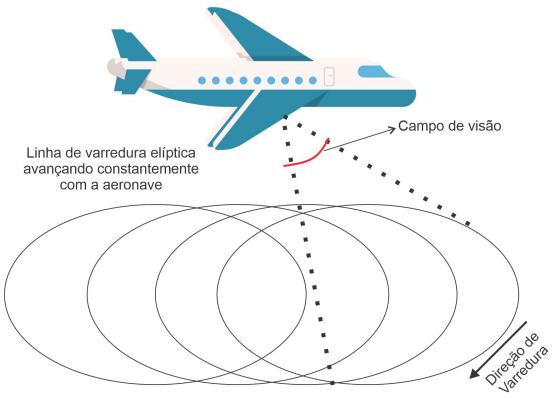
(a) Espelho de rotação constante.



(b) Oscilação de espelho.



(c) Conjunto de fibra-óptica.



(d) Escâner elíptico.

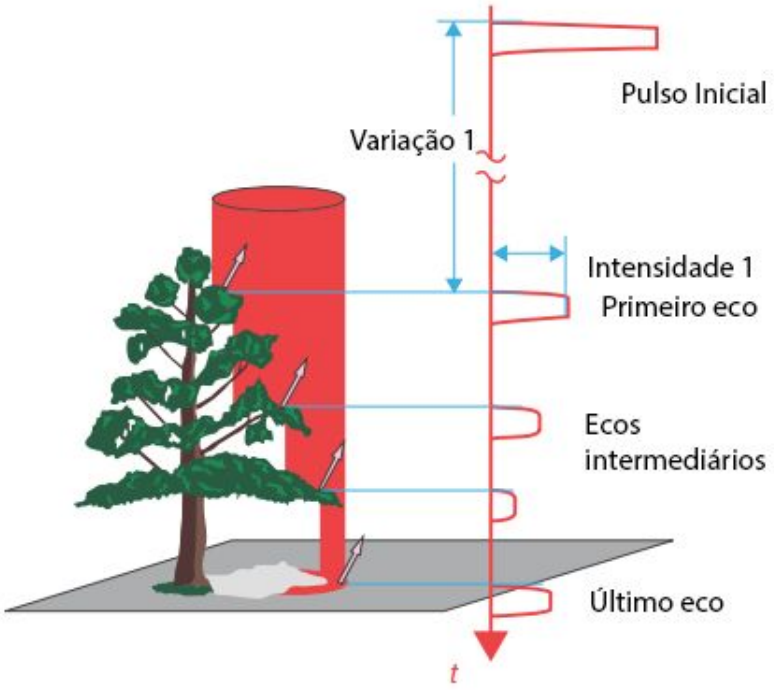
Fonte: adaptado de El-Sheimy, Valeo e Habib (2005).

O comprimento de onda do feixe luminoso de operação do LASER embarcado em aeronaves está entre 800 nm e 1550 nm, e a escolha do LASER apropriado deve considerar

o objeto a ser escaneado, como a varredura em regiões cobertas por neve que possuem baixa reflexão no comprimento de onda de 1550 nm (BERALDIN; BLAIS; LOHR, 2010). Há também o LASER batimétrico com feixes nas faixas de 1064 nm e 532 nm (infravermelho e verde), sendo o verde ideal para a penetração em águas e o infravermelho para a delimitação do início da superfície aquática, com funcionamento análogo aos aerotransportados destinados a levantamentos topográficos (NASCIMENTO, 2019).

Pode-se também ter múltiplos ecos, dependendo do tipo e da orientação da superfície de incidência do LASER. Isso resulta na possibilidade de o receptor receber múltiplos retornos de um mesmo pulso emitido, podendo cada retorno ter diferentes intensidades. Essa situação ocorre, por exemplo, quando um pulso atinge a borda de um telhado e o solo, retornando ao receptor dois sinais. Um dos principais responsáveis por esse efeito é o tamanho do *footprint*, determinado pelo diâmetro de abertura do feixe luminoso e altura da plataforma sobre o terreno, e quanto maior o *footprint*, maior a possibilidade de ocorrer os múltiplos ecos. Na Figura 3.11, pode-se constatar a incidência de um feixe em uma árvore com diferentes ecos de retorno e em diferentes intensidades.

Figura 3.11 – Múltiplos ecos em um pulso.



Fonte: adaptado de Beraldin, Blais e Lohr (2010).

3.1.1 Modelos digitais de superfície

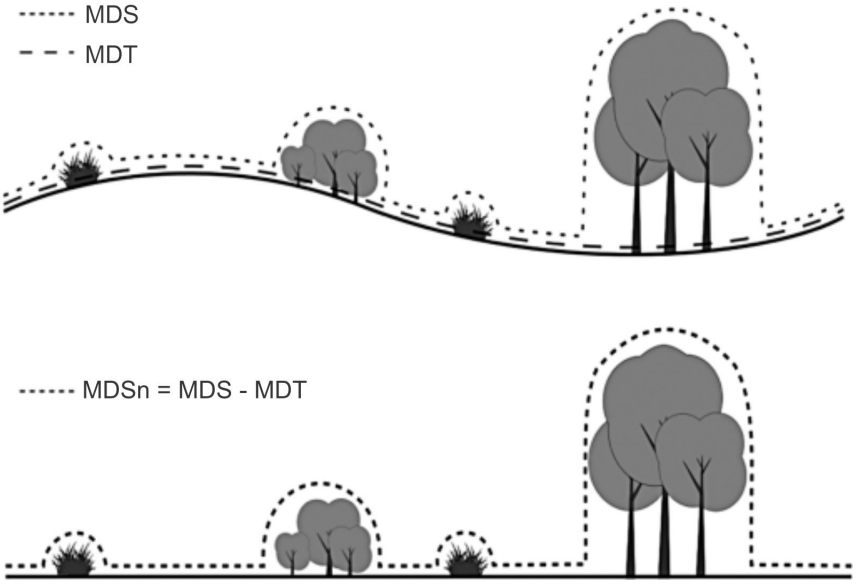
Miller e Laflamme (1958 apud EL-SHEIMY; VALEO; HABIB, 2005)[1] descrevem o Modelo Digital de Terreno (MDT) como uma categoria de representação matemática da superfície terrestre ou uma simples representação estatística de uma superfície contínua do solo, composta por um grande número de pontos com coordenadas conhecidas. O MDT também pode ser definido como um modelo digital simples da superfície terrestre (BRIESE, 2010). De qualquer forma, as aplicações do MDT são diversas, incluindo usos militares, várias engenharias, geociências, entre outros, sendo uma ferramenta importante para modelar e analisar informações espaciais.

A qualidade desses modelos pode ser avaliada de duas maneiras: a primeira envolve analisar a qualidade dos dados que o geraram, enquanto a segunda concentra-se na qualidade do modelo. Em relação à qualidade dos dados, Briese (2010) considera a análise da densidade de pontos, distância entre os pontos e um mapa de acurácia dos dados. Ao abordar a qualidade do modelo, duas perspectivas podem ser consideradas — qualidade externa e interna — que envolvem: a qualidade externa, que compara o modelo estimado com dados de controle externos; e a qualidade interna, que estima a qualidade interna com base na qualidade dos dados de entrada.

Há outros termos similares ao MDT, que Li, Zhu e Gold (2005) define basicamente como sinônimos, com pequenas variações dependendo da aplicação. Para compor o MDT, este deve incluir as formas do terreno, características locais (como estradas) e recursos naturais. Além do MDT, outro termo comum é o MDS (Modelo Digital de Superfície), derivado dos primeiros ecos captados por um sistema LiDAR. Segundo Briese (2010), o MDS descreve a cobertura vegetal e objetos antrópicos. Os primeiros ecos correspondem às superfícies mais altas de uma região. Refinando esses dados, é possível usar os ecos para gerar um MDS (BRIESE, 2010). Com o MDS e o MDT em mãos, é possível gerar um MDSn, comumente utilizado em extrações de feições em dados LiDAR. O MDSn é definido pela subtração do MDS pelo MDT e, segundo Briese (2010), é um modelo importante para separar edificações de outros objetos, como a vegetação. A Figura 3.12 apresenta a representação desses modelos.

[1] MILLER, C. L.; LAFLAMME, R. A. *The Digital Terrain Model: Theory & Application.* [S.l.]: MIT Photogrammetry Laboratory, 1958.

Figura 3.12 – Modelos digitais.



Fonte: adaptado de Mirosław-Świątek et al. (2016).

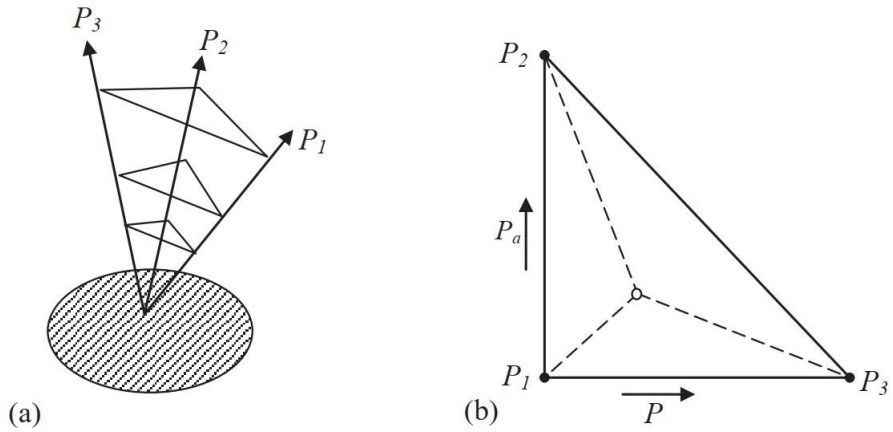
4 MODELOS DE CORES RGB E IHS

Uma imagem representa, segundo Jain (1989, p. 49), “uma distribuição espacial de quantidades físicas como luminância e frequências espaciais de um objeto”. A percepção humana da imagem advém dos atributos relacionados a ela, como brilho, cores e bordas, condicionados, segundo Jain (1989, p. 60), a 12 níveis de cinza e algumas centenas de tons de cores. E as cores em uma imagem são, primordialmente, o maior fator de identificação de objetos pela visão humana (GONZALEZ; WOODS, 2008).

Para melhor simbolizar as composições de cores, a padronização em modelos de cores é usada com representação tridimensional com um ponto com coordenada conhecida, pois é necessária a combinação de três estímulos independentes para percepção visual (MENESES, 2012). Os modelos são importantes porque um único modelo de cores não pode representar todo o universo de cores, fazendo com que cada modelo seja adequado em um objetivo distinto (PEDRINI; SCHWARTZ, 2008). Por exemplo, o modelo mais utilizado em TVs, monitores e câmeras é o RGB, e no processamento de imagens o modelo IHS é comumente empregado. Já o modelo empregado em *plotters* e impressoras é o CMYK (ciano, magenta, amarelo e preto).

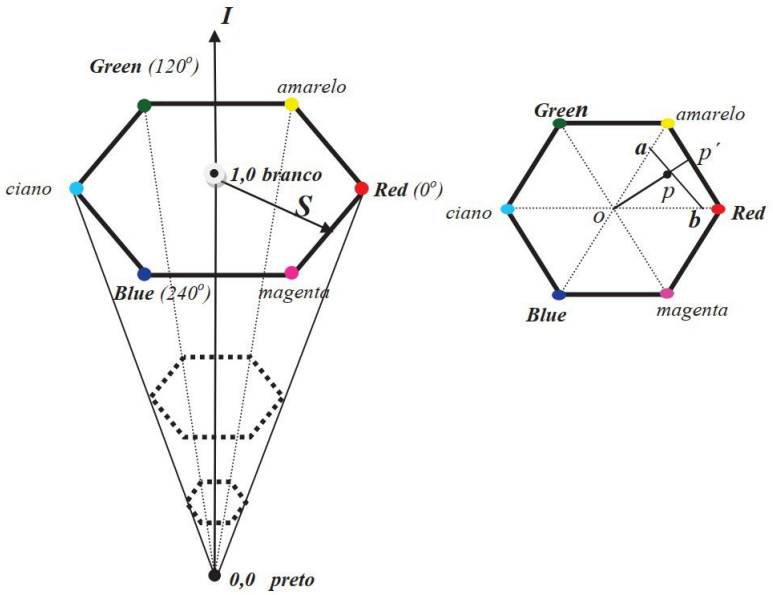
A representação tridimensional das cores pode ser elaborada, segundo Meneses (2012), de três formas: graficamente, numericamente ou em notação vetorial (Figura 4.13). Sendo assim, um triângulo de cor é usado para indicar a proporção resultante da mistura de cores primárias (RGB) compondo o modelo de cores RGB, em que cada cor pode ser representada por suas coordenadas (coordenadas de tricromacidade). Cada eixo do triângulo representa uma cor básica, definida pelo sistema de especificações tristímulo adotado pela *Commission Internationale de l’Eclairage (CIE)* como padrão internacional de colorimetria. Outra representação é o sólido hexacone (Figura 4.14), concebido pela combinação de Intensidade, *Hue* e Saturação, compondo o modelo IHS (MENESES, 2012).

Figura 4.13 – Representação gráfica e vetorial da mistura de cores, em que as quantidades de energia são representadas pelos vetores P_1, P_2 e P_3, em (a) os planos triangulares são definidos pela combinação das energias, e em (b) a projeção do plano triangular define o diagrama de vetores em que pode-se expressar a quantidade de energia por coordenadas.



Fonte: Meneses (2012)

Figura 4.14 – Geometria hexacone para representação do modelo IHS.



Fonte: Meneses (2012)

Segundo Meneses (2012, p. 129), a intensidade, matiz (*hue*) e saturação são "atributos da cor, fortemente percebidos pela visão", sobre a visão humana. Dessa forma, esse modelo não deve ser entendido como um arranjo de cores semelhante ao RGB, mas como uma decomposição numérica do RGB (GONZALEZ; WOODS, 2008). Pode-se descrever as componentes do modelo IHS como: a intensidade, também conhecida como brilho, é entendida como a percepção da cor; quanto maior, mais perceptível é a cor; a *hue*, também chamado matiz, que corresponde à cor pura,

ou seja, é associado ao comprimento de onda predominante; e a saturação é uma componente que indica a quantidade de cor branca no *hue*, ou seja, o grau de mistura da cor (GONZALEZ; WOODS, 2008). Portanto, o modelo IHS permite aplicações baseadas em atributos de cores, e não somente na cor, como ocorre no modelo RGB.

A conversão do modelo de cores RGB para IHS consiste em extrair os componentes independentes que caracterizam a cor. Os componentes de intensidade e de saturação são convertidos do intervalo 0 – 255 para 0 – 1, e o componente matiz pode ser expresso em ângulos no intervalo 0° a 360° (MENESES, 2012). A transformação é apresentada pelas Equação 4.1 a Equação 4.4, assumindo que os valores de RGB e IHS estão no intervalo [0,1] para IHS [0,1] (GONZALEZ; WOODS, 2008).

$$H = \begin{cases} \theta & se\ B \leq G \\ 360 - \theta & se\ B > G \end{cases} \tag{4.1}$$

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\} \tag{4.2}$$

$$I = \frac{1}{3}(R+G+B) \tag{4.3}$$

$$S = 1 - \frac{3}{(R+G+B)}\left[\min(R,G,B)\right] \tag{4.4}$$

Deste modo, consegue-se perceber que as imagens podem ser representadas por diversos modelos de cores, além dos apresentados nesta seção. Dependendo do modelo de cor considerado, diferentes componentes deste modelo podem ser associados aos diferentes canais de entrada de uma rede neural, possibilitando que as redes tenham a informação de cor incorporada em sua arquitetura. Essa informação pode contribuir com o processo de classificação de alguns objetos de interesse. O modelo IHS, segundo Meneses (2012), é composto nessa sequência pela percepção do olho humano, sendo maior para intensidade do que para a matiz e saturação. Tal fato não impede outras combinações para além do agradável ao visível, principalmente ao tratarmos do uso de algoritmos computacionais que não seguem os preceitos da visão humana.

5 EXTRAÇÃO DE EDIFICAÇÕES

A extração de edificações e a sua modelagem tridimensional podem ser realizadas por meio do uso de imagens aéreas por estereoscopia. Segundo Brenner (2010), esse processo pode ser considerado lento e árduo quando comparado com as opções disponíveis atualmente. Com o surgimento dos sistemas LiDAR, a obtenção sistemática de dados 3D tornou-se viável em larga escala, aliada ao desenvolvimento de aplicações para a aquisição de dados terrestres, o que aumentou a quantidade de informações disponíveis em uma cidade, incluindo o mapeamento de fachadas, por exemplo (BRENNER, 2010).

A detecção de edificações a partir de nuvem de pontos obtidos por sistemas de varredura a LASER aerotransportados pode ser elaborada tradicionalmente por meio de filtros. Isso parte do princípio de que as edificações estarão sempre acima do terreno, permitindo sua separação pela diferença de altura entre a superfície do terreno e os objetos. Embora esse processo seja, a princípio, simples, inúmeras dificuldades estão relacionadas ao uso desses filtros, como a capacidade de diferenciar as edificações dos outros objetos urbanos, principalmente quando há sobreposição entre eles. Essas dificuldades podem ser minimizadas pelo uso de sistemas LiDAR com recepção de múltiplos pulsos (BRENNER, 2010).

O uso de algoritmos para torná-la semi-automática possui enfoque em identificar os limites internos e externos das edificações, assim como os segmentos que a compõem. Segundo Awrangjeb, Ravanbakhsh e Fraser (2010), a extração envolve uma primeira etapa de separação das edificações dos pontos de terreno, incluindo árvores e outros objetos. A segunda etapa consiste na produção e regularização dos segmentos das bordas das edificações. Diversos algoritmos podem ser aplicados para essa regularização, tais como o de Douglas–Peucker, o uso de triangulação de *Delaunay*, o emprego do algoritmo α-*shape* (SANTOS; GALO; CARRILHO, 2018) e o método iterativo CD-spline (changeable degree spline) (SANTOS; GALO; HABIB, 2020).

Dentre as aplicações das extrações, Brenner (2010, p. 170) cita o "planejamento da localização de antenas para fornecedores de telefones celulares, sistemas de informação turística, *marketing* urbano, sistemas de navegação intermodal, simulações de microclima e propagação de ruído e jogos de computador". Outra aplicação chave na extração de contornos de edificações é o auxílio na manutenção de sistemas cadastrais e na resolução de questões de ocupação do solo, além de fornecer subsídios para a modelagem 3D de cidades. É importante destacar que o cadastro tende a se beneficiar diretamente de tecnologias que permitam a atualização rápida e constante, e que possibilitem o registro temporal das parcelas e o controle da atividade pública (AMORIM; PELEGRINA; JULIÃO, 2018). Sendo assim, metodologias eficientes de extração de edificações tendem a impactar a sociedade em diferentes campos, principalmente na administração pública.

Mais recentemente, os algoritmos de *deep learning*, que possuem redes neurais convolucionais, demonstraram que podem ser usados para mitigar problemas inerentes à extração, como seu uso em imagens de alta resolução espacial, vias e contornos de edificações (XU et al., 2018; ZHANG; LIU; WANG, 2018; SHI; LI; ZHU, 2019). Esses algoritmos foram replicados para diferentes objetivos, como demonstram os estudos de detecção de derramamento de óleo (YEKEEN; BALOGUN; YUSOF, 2020) e classificação e contagem de gado (XU et al., 2020).

5.1 Métricas para avaliação das extrações

A avaliação das extrações constitui um importante passo para a valoração do produto gerado e permite comparações entre técnicas e metodologias distintas. Para esse processo, é necessário adotar padrões de avaliação da qualidade dos dados, avaliando as métricas comumente usadas em outras pesquisas da mesma temática. Diversas métricas podem ser utilizadas, podendo-se observar que estudos recentes de extração de edificações por *deep learning* demonstram que a avaliação da qualidade pode ser definida principalmente pelos estimadores *precision* (também denominado *correctness*), *recall* (também denominado como *completeness*), *F1-Score*, *overall accuracy (OA)* (ou apenas *accuracy*) e *Matthews Correlation Coefficient (MCC)* (XU et al., 2018; ZHANG; LIU; WANG, 2018; MEYER; LEMARCHAND; SIDIROPOULOS, 2020; YEKEEN; BALOGUN; YUSOF, 2020; WANG; ZHANG; DAI, 2020; DIAKOGIANNIS et al., 2020).

A determinação destas métricas pode ser feita a partir dos valores de verdadeiro positivo (*tp – true positive*), verdadeiro negativo (*tn – true negative*), falso positivo (*fp*) e falso negativo (*fn*). O *tp* é uma medida que corresponde ao número de pixels em que o rótulo atribuído pelo algoritmo foi correto; *tn* é o *true negative*, que conta o número de pixels no qual a rotulação semântica previu corretamente em que não é o rótulo de interesse; *fp* é o *false positive*, que soma todos os pixeis no qual o rótulo semântico determinado pelo algoritmo é incorreto; e *fn* são os *false negative*, medida que conta todos os pixels em que deveriam ser atribuídos rótulos específicos, mas não receberam. Deste modo, com base nestes valores pode-se estimar a métrica *precision*, que permite medir a proporção de rótulos previstos corretamente pela rede, a partir da Equação 5.1. O estimador *recall* mede a proporção de pixeis cujo rótulo semântico foi previsto corretamente ao se comparar com o rótulo de referência (Equação 5.2) e o *F1-Score* resume o desempenho das previsões da rede e, segundo Sokolova, Japkowicz e Szpakowicz (2006, p. 3) “beneficia algoritmos altamente sensíveis e desafia algoritmos com maior especificidade” (Equação 5.3). A *overall accuracy (OA)* determina o quão bem o pixel é classificado (Equação 5.4). O *Matthews Correlation Coefficient (MCC)* (Equação 5.5) permite medir a qualidade de um dado com classes desbalanceadas, ou seja, com diferentes tamanhos entre as classes.

$$precision = \frac{tp}{tp + fp} \tag{5.1}$$

$$recall = \frac{tp}{tp + fn} \tag{5.2}$$

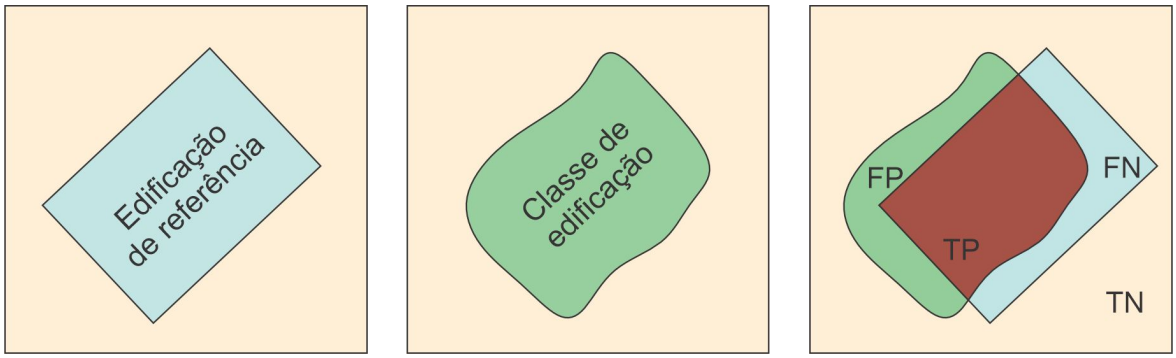
$$F1 - Score = 2.\frac{precision.recall}{precision + recall} \tag{5.3}$$

$$OA = \frac{tp + tn}{tp + fp + tn + fn} \tag{5.4}$$

$$MCC = \frac{tp.tn - fp.fn}{\sqrt{(tp + fp).(tp + fn).(tn + fp).(tn + fn)}} \tag{5.5}$$

Na Figura 5.15 pode-se observar, de modo gráfico, a comparação entre os valores de tp, tn, fp e fn.

Figura 5.15 – Identificação e localização do *tp*, *tn*, *fp* e *fn* ao fazer a comparação entre uma edificação de referência (a), o resultado da edificação gerada por um algoritmo de classificação (b) e na sobreposição da edificação de referência e a gerada por um algoritmo (c).



(a) Edificação de controle, ou referência.

(b) Edificação classificada, ou segmentada.

(c) Sobreposição entre o controle e a classificação.

Fonte: Buján et al. (2012).

6 MATERIAL E METODOLOGIA

Antes do início dos estudos propostos, um experimento preliminar foi realizado com o propósito de aprimorar alguns aspectos teóricos e práticos ligados ao tema tratado, sendo este experimento descrito no Apêndice A. Este experimento foi importante, uma vez que forneceu subsídios para o estabelecimento da metodologia final e a realização dos experimentos finais, dando indicações relativas à capacidade computacional requerida, reconhecimento do *dataset* e consolidação das bases do ambiente de execução do treinamento, validação e análise dos resultados. O experimento propiciou a fundamentação da metodologia, como visto na Figura 6.16.

6.1 Material

Os seguintes materiais foram usados na condução desta pesquisa:

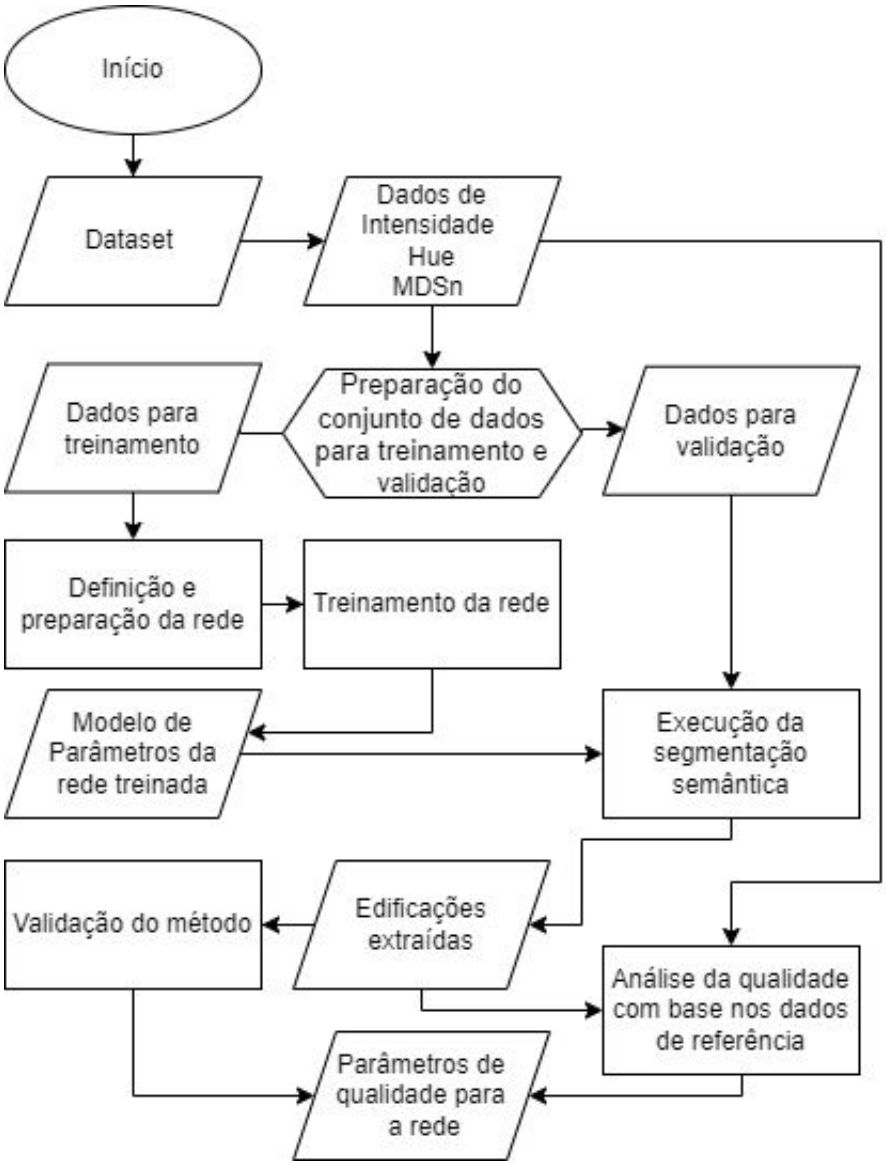
- Conjunto de dados internacional para *benchmark* contendo as *true* ortofotos (GSD de 5cm) e nuvem de pontos LiDAR (distância entre as amostras 5cm) da cidade de *Potsdam*, disponibilizados pela *International Society for Photogrammetry and Remote Sensing* (ISPRS) (ISPRS-WG-III/4, 2018);
- Plataforma *Python* e bibliotecas (incluindo a biblioteca *MXNET*) para suporte e manipulação no ambiente *Jupyter Notebook*;
- Algoritmos relacionados às redes selecionadas disponibilizado pelos autores publicamente no portal *GitHub* (DIAKOGIANNIS et al., 2020; RONNEBERGER; FISCHER; BROX, 2015);
- Computador equipado com uma GPU (*Graphics Processing Unit*) *Nvidia*, com drivers CUDA (*Compute Unified Device Architecture ou Arquitetura*) e cuDNN (*CUDA Deep Neural Network library*) para operações locais;
- Ambiente virtual do *Google Colaboratory* (ou Colab) em sua versão *Pro*.

6.2 Metodologia

Essa pesquisa pode ser classificada, segundo sua finalidade, em uma pesquisa de desenvolvimento experimental, partindo de conhecimentos anteriores com enfoque na melhoria de segmentação semântica de edificações, por *deep learning*, a partir da combinação de dados LiDAR e imagens ópticas. Na Figura 6.16 observa-se o fluxograma de trabalho da metodologia proposta.

Após as segmentações, a extração é feita e analisada estatisticamente pelas métricas de qualidade de dados descritas na Seção 5.1, e comparadas com a execução da metodologia em imagens RGB sem combinação ou decomposição dos canais. Por fim, os resultados serão apresentados e a hipótese inicial colocada será avaliada.

Figura 6.16 – Fluxograma da metodologia proposta.



Na seção seguinte, são apresentados, além da área de estudos, os detalhes da metodologia, sintetizados no fluxograma anterior.

6.2.1 Área de estudos

A área de estudos da pesquisa é concentrada no conjunto de dados de imagens ópticas e nuvem de pontos LiDAR de *Potsdam*, por ser público e de acesso gratuito, permitindo sua avaliação e comparação com resultados obtidos por outros pesquisadores. O *dataset* de *Potsdam* é dividido em 38 partes (Figura 6.17), possuindo *true* ortofoto e MDT com distância entre os

pontos no terreno de 5 cm nos produtos. As *true* ortofotos possuem resolução radiométrica de 8 bits, com três diferentes composições dos canais: IRRG, RGB e RGBIR; o MDT está codificado em 32 bits em níveis de cinza no formato PNG (Figura 6.18).

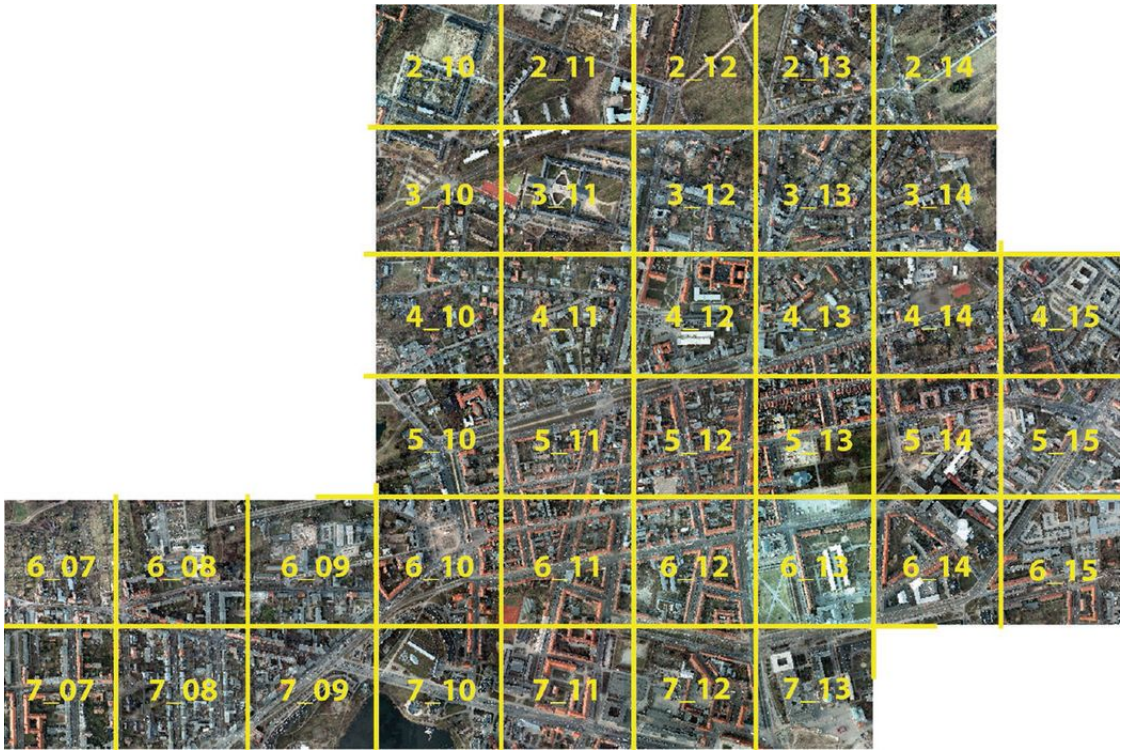


Figura 6.17 – Mosaico do *dataset* de *Potsdam*.

Fonte: ISPRS-WG-III/4 (2018)

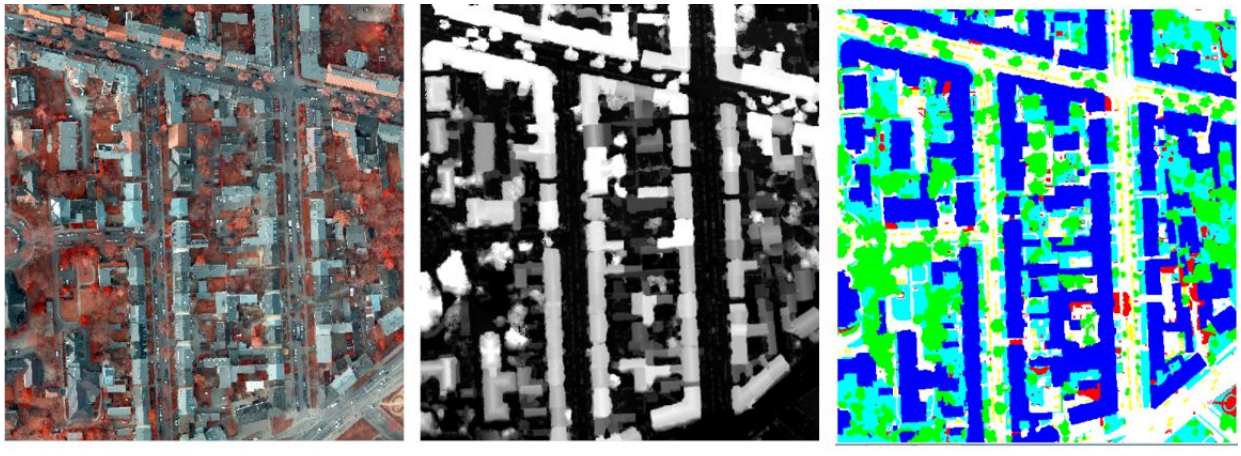


Figura 6.18 – Exemplo da composição presente no *dataset* de *Potsdam*. (a) é a true ortofoto, (b) é o MDSn, e (c) são os rótulos.

Fonte: ISPRS-WG-III/4 (2018)

6.2.2 Preparação dos dados

Os dados foram inicialmente preparados para possuírem a mesma composição: imagens ópticas e MDSn. Na Subseção 6.2.2.1, será descrita a preparação dos dados da composição proposta pela hipótese, ou seja, o uso da matiz, intensidade e *MDS* normalizado (HInDSM). Na Subseção 6.2.2.2 é feita a descrição de como os dados RGB foram preparados para o treinamento.

É importante destacar não haver dados LiDAR para todas as áreas que possuem *true* ortofoto, desse modo, apenas as regiões com ambos dados foram usadas na pesquisa, totalizando 24 pares (RGB e MDSn). Dentre essas, há uma imagem com dimensão 5999x5999 pixeis que impossibilita combinações, tendo sido redimensionada para 6000x6000 pixeis utilizando a técnica de interpolação por vizinho mais próximo.

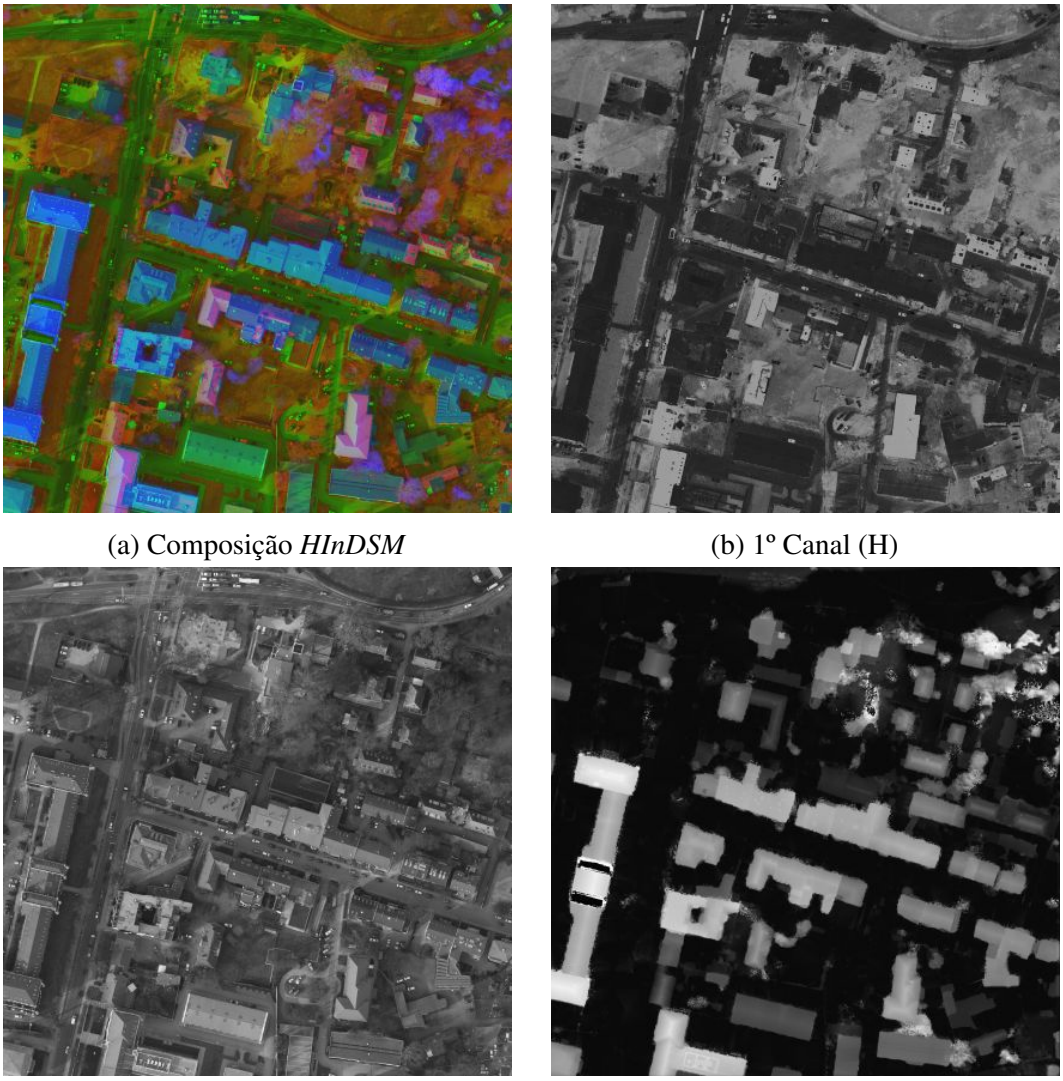
6.2.2.1 *Dataset HInDSM*

No caso do MDSn, o *dataset de Potsdam* já os fornece em dois formatos, ambos normalizados. O primeiro pelo *software LASTools* e o segundo feito pela *ISPRS*, ambos em formato *PNG* representado em níveis de cinza com 8 bits sem sinal. Além disso, o *dataset* possui os arquivos brutos da nuvem de pontos, que não foram usadas nessa pesquisa.

Para os testes da hipótese, as imagens RGB foram transformadas para o modelo de cores IHS, realizada por algoritmos em Python, com base nas equações apresentadas. Sendo o objetivo das transformações a criação da composição do *dataset HInDSM*, o algoritmo foi projetado para, ao final dos cálculos, combinar o MDSn com a resultante dos cálculos de transformação, compondo uma imagem 8 bits sem sinal com os três canais. O primeiro composto pelo *hue* (ou matiz simbolizado pela letra H), o segundo pela intensidade (I) e o terceiro pelo MDSn. A saturação foi excluída do modelo, substituída pelo MDSn. Ao final, 24 imagens foram criadas com estas características, ou seja, contendo as bandas H, I, e nDSM, que combinadas resultam em *HInDSM*.

Sobre as bibliotecas usadas, tem-se que: a biblioteca *tifffile* foi usada para salvar as imagens em formato ".TIF"mantendo as características de entrada; para união das camadas, foi usada a biblioteca *OpenCV*; a biblioteca *numpy* foi usada para trabalhar com as imagens em formato matricial; a biblioteca *math* ofereceu suporte às operações matemáticas; a biblioteca *tkinter* permite a abertura das imagens diretamente na execução do código, facilitando a criação da lista de imagens (tanto das ortofotos, como dos MDSn); a biblioteca *os* foi usada para suporte nas operações que executam comandos no sistema operacional. No Apêndice B.1 é apresentado parte do código em linguagem *Python* utilizado para realizar as transformações envolvidas na preparação do *dataset HInDSM*.

A Figura 6.19 apresenta um exemplo do resultado das transformações que permitiram gerar a combinação *HInDSM*.



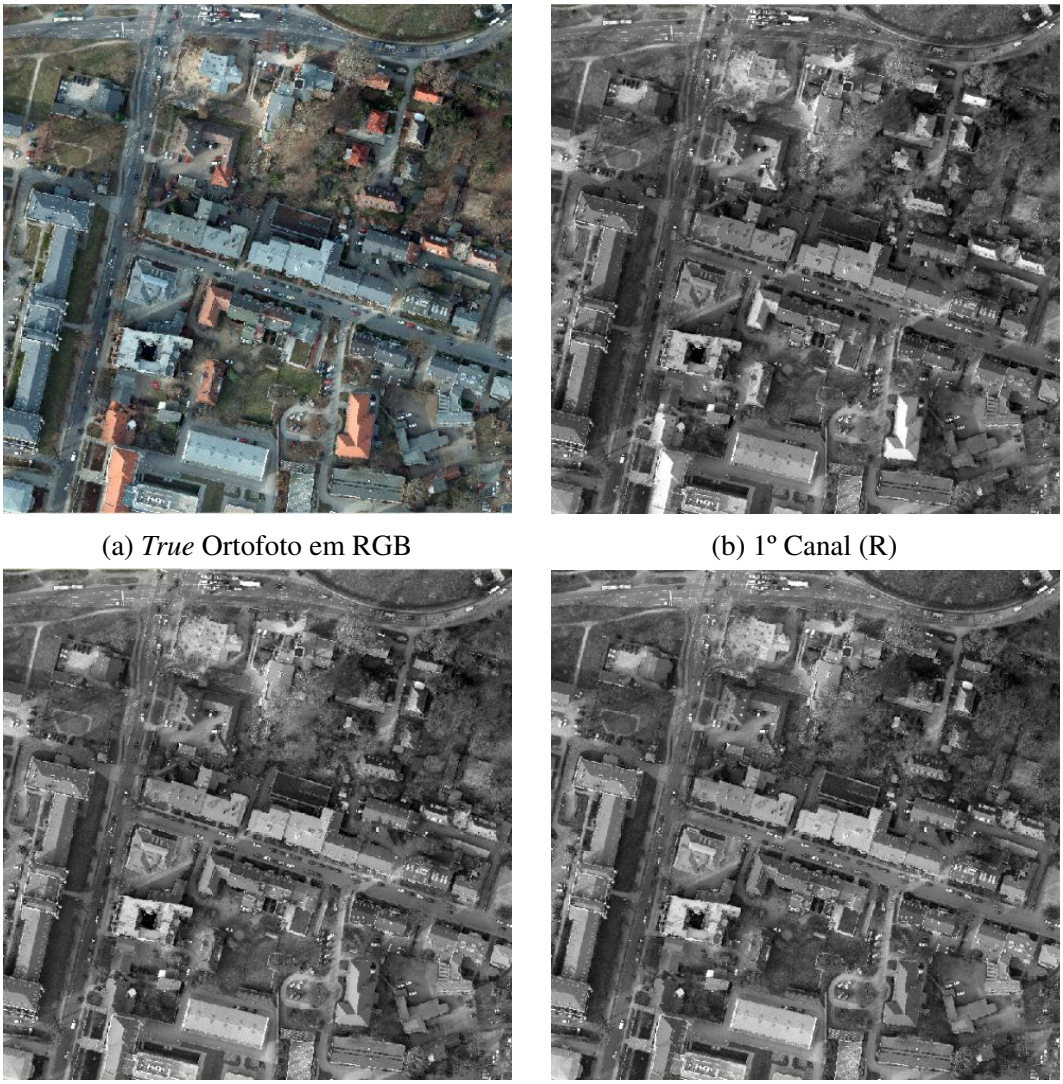
(a) Composição *HInDSM* (b) 1º Canal (H)

(c) 2º Canal (I) (d) 3º Canal (*nDSM*)

Figura 6.19 – Composição *HInDSM* - top_potsdam_3_12.

6.2.2.2 *Dataset RGB*

Para o *dataset RGB* não houve modificações nos arquivos originais do *dataset Potsdam*. Na Figura 6.20 é apresentado um exemplo do *dataset RGB* e respectivos canais.



(a) *True* Ortofoto em RGB (b) 1º Canal (R)

(c) 2º Canal (G) (d) 3º Canal (B)

Figura 6.20 – Combinação RGB - top_potsdam_3_12.

6.2.2.3 Rótulos

Os rótulos fornecidos pelo *dataset* de *Potsdam* possuem 6 classes: edificações, superfície impermeável, vegetação baixa, árvores, fundos e carros. Para esta pesquisa, as classes superfície impermeável, vegetação baixa, árvores, fundos (*background*) e carros foram unidas em uma única categoria, agora chamada de *background*. Desse modo, a classe edificações fica isolada, e tem-se um rótulo binário entre edificações e *background*. Os rótulos foram iguais em ambos os *datasets* (RGB e HInDSM).

6.2.2.4 Recorte dos *datasets* - criação dos *tiles*

Ambos *datasets* foram recortados utilizando a ferramenta fornecida por Diakogiannis et al. (2020), em *Python*, que permite a configuração completa dos parâmetros do recorte. As bibliotecas envolvidas no recorte são: *rasterio, numpy, glob e cv2*.

Os parâmetros utilizados neste processo e o espaço alocado estão na Tabela 6.2.

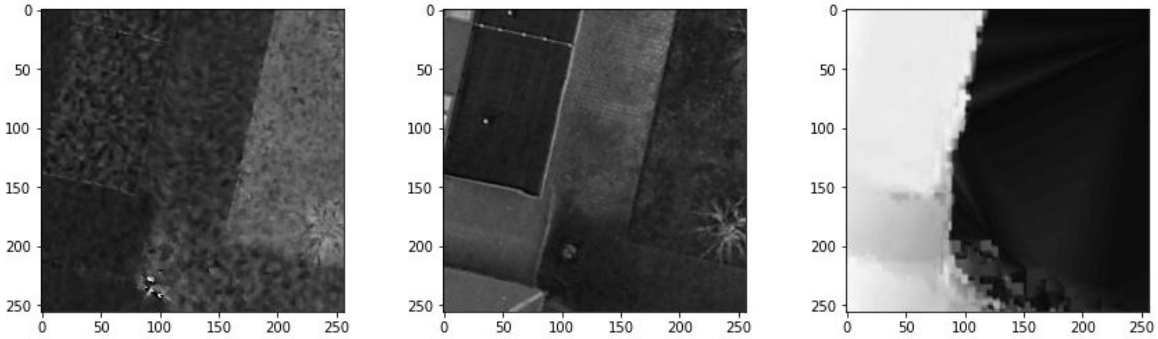
Tabela 6.2 – Parâmetros de recorte.

Dataset	**Tamanho do *tile***	**Nº Classes**	**Reservado para validação**	**Espaço alocado**
HInDSM	256x256	2	10%	∼80GB
RGB	256x256	2	10%	∼80GB

No processo de recorte, são geradas mais duas informações para os dados incorporarem maior contexto ao treinamento. A primeira é um mapa de bordas entre os objetos, e a segunda é um mapa de distâncias das bordas aos centros. Ao final do processo, tem-se um *tile* com 3 canais de imagem e um *tile* correspondente de máscaras com 6 canais. O *tile* de imagem armazena, nos seus canais, os canais correspondentes do *dataset* recortado (R-G-B, ou H-I-nDSM). No *tile* de máscaras, os dois primeiros canais são reservados aos dois rótulos (edificações e *background*), os dois próximos são reservados ao mapa de bordas das duas classes, e os dois últimos ao mapa de distâncias das duas classes.

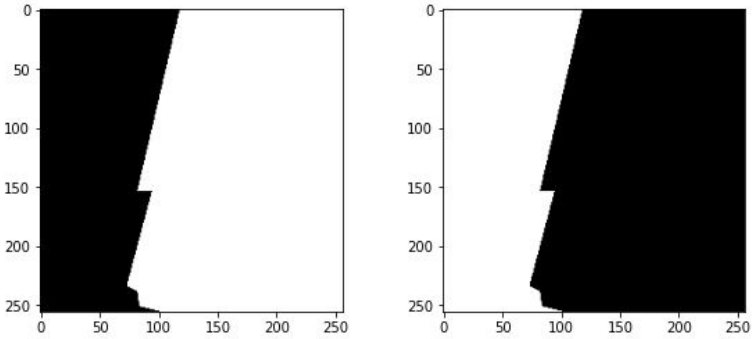
As bordas são geradas pela função $MORPH\_CROSS$ presente na biblioteca OpenCV com uma dilatação de 1 pixel para aumentar a área do objeto. O mapa de distâncias é gerado pela mesma biblioteca, pela função $distanceTransform$, que corresponde à aplicação da Transformada de Distância, em que o valor de cada pixel é substituído por sua distância até o pixel de fundo mais próximo.

Como exemplo da composição dos *tiles* do *dataset HInDSM*, a Figura 6.21 mostra os canais *hue* (Figura 6.21a), intensidade (Figura 6.21b) e *nDSM* (Figura 6.21c) de um *tile* utilizado na rede. O *tile* de máscara da mesma área é composto pelos rótulos de referência de *background* (Figura 6.22a) e edificação (Figura 6.22b), das bordas (Figura 6.23) e do mapa de distâncias (Figura 6.24). O *dataset* RGB apenas possui modificação nos canais dos *tiles* de imagem, compostos pelos canais R, G e B.



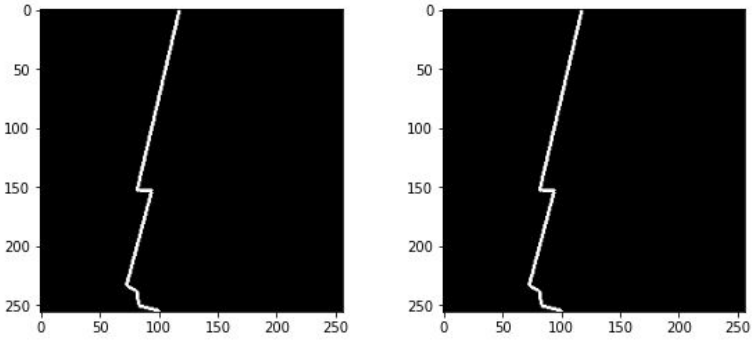
(a) Canal H (b) Canal I (c) Canal nDSM

Figura 6.21 – *Tiles* de imagem.



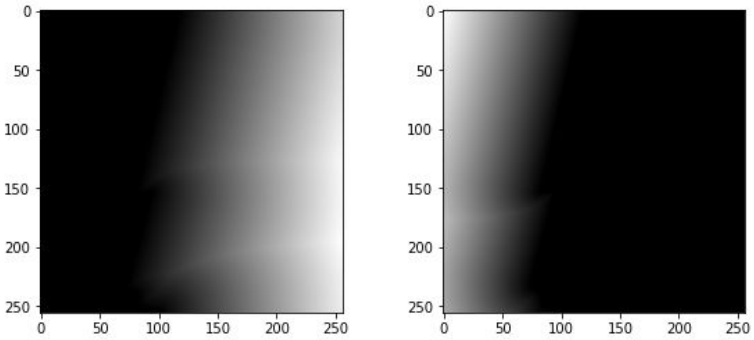
(a) Rótulo de *background* (b) Rótulo de *edificação*

Figura 6.22 – *Tiles* de rótulo de referência.



(a) Borda da edificação (b) Borda do *background*

Figura 6.23 – *Tiles* de bordas.



(a) Mapa de distância do *background* (b) Mapa de distância da *edificação*

Figura 6.24 – *Tiles* de mapa de distâncias.

6.2.3 Treinamento

Nesta seção será discutido o treinamento do modelo de parâmetros na arquitetura de *deep learning* no *dataset* utilizado nesta pesquisa.

6.2.3.1 Ambiente de treinamento

O ambiente de treinamento utilizado nesta etapa da pesquisa foi o *Google Colaboratory*, mais conhecido como *Colab*. O *Colab* é um ambiente em nuvem, hospedado pela empresa *Google*, visando incentivar pesquisas em *deep learning* e aprendizado de máquina. A máquina é virtual, e o acesso é feito por uma interface baseada no *Jupyter Notebook*, sendo possível a execução dos códigos em etapas. A linguagem principal é *Python*, e a maquina já vem com diversas bibliotecas pré-instaladas para suporte nas operações de aprendizado profundo.

Uma vantagem do uso do *Colab* é a possibilidade acessar *GPUs* rápidas, com mais memória *VRAM* do que as de uso doméstico. Os *drivers* essenciais para esta pesquisa já estavam disponíveis no ambiente, como o *driver CUDA* e *CUDNN*. A versão do *Colab* utilizada nesta pesquisa foi a *Colab Pro* e, apesar de ser um serviço pago, o *Google* não garante o *hardware* que será disponibilizado em todos os acessos. Nesta pesquisa, manteve-se a melhor placa de vídeo disponível pelo serviço no momento do uso da plataforma, a placa *NVIDIA P100*.

Para a execução dos algoritmos foram efetuadas as seguintes modificações e instalações:

- Substituição da versão da biblioteca *OpenCV* para a versão 4.1.0.25, devido às mudanças ocorridas nas novas versões que incompatibilizariam os algoritmos;
- Instalação da biblioteca de suporte às operações de *deep learning MXNet*, em sua versão compatível com o driver *CUDA* 11.0 (mxnet-cu110);
- Conexão com o *Google Drive*, para *download* dos *datasets* e armazenamento dos modelos de parâmetros e;
- Com a conexão do *Google Drive* criada, acessar os algoritmos da arquitetura diretamente salvos no repositório do *Google*. Desse modo, uma pasta foi mantida com os algoritmos que seriam usados nos treinamentos.

Apesar de altamente configurável, o *Google Colab* não permite que a execução exceda um dia completo. Ou seja, a cada 24 horas a maáquina virtual é reiniciada, e os processos perdidos. Para mitigar esse problema, o treinamento foi dividido em partes, de modo que antes de completar 24 horas o modelo de parâmetros era salvo. Assim, após a reinicialização automática, o processo continuava recuperando e aplicando o modelo de parâmetros que havia sido gerado antes de reiniciar. Deste modo, não há prejuízos no aprendizado por ter a execução particionada, apenas no critério de parada do treinamento que será descrito na Subseção 6.2.3.2.3.

6.2.3.2 Parâmetros de treinamento

Os parâmetros de treinamento foram mantidos para corresponderem aos executados pelos estudos de Diakogiannis et al. (2020), como a função custo, o *batch normalization*, as

transformações executadas no *dataset* e a agregação manual de gradiente. Apenas o critério de parada difere, devido ao ambiente de execução da arquitetura, mas sem prejuízos ao aprendizado.

A função custo, ou seja, a função cujo cálculo permite ajustar os pesos da rede baseando-se no custo do aprendizado, foi definida como a *Tanimoto* com complementação, exemplificada na Subseção 2.3.2.

O *batch normalization*, ou normalização do *batch*, refere-se a uma técnica proposta por Ioffe e Szegedy (2015) para reduzir o tempo de operação das arquiteturas de *deep learning* por tornar as redes mais estáveis (acurácia estável) e reduzir consequentemente o tempo de convergência do aprendizado. Os parâmetros para normalização (média e desvio-padrão) são aplicados nas imagens durante a execução da fase *encoder*, e na reconstrução os mesmos parâmetros são usados no processo inverso.

Para a definição dos valores de normalização, foram feitas algumas transformações, como já citado, sendo apresentado no Apêndice B.2 os respectivos códigos. Assim, foi possível preparar todas as 24 imagens que serão usadas no treinamento, separadamente por cada canal. Desse modo, ao final, consegue-se determinar a média e o desvio-padrão do *dataset* completo. As bibliotecas usadas no processo foram *OpenCV*, *tkinter*, *os*, *numpy* e *tifffile*. Os parâmetros citados são apresentados na Tabela 6.3 para cada *dataset*.

Tabela 6.3 – Parâmetros de normalização.

Dataset	**Parâmetro**	**Hue**	**Intensidade**	**nDSM**
HInDSM	Média	86.0153	87.2009	46.2624
	Desvio-padrão	38.8891	34.8285	55.0568
Dataset	**Parâmetro**	**Red**	**Green**	**Blue**
RGB	Média	85.8459	91.7766	84.9924
	Desvio-padrão	35.7828	35.1317	36.5129

6.2.3.2.1 Transformações

Durante o carregamento dos dados na VRAM são feitas transformações para incrementar o contexto das imagens na arquitetura. Os estudos de Diakogiannis et al. (2020) incluíam uma transformação de RGB para HSV, e essa transformação era armazenada com as máscaras, somando-se mais três informações (totalizando 9 canais, 6 observados na operação de recorte, e 3 na transformação). Como o dado inserido neste estudo é composto das componentes H-I-nDSM, esse componente o transformaria erroneamente em HSV. Desse modo, essa transformação foi substituída por uma cópia da camada H-I-nDSM (no caso do *dataset HInDSM*) e uma cópia do RGB para manter a coerência de transformações (no caso do *dataset RGB*), conservando o mesmo número de canais nas máscaras com a transformação e igualando o contexto previsto na arquitetura.

Mais três transformações foram feitas no dado: rotação, reflexão e modificações na escala. Essas transformações possuíram um intervalo definido inicialmente, mas a execução é feita aleatoriamente no intervalo, em correspondência ao trabalho de Diakogiannis et al. (2020): o intervalo de rotação entre -85°e +85°; centro dos *tiles* com intervalo de 0 a 256; e o fator de multiplicação da escala entre 0,25 e 1,25. A biblioteca usada para as transformações foi a *OpenCV*, sendo utilizadas as funções $cv2.getRotationMatrix2D$ e $cv2.warpAffine$. Todo o processo foi executado conforme os estudos de Diakogiannis et al. (2020) para as comparações futuras.

6.2.3.2.2 Agregação manual de gradiente

Um dos grandes desafios na execução das arquiteturas de *deep learning* é o *hardware* suportar as operações, pois isso afeta diretamente a eficiência dos cálculos. No caso do carregamento dos dados, a memória *VRAM* (memória de vídeo) é um componente importante a considerar, devido à sua maior eficiência no tempo de processamento em relação à memória *RAM*.

O tamanho do *batch* – quantidade de dados carregada simultaneamente na memória em cada iteração - influência no treinamento devido às constantes atualizações dos pesos. Por exemplo, para um *batch* de 6 *tiles*, após o processamento de 6 *tiles*, os pesos são atualizados, e para um *batch* de 264 *tiles*, os pesos são atualizados considerando os 264 *tiles* processados. De forma geral, um *batch* grande causa baixas oscilações no aprendizado e um *batch* pequeno, por sua vez, causa grandes oscilações. A razão para isso é o maior contexto durante uma iteração.

Especificamente, neste trabalho, o carregamento de 6 *tiles* ocupava toda a memória *VRAM* disponível, ou seja, não seria possível carregar mais tiles simultaneamente. Para resolver a questão, foi utilizada a biblioteca *GLUON*, embarcada na *MXNET*, que traz a estratégia de agregação manual de gradientes, que consiste em modificar a forma de armazenamento dos gradientes e o momento em que os pesos são calculados, possibilitando definir um tamanho de *batch* maior que o fisicamente possível em memória.

Essa estratégia foi aplicada utilizando um tamanho de *batch* de 264 com carregamento simultâneo de 6 *tiles*, cujo funcionamento é explicado a seguir. Ao término da leitura de 6 *tiles*, os gradientes são agregados em uma lista, em lugar de realizar imediatamente a atualização dos pesos, sendo essa atualização adiada até que um número *n* de leituras seja alcançado. O parâmetro *n* representa, portanto, a quantidade de iterações necessárias para totalizar o tamanho do *batch* – *n* = 264/6=44 – para, em seguida, realizar as atualizações dos pesos. Dessa forma, somente após essas 44 iterações, os pesos eram atualizados. Nesse contexto, para a rede neural, ocorre uma espécie de carregamento simultâneo dos 264 *tiles*, embora todo o processo seja executado em fases distintas, com a agregação manual de gradientes em cada fase.

Vale destacar que, no exemplo descrito, não há ganho no tempo de execução, uma vez

que o processo é dividido em partes, mantendo-se a necessidade de execução sequencial das 44 iterações para efetivar as atualizações nos pesos. Entretanto, o mesmo princípio pode ser usado para o paralelismo de *GPUs*, fazendo com que um *cluster* com várias *GPUs* execute os cálculos separadamente e alimentando uma lista de gradientes. Dessa maneira, a velocidade de execução pode ser incrementada pela adição de mais *GPUs*.

Na implementação, é feita uma modificação no modo como o gradiente é armazenado, passando do padrão *'write'* (ou seja, escrevendo e sobrepondo os gradientes anteriores) para *'add'* (adicionando/agregando os gradientes em uma lista), como pode-se ver no Apêndice B, no Código B.3. Ao final das iterações, os pesos são calculados e é necessário reiniciar a lista dos gradientes anteriores, como pode-se ver no Apêndice B.4. Desse modo, os valores são zerados para o processamento do próximo *batch*. No Código B.5 é apresentado o código de execução do que foi descrito nessa seção.

6.2.3.2.3 Critério de parada

O critério de parada corresponde a uma condição, ou condições, mínima(s) que determina o momento de finalização do treinamento. Um deles corresponde ao conhecido por *Early Stopping*, que realiza a parada ao primeiro sinal de estabilização do aprendizado. O objetivo da parada é impedir o *overfitting* ao se realizarem diversas épocas, ou o *underfitting* quando poucas épocas são vistas, ou seja, a parada deve ser no pico do aprendizado.

Devido ao ambiente do *Colab* restringir por tempo o uso da máquina, o critério de parada teve de ser avaliado manualmente a partir da média móvel da função custo estimada para cinco épocas e sob a acurácia no *dataset* de treinamento, sendo avaliada a estabilidade dos índices. Para cada época, um modelo de parâmetros foi armazenado, o que permitiu a escolha da época com melhor acurácia e baixo valor da função custo.

6.2.3.3 Treinamento no *dataset HInDSM* e *RGB*

Em resumo, o treinamento com o *dataset HInDSM* foi iniciado com as seguintes características:

- Tamanho do *batch*: 264 pela agregação manual de gradientes;
- Carregamento aleatório dos *tiles*;
- 10% dos *tiles* são destinados à validação e não são vistos nessa etapa;
- Total de *tiles* disponíveis para treinamento: 41.760 de imagens e 41.760 de máscaras;
- Dimensão dos *tiles*: 256x256 pixeis;
- Aplicado os parâmetros de normalização do *batch* específicos para o *dataset HInDSM*;

- Normalização do *batch* e transformações aplicadas ao carregar os dados na memória;
- Treinamento pausado a cada 16 épocas para reinicialização do ambiente de execução (*Colab*).

6.2.4 Validação dos modelos

O conjunto de validação, que corresponde a 10% dos dados de cada *dataset* (6.816 *tiles* e 6.816 máscaras), foi usado para validar o modelo. Os resultados foram avaliados no *dataset* em termos qualitativos e quantitativos, com o cálculo das métricas para cada *tile* e seu rótulo correspondente.

A análise qualitativa foi feita para dois dados diferentes: sobre *tiles* não vistos pela rede, e sobre três áreas (três ortofotos) que continham dados vistos e não vistos. As métricas de validação quantitativa são *precision*, *recall*, *F1-Score*, *Overall accuracy (OA)* e *Mathews Correlation Coefficient (MCC)*, como apresentado na Seção 5.1.

7 RESULTADOS

Neste capitulo serão apresentados e discutidos os resultados do treinamento e validação dos experimentos deste trabalho.

7.1 Treinamento do modelo de parâmetros

Tratando do critério de parada, tem-se que o modelo de parâmetros do *dataset HInDSM* foi treinado até atingir o maior valor de acurácia no *dataset* de treinamento. O modelo feito pelo *dataset RGB* foi treinado até atingir a mesma quantidade de épocas do obtido com o *dataset HInDSM*. A maior acurácia no conjunto de treinamento foi obtida na época 205, de 97,95%, e mais duas épocas foram treinadas para garantir que realmente o modelo havia estabilizado, totalizando 207 épocas. A maior acurácia do modelo obtido pelo *dataset RGB* foi de 96,60% na época 201, e apesar disso, o treinamento foi conduzido até a época 206. Na Tabela 7.4 é apresentado o resumo das métricas para o critério de parada.

Para o critério de parada também foi avaliada a estabilização da acurácia durante o treinamento. Na Figura 7.25 é apresentada a evolução da acurácia no treinamento em função das épocas a partir da época 144, sendo possível identificar uma tendência de estabilização do modelo *HInDSM*, com baixa oscilação ao se comparar com a evolução do modelo *RGB*.

Tabela 7.4 – Maiores acurácias do treinamento para cada *dataset*.

Dataset	**Época**	**Função custo**	**Acurácia do treinamento**
HInDSM	205	0,16785	97,95%
RGB	201	0,18648	96,60%

No *dataset HInDSM*, o menor valor da função custo (Figura 7.26) foi observado na época 170, sendo ele 0,16437, ou seja, 0,00348 de diferença entre o *HInDSM* e *RGB*. No *dataset RGB*, o menor valor da função custo encontrado também não correspondeu com o maior valor de acurácia. Na época 196, o modelo teve a função custo de 0,17766, sendo 0,00882 menor que a encontrada na época 201. Isto decorre da função custo considerar quatro variáveis para o cálculo: custo de segmentar, custo de identificar as bordas, custo de identificar o mapa de distâncias e custo das "cores"(correspondência entre as cores de referência e as preditas). Sendo assim, a acurácia foi considerada por apresentar uma medida única baseada na segmentação.

A função custo foi considerada para reconhecer o platô de aprendizado atingido pelos modelos. No caso do *HInDSM*, a função custo atingiu o equilíbrio próximo da época 105, além de se apresentar mais estável e com rápida convergência ao platô de aprendizado, se comparado

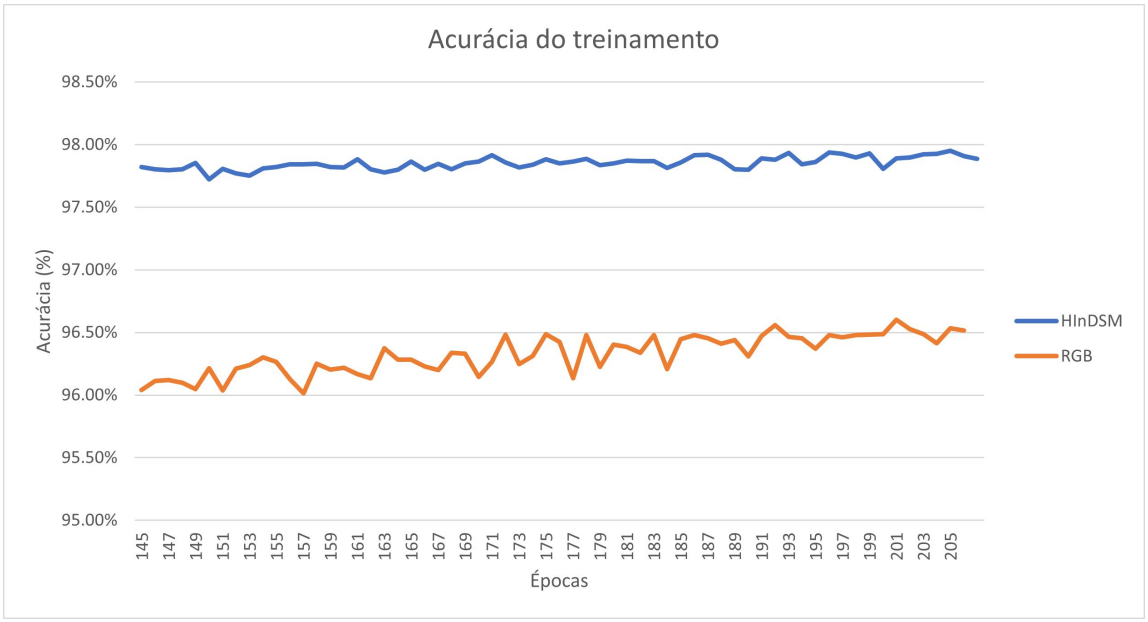


Figura 7.25 – Acurácia do treinamento em cada modelo.

ao modelo *RGB*. Tal ocorrência traz o indício da eficiência no treinamento com a presença do nDSM.

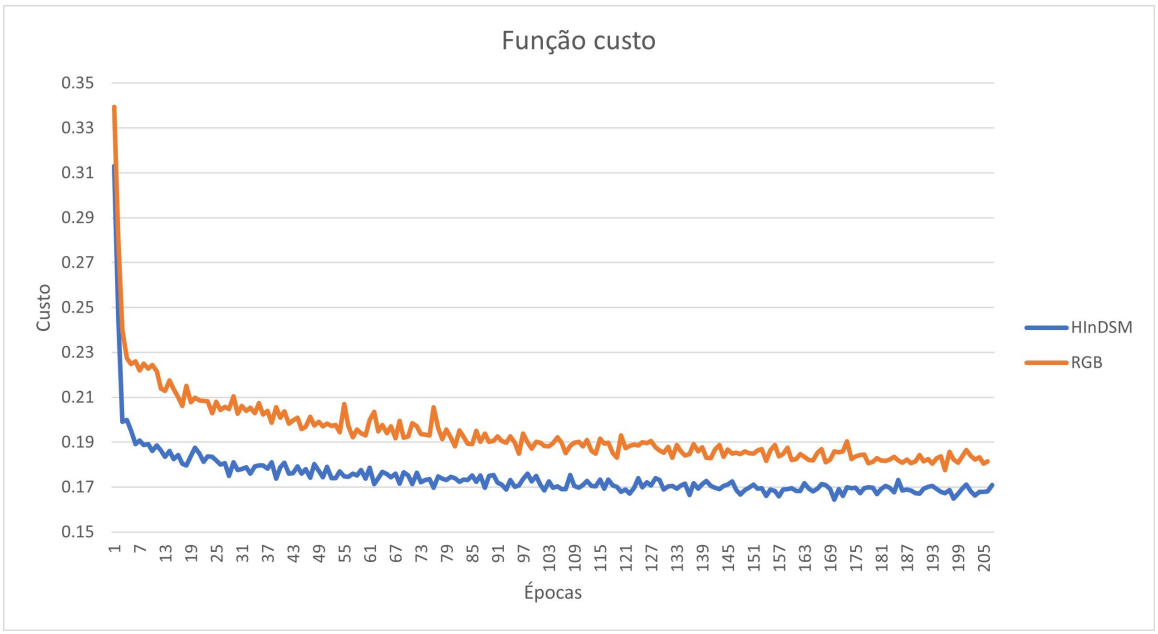


Figura 7.26 – Função custo em cada modelo.

Para definir a parada, a média móvel da função custo foi avaliada com um intervalo fixado em 5 épocas, ou seja, a média é calculada considerando os 5 valores anteriores. Deste modo, detectam-se tendências de redução, estabilização ou aumento da função custo. A Figura 7.27 mostra uma estabilização a partir da época 145, com a linha de previsão sem tendências de aprendizado, indicando a parada. Optou-se por continuar o aprendizado até o momento em que o valor da função custo atingisse seu menor índice, e apresentasse tendência de crescimento,

evitando-se o *overfitting* e o *underfitting*.

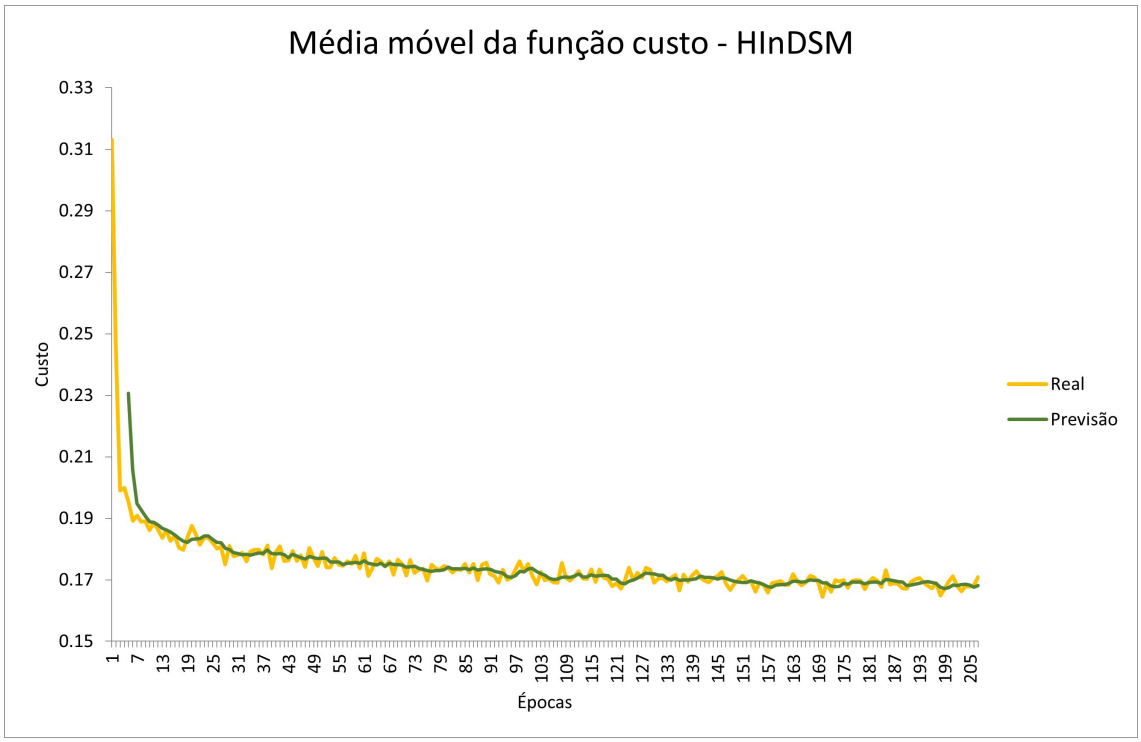


Figura 7.27 – Média móvel da função custo.

O tempo de treinamento foi um fator monitorado a cada época. Em média, o *dataset HInDSM* apresentou tempo de execução médio de cada época de 01h24m29s, e o *dataset RGB* teve tempo médio de 01h24m33s. Apesar da diferença apresentada, tal fato decorre do ambiente de execução dos processos, em que uma máquina pode estar executando outros processos simultaneamente no ambiente virtual e do *hardware* disponível. A ocorrência também foi vista quando um novo ambiente virtual era definido, e o tempo de execução reduzia ou aumentava em 100 segundos. Deste modo, o tempo por época não foi influenciado pela metodologia, já que o mesmo número de *pixels* é analisado em ambos *datasets*. No geral, a execução do treinamento foi de 03h29 em cada época para ambos (*HInDSM* e *RGB*), ou seja, aproximadamente doze dias considerando todas as épocas.

7.2 Validação do modelo

Nesta seção, o modelo de parâmetro escolhido em cada treinamento será avaliado quantitativamente (Subseção 7.2.1) e qualitativamente (Subseção 7.2.2). Devido à presença de modelo de parâmetros salvos durante as épocas, foi possível identificar as métricas continuamente ao longo do treinamento, e serão usadas nas avaliações.

7.2.1 Avaliação Quantitativa

O modelo de parâmetros treinado foi validado em 10% do *dataset*, ou seja, em dados não vistos pela rede ao ser treinado. Para a avaliação, foram construídas matrizes de confusão para estimar o desempenho do modelo analisando pixel-a-pixel.

A partir dos modelos de parâmetros definidos no treinamento dos *datasets*, ambos foram avaliados sob as métricas *precision*, *recall*, *accuracy*, *F1-Score* e *mathews correlation coefficient* (MCC). A matriz de confusão do *HInDSM* está na Tabela 7.5 e as métricas na Tabela 7.6. Para o modelo *RGB*, a matriz de confusão está na Tabela 7.7 e as métricas na Tabela 7.8.

Tabela 7.5 – Matriz de confusão do modelo na época 205 - *HInDSM*.

Modelo na época 205 - *HInDSM*				
		Predito		
		Background	**Edificações**	**Total**
Referência	***Background***	323.904,626	4.502,493	328.407,119
	Edificações	3.887,975	114.398,282	118.286,257
	Total	327.792,601	118.900,775	446.693,376

Tabela 7.6 – Métricas do modelo na época 205 - *HInDSM*.

%	***Background***	**Edificações**
Precision	98,551	96,555
Recall	98,767	95,967
Accuracy	98,025	98,025
F1-Score	98,659	96,260
MCC	94,920	94,920

Tabela 7.7 – Matriz de confusão do modelo na época 201 - RGB.

Modelo na época 201 - RGB				
		Predito		
		Background	**Edificações**	**Total**
Referência	***Background***	323.038,521	5.368,598	328.407,119
	Edificações	7.568,626	110.717,631	118.286,257
	Total	330.607,147	116.086,229	446.693,376

Tabela 7.8 – Métricas do modelo na época 201 - RGB.

%	***Background***	**Edificações**
Precision	97,711	95,375
Recall	98,365	93,601
Accuracy	97,104	97,104
F1-Score	98,037	94,480
MCC	92,525	92,525

As métricas demonstram maior correspondência no modelo *HInDSM* em todos os segmentos avaliados, incluindo na identificação do *background*. A diferença entre as métricas é vista na Tabela 7.9.

Tabela 7.9 – Diferença percentual entre as métricas estimadas para o modelo *HInDSM* (época 205) e *RGB* (época 201).

Diferença (%)	**Background**	**Edificações**
Precision	+0,84	+1,18
Recall	+0,40	+2,37
Accuracy	+0,92	+0,92
F1-Score	+0,62	+1,78
MCC	+2,40	+2,40

Como o objeto de estudos desta pesquisa são as edificações, pode-se inferir que as edificações tiveram os melhores resultados no modelo *HInDSM*, com crescimento de 1,78% no *F1-Score* ao se comparar com as obtidas no modelo *RGB*.

A cada época treinada, um modelo de parâmetros foi gerado e armazenado. Ao final do treinamento, todos os modelos foram recuperados e avaliados no *dataset* de validação, deste modo verificaram-se os melhores valores que poderiam ser obtidos nos resultados, e a progressão que o aprendizado teve no decorrer das épocas. O melhor modelo de parâmetros do *HInDSM*, ao avaliar todos os modelos gerados, foi o da época 203. O melhor modelo de parâmetros do *RGB* foi o *200*. A matriz de confusão foi construída para ambos, e estão na Tabela 7.10 e Tabela 7.11. As métricas encontradas nesses modelos estão na Tabela 7.12 e Tabela 7.13. Com isto, o maior valor de *F1-score* encontrado foi de 96,601% no modelo *HInDSM*, contra 94,708% no *RGB*. A diferença entre os modelos está na Tabela 7.14, próximas às vistas na Tabela 7.9, mantendo a vantagem do modelo *HInDSM*.

Tabela 7.10 – Matriz de confusão do modelo na época 203 - *HInDSM*.

Modelo na época 203 - *HInDSM*				
		Predito		
		Background	**Edificações**	**Total**
Referência	***Background***	324.019,716	4.387,403	328.407,119
	Edificações	3.678,833	114.607,424	118.286,257
	Total	327.698,549	118.994,827	446.693,376

Tabela 7.11 – Matriz de confusão do modelo na época 200 - RGB.

Modelo na época 200 - RGB				
		Predito		
		Background	**Edificações**	**Total**
Referência	***Background***	322.397,763	6.009,356	328.407,119
	Edificações	6.485,598	111.800,659	118.286,257
	Total	328.883,361	117.810,015	446.693,376

Tabela 7.12 – Métricas do modelo *HInDSM* na época 203.

%	***Background***	**Edificações**
Precision	98,877	96,313
Recall	98,664	96,890
Accuracy	98,194	98,194
F1-Score	98,771	96,601
MCC	95,372	95,372

Tabela 7.13 – Métricas do modelo RGB na época 200.

%	***Background***	**Edificações**
Precision	98,028	94,899
Recall	98,170	94,517
Accuracy	97,203	97,203
F1-Score	98,099	94,708
MCC	92,807	92,807

Tabela 7.14 – Diferença percentual entre as métricas estimadas para o modelo *HInDSM* (época 203) e *RGB* (época 200).

Diferença (%)	***Background***	**Edificações**
Precision	+0,85	+1,41
Recall	+0,49	+2,37
Accuracy	+0,99	+0,99
F1-Score	+0,67	+1,89
MCC	+2,56	+2,56

Com os resultados da validação para cada modelo, as Figuras 7.28 e 7.29 foram construídas, e apresentam a evolução do treinamento no *dataset* de validação com o decorrer das épocas. É possível identificar que o *dataset RGB* possui maior variabilidade que o visto no *dataset HInDSM*, ou seja, incluir dados de nDSM possibilita um treinamento com baixa oscilação entre as épocas, com um crescimento contínuo e com maior convergência.

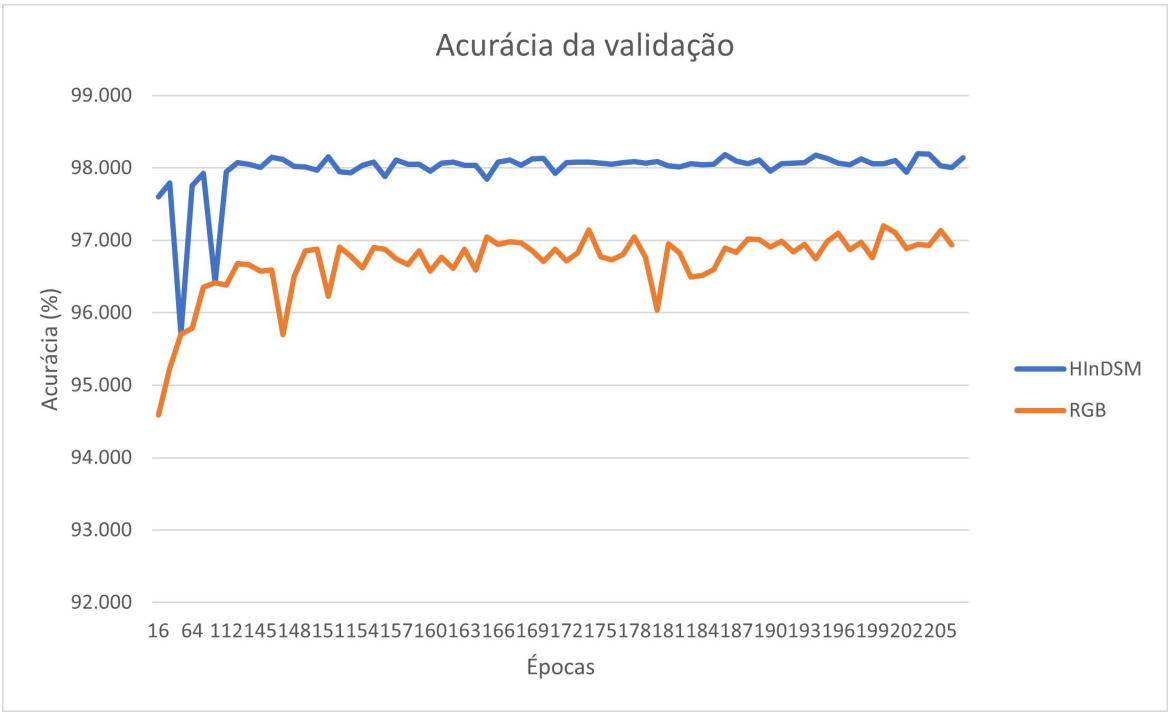


Figura 7.28 – Acurácia da validação.

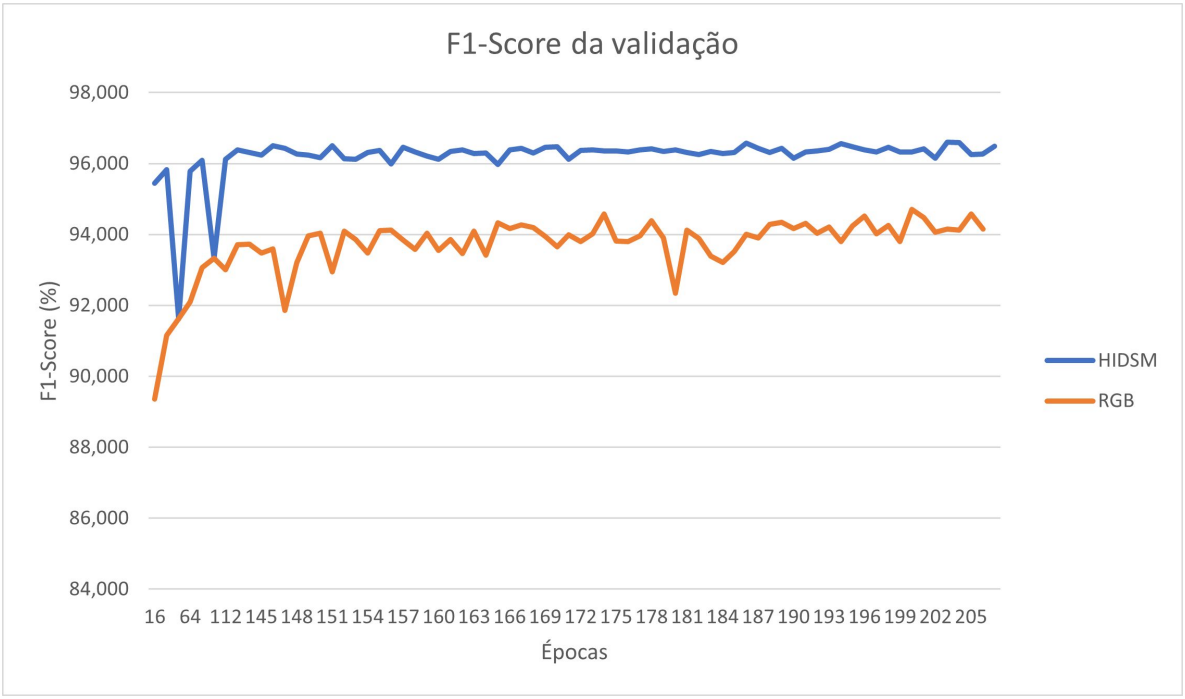


Figura 7.29 – F1-Score da validação.

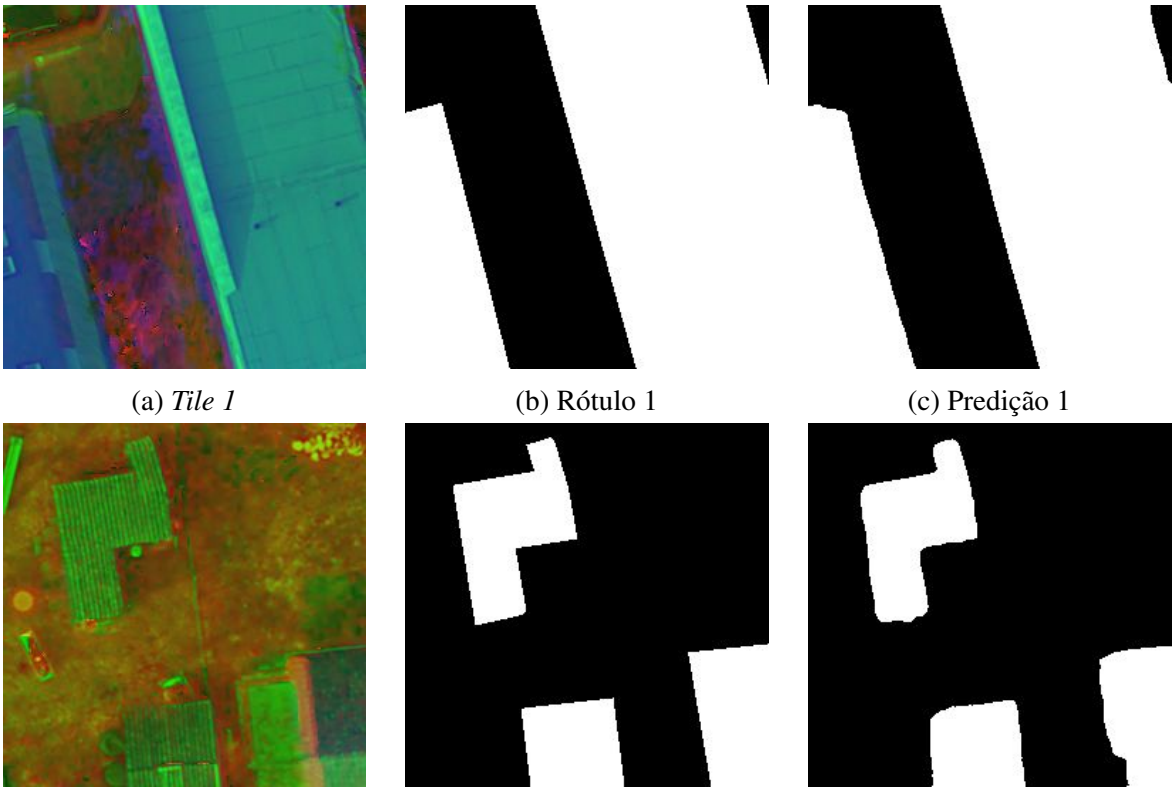
A partir das métricas apresentadas nesta seção, ficou evidente que os modelos escolhidos inicialmente não atingiam as maiores métricas na validação dos dados. Isto se deve ao fato do *dataset* de validação ser inédito para a arquitetura, e desta forma os valores obtidos diferem dos obtidos na etapa de treinamento. Como num primeiro momento a função custo e a acurácia calculada são feitas no *dataset* de treinamento, é natural que as métricas tenham pequena variação

no *dataset* de validação. Deste modo, mudou-se a escolha do modelo de parâmetros para o melhor valor de *acurácia* e *F1-score* baseando-se nos resultados da validação: modelo na época 203 no *dataset HInDSM*; e modelo na época 200 no *dataset RGB*.

Na próxima seção, os dois primeiros modelos de parâmetros serão comparados qualitativamente com os dois modelos finais para fins de verificação de melhorias. Para facilitar o entendimento, os modelos serão nomeados com o nome do *dataset* e o número de épocas: *HInDSM_205, HInDSM_203, RGB_201*, *RGB_200*, *HInDSM_203* e *RGB_200*.

7.2.2 Avaliação Qualitativa

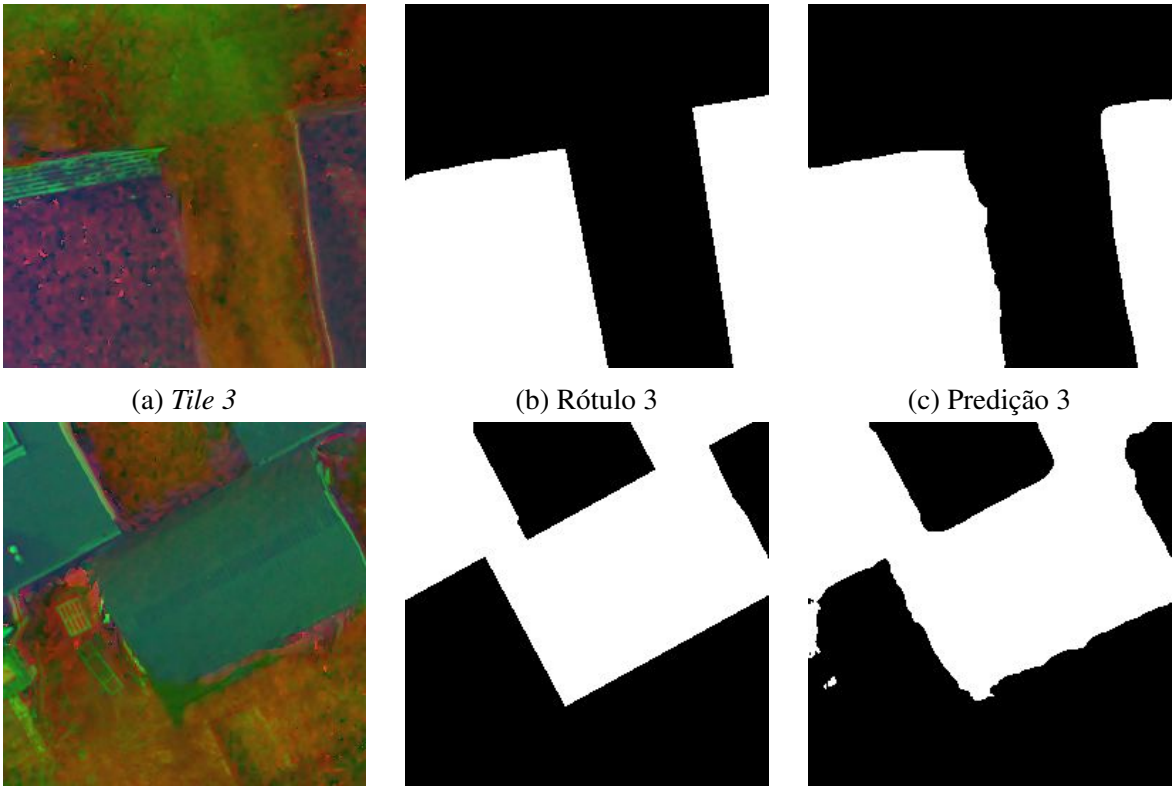
Para a avaliação qualitativa, os *tiles* de validação foram selecionados em diferentes regiões. Para o modelo *HInDSM_205* a Figura 7.30 apresenta na primeira coluna os *tiles* inseridos na predição, o rótulo e a camada de saída (ou predição). Na Figura 7.31 outra região é mostrada.



(a) *Tile 1* (b) Rótulo 1 (c) Predição 1

(d) *Tile 2* (e) Rótulo 2 (f) Predição 2

Figura 7.30 – Predição 1 com o modelo *HInDSM_205* em *tiles* de validação.



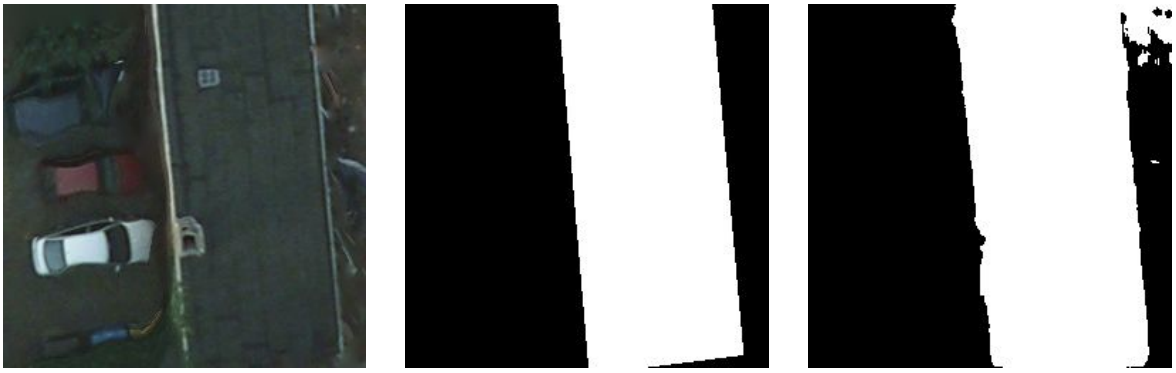
(a) *Tile 3* (b) Rótulo 3 (c) Predição 3

(d) *Tile 4* (e) Rótulo 4 (f) Predição 4

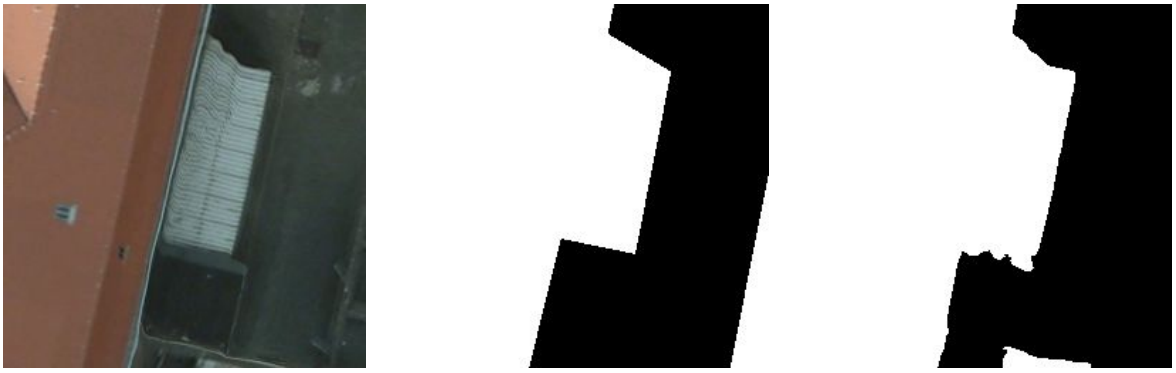
Figura 7.31 – Predição 2 com o modelo *HInDSM_205* em *tiles* de validação.

É possível inferir pela Figura 7.30f que a predição possui cantos mais arredondados que os rótulos e, apesar disso, a predição tende a acompanhar mais o contorno da edificação, incluindo as distorções da ortofoto encontradas na Figura 7.30d. O mesmo fenômeno é visível na Figura 7.31f, possuindo cantos com distorção correspondendo ao que é visto na Figura 7.31d. Os cantos mais arredondados na predição se devem ao fato da rede inferir pixel-a-pixel, não havendo camadas de generalização. No geral, a predição em destaque possui baixo ruído e grande correspondência com as edificações.

Nas Figuras 7.32 e 7.33 são apresentadas as predições elaboradas pelo modelo *RGB_201*. Na Figura 7.32c percebe-se cantos mais arredondados, como constatados no modelo *HInDSM_205*, mas com a presença de uma região de falso positivo ao se comparar com o rótulo (Figura 7.32b). Nas Figuras 7.33a e 7.33d também fica evidente a presença de regiões com falso negativo. No geral, o modelo *RGB_201* apresenta mais ruídos e distorções, e regiões com falsos positivos e negativos, ao se comparar com o modelo *HInDSM_205*.

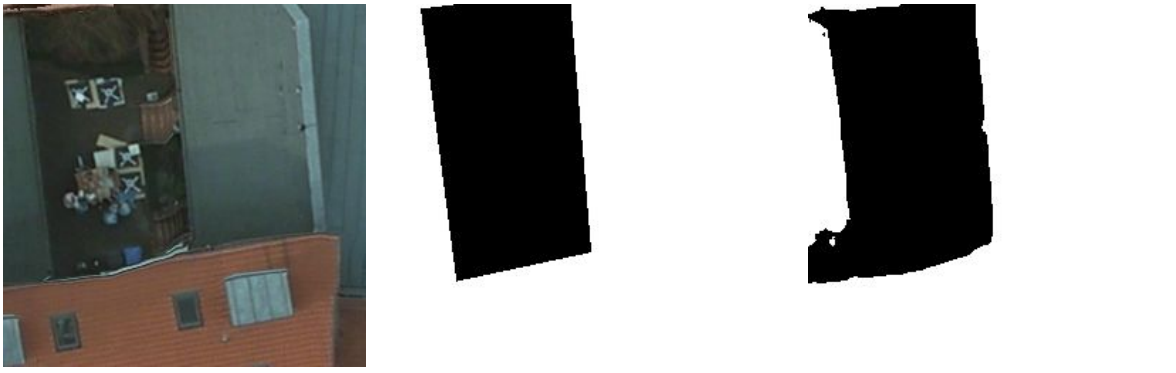


(a) *Tile 5* (b) Rótulo 5 (c) Predição 5



(d) *Tile 6* (e) Rótulo 6 (f) Predição 6

Figura 7.32 – Predição 3 com o modelo *RGB_201* em *tiles* de validação.

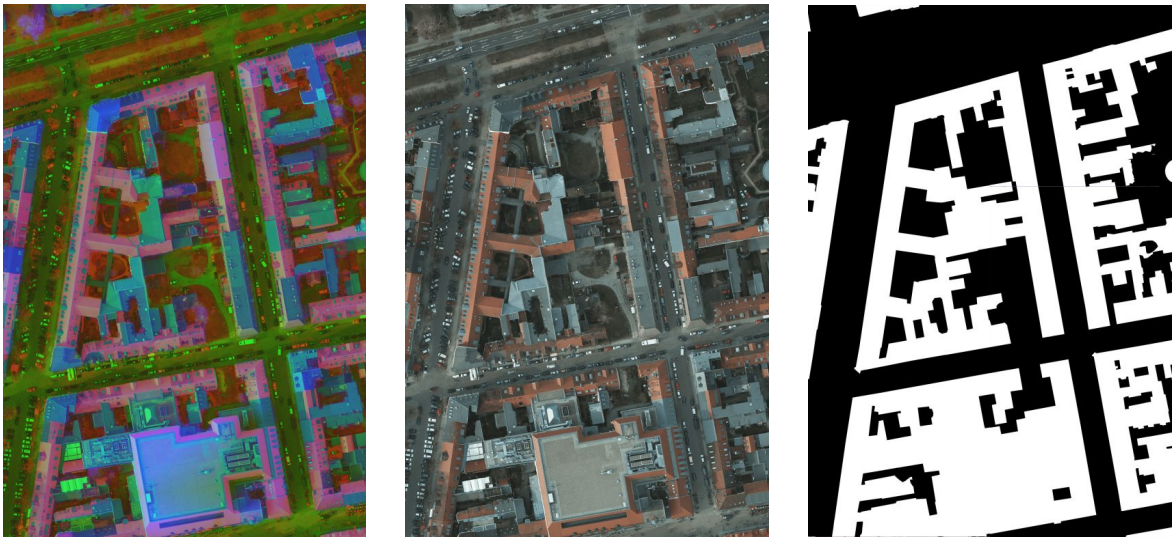


(a) *Tile 7* (b) Rótulo 7 (c) Predição 7

(d) *Tile 8* (e) Rótulo 8 (f) Predição 8

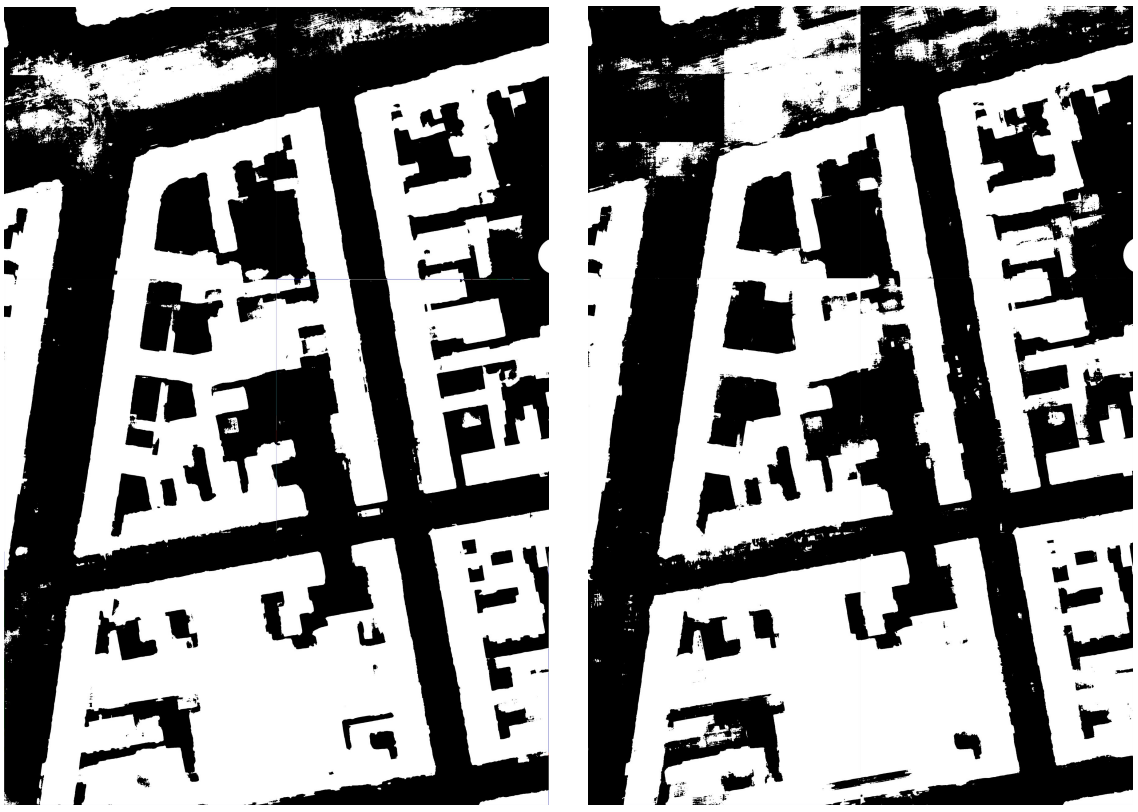
Figura 7.33 – Predição 4 com o modelo *RGB_201* em *tiles* de validação.

Para uma avaliação de uma área maior que as encontradas nos *tiles* mostrados anteriormente, foram selecionadas duas regiões contendo áreas vistas e não-vistas pela arquitetura. O objetivo desta composição é analisar visualmente o desempenho em uma região completa. Na Figura 7.34 vê-se o *dataset HInDSM* e a inferência na Figura 7.35. O *dataset RGB* e a inferência estão nas Figuras 7.36 e 7.37, respectivamente.



(a) *HInDSM* da área 1 (b) *RGB* da área 1 (c) Rótulo da área 1

Figura 7.34 – Dados dos *HInDSM* e *RGB* da área 1.

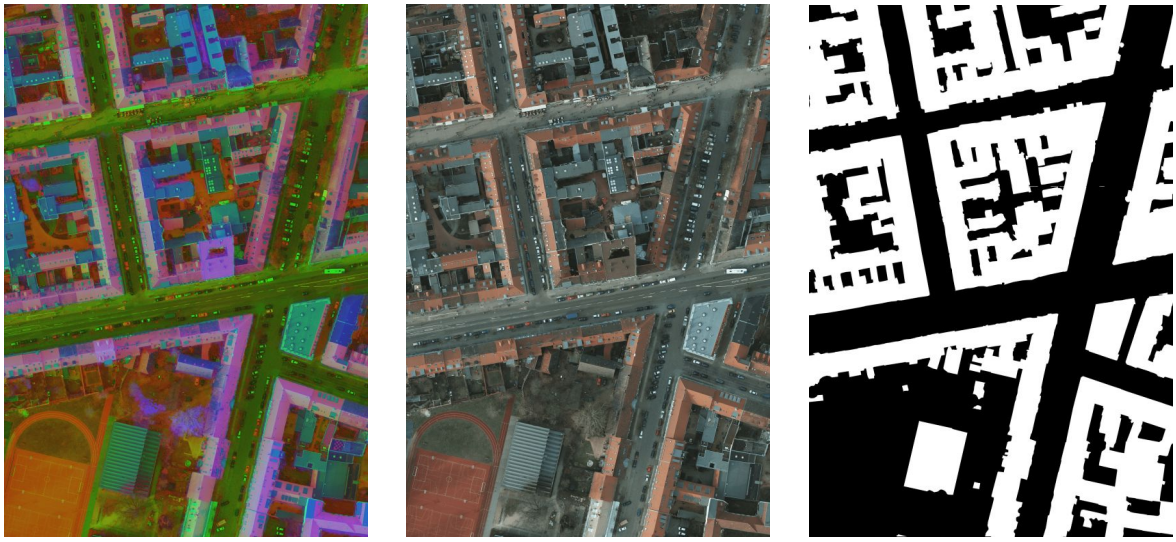


(a) Inferência da área 1 com *HInDSM_205* (b) Inferência da área 1 com *RGB_201*

Figura 7.35 – Predição da área 2 no *RGB_201* e *HInDSM_205*.

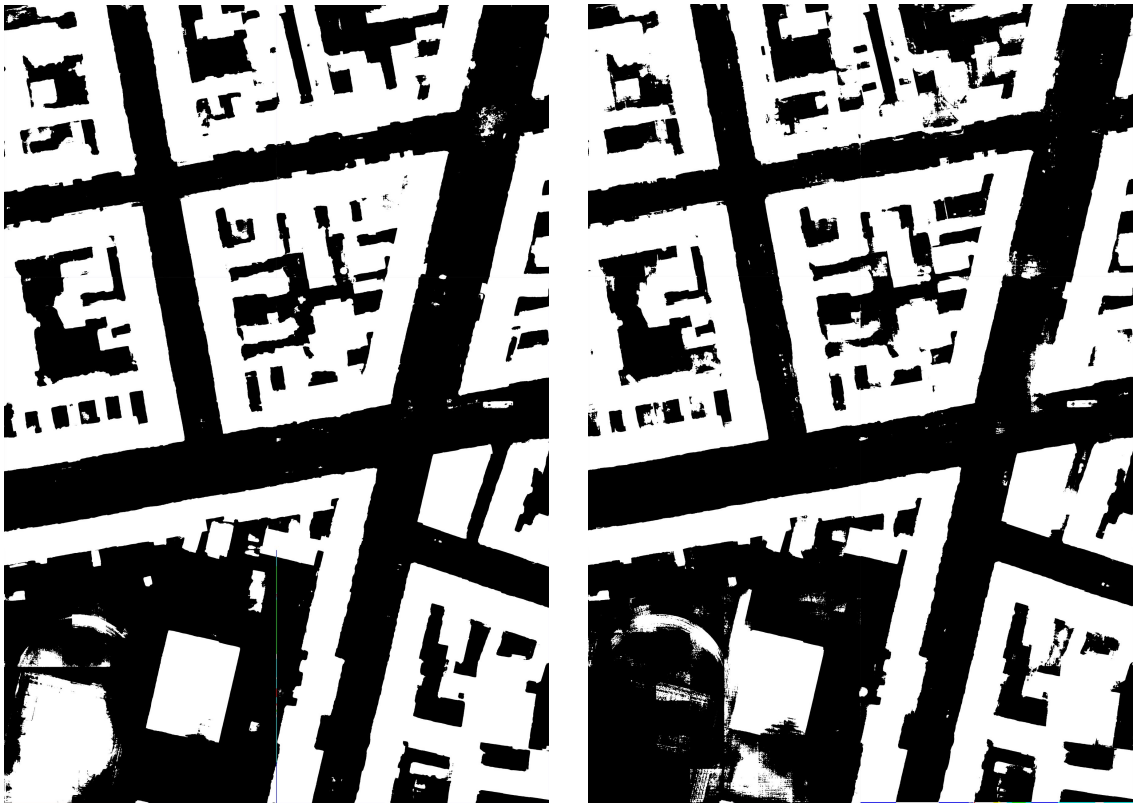
Ao se comparar as Figuras 7.35a e 7.35b, verifica-se uma tendência de identificação do canteiro central como edificação nas duas metodologias (canto superior direito), sendo mais intensa na realizada com o *dataset RGB*. Outro ponto de destaque é a presença de falsos positivos mais intensos no *RGB_201* que no *HInDSM_205*. Além disso, o *RGB* apresenta mais ruídos e deformidades.

Nas Figuras 7.37a e 7.37b, uma pista de atletismo localizada no canto inferior esquerdo foi erroneamente segmentada como edificação no *HInDSM*. Tal fato ocorre com menor intensidade no *RGB_201*, apesar de parte da pista estar presente. Não obstante, o *RGB_201* tem mais dificuldades em regularização das edificações nesta mesma região, devido à presença de vegetação, o que ocorre com menor extensão no *HInDSM_205*. Outro ponto a se observar é que a segmentação de edificações tem maior correspondência com o rótulo no *HInDSM_205*, com menos ruídos e distorções.



(a) *HInDSM* da área 2 (b) *RGB* da área 2 (c) Rótulo da área 2.

Figura 7.36 – Dados dos *datasets HInDSM* e *RGB* da área 2

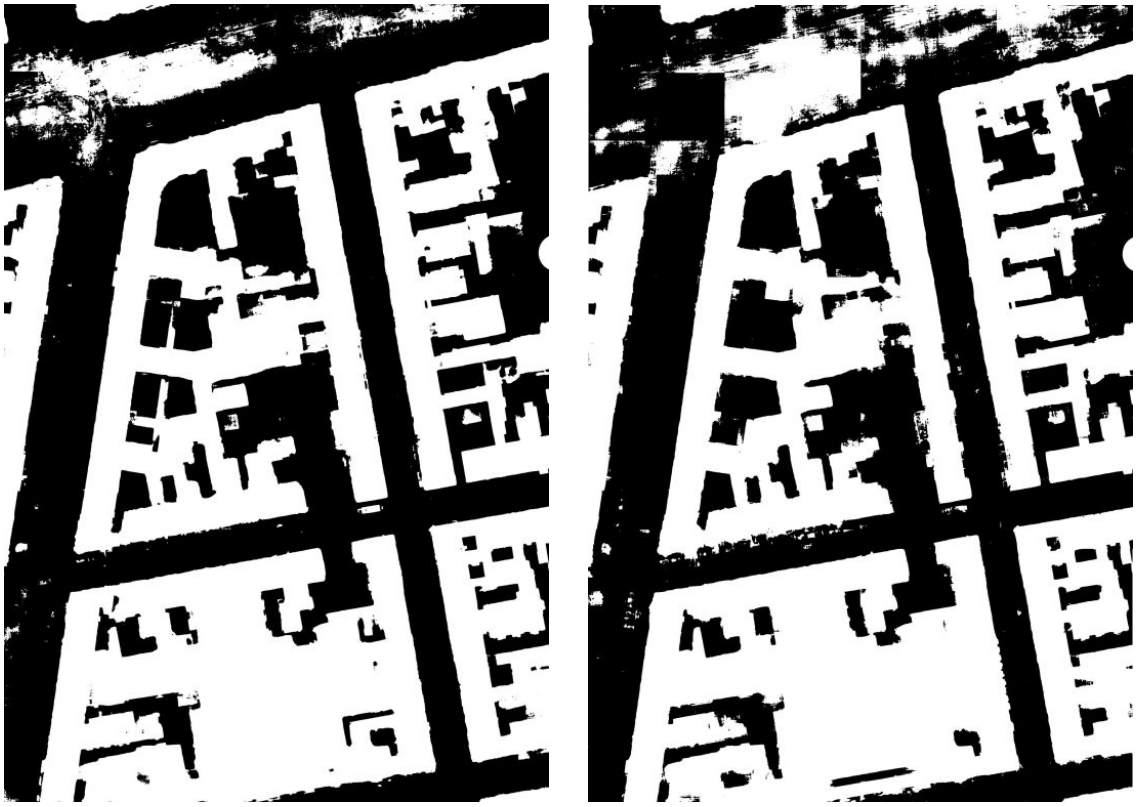


(a) Inferência da área 2 com *HInDSM_205* (b) Inferência da área 2 com *RGB_201*

Figura 7.37 – Predição da área 1 no *RGB_201* e *HInDSM_205*.

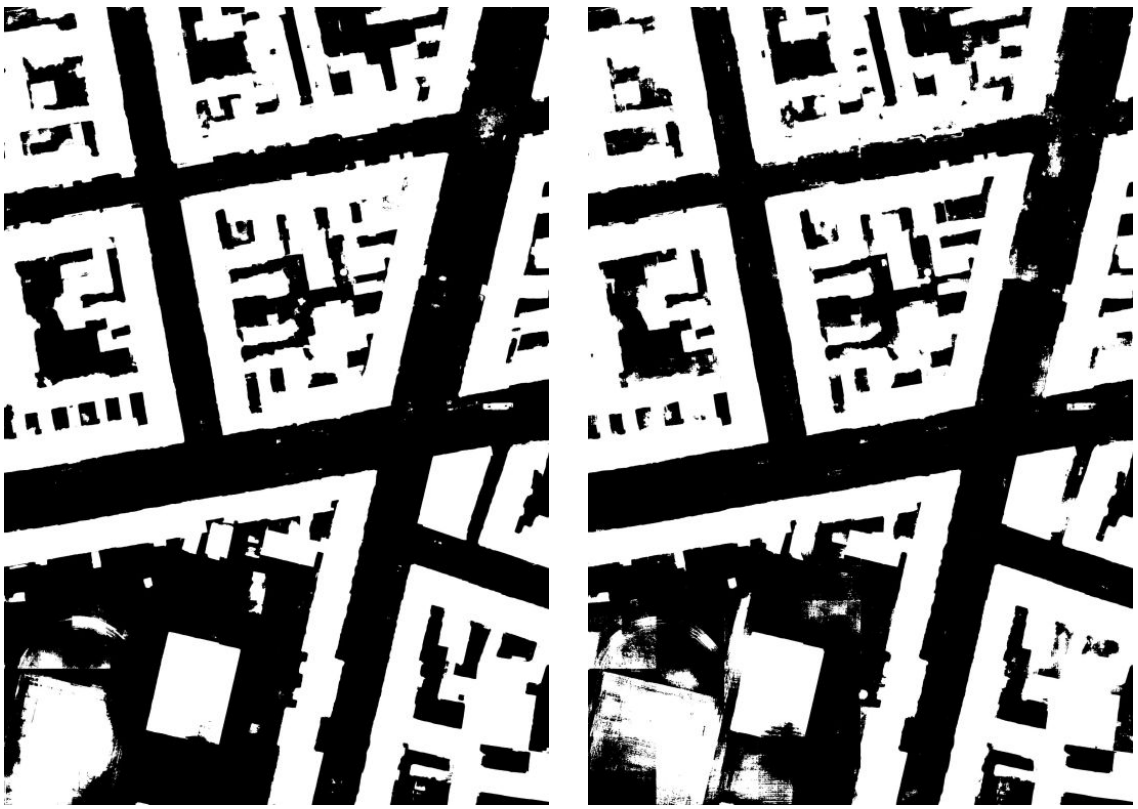
Apesar de ambos os *datasets* apresentam erros visuais de segmentação, ao considerar o panorama apresentado, as edificações são mais bem definidas no *HInDSM_205* do que as vistas no *RGB_201*, com menos distorções e ruídos, e baixa quantidade de falsos positivos pela análise qualitativa. Constata-se que, em uma equidade de cenário, o produto do treinamento com o modelo digital de elevação permite melhor segmentação de edificações em meio urbano, do que o visto no *RGB_201*.

Ao comparar o modelo com melhor métrica no *dataset* de validação (*HInDSM_205*) com o modelo com a melhor métrica no *dataset* de treinamento (*HInDSM_203*), percebe-se que há pouca variação visível, com a presença de mais ruídos no *HInDSM_203* que no *HInDSM_205*. Entre os modelos *RGB_200* e *RGB_201* também tem poucos elementos diferenciáveis, com mais intensidade nos ruídos. Apesar disso, o *HInDSM_203* apresenta os melhores resultados em comparação com o *RGB_200*, como visto nas duas áreas analisadas nas Figuras 7.38 e 7.39.



(a) Predição da área 1 com *HInDSM_203* (b) Predição da área 1 com *RGB_200*

Figura 7.38 – Predição da área 1 no *HInDSM_203* e *RGB_200*.



(a) Predição da área 2 com *HInDSM_203* (b) Predição da área 2 com *RGB_200*

Figura 7.39 – Predição da área 2 no *HInDSM_203* e *RGB_200*.

Considerando o exposto, pode-se inferir que o melhor modelo de predição obtido nos treinamentos da metodologia proposta foi o *HInDSM_203*, com baixo ruído e alta correspondência com a referência. Apesar de o modelo *RGB_200* ter alcançado as melhores métricas entre o treinamento do *RGB*, quando comparado com o produzido pelo *HInDSM* apresenta mais ruídos e algumas falhas na predição.

8 CONCLUSÕES

Os resultados encontrados no modelo usando a composição de cores *Hue* e Intensidade unido com o modelo digital de superfície normalizado demonstrou ser superior ao uso do modelo tradicional (com a composição de cores *RGB*). Apesar do resultado positivo visto nas métricas, como o ganho de 1,89% em termos de *F1-Score*, outros pontos se destacaram na metodologia proposta, como a estabilidade do treinamento, alta convergência e consequente redução do tempo de treinamento do modelo de parâmetros. Ao comparar o uso do modelo *HInDSM* ao uso das componentes *RGB* foi possível identificar que o uso do primeiro modelo permitiu uma convergência mais rápida da rede, tanto em termos de custo quanto acurácia.

O destaque do *HInDSM* pode ser justificado por permitir a arquitetura reconhecer as edificações em meio urbano devido à presença da informação de altura. O uso desta informação convertida em uma imagem com níveis de cinza facilita ao modelo diferenciá-los de outros objetos urbanos, como vias, carros e vegetação. O emprego do modelo de cor *Hue* e Intensidade como camadas da imagem composta também trazem maior contexto ao treinamento.

A cada canal adicionado no processamento, aumenta-se o custo computacional para o aprendizado. Deste modo, apesar de ser viável adicionar o *MDSn* em níveis de cinza juntamente com a informação de cor, o custo computacional se amplia quando comparado ao utilizado nesta pesquisa devido à presença de mais um canal de processamento pela arquitetura, limitando mais o uso de GPUs nos processos de *deep learning*. Deste modo, alternativas como a proposta nesta pesquisa, que reduzem o número de dados com melhoria dos resultados, trazem avanços aos estudos de segmentação de edificações por *deep learning*.

A principal limitação de *hardware* detectada na pesquisa foi a impossibilidade de se aumentar o *batch* do treinamento devido ao tamanho da memória *VRAM* presente nas máquinas do *Google Colab*. O problema decorre do processamento de imagens ter alto consumo de memória. Tal limitação foi superada ao utilizar a metodologia de agregação manual de gradientes, permitindo o *batch* ser aumentado para 264 *tiles* para cada atualização dos pesos, trazendo mais estabilidade aos processos e maior contexto para a arquitetura durante o treinamento.

Apesar de a segmentação ter atingido 96,6% de *F1-Score*, o tempo de treinamento da arquitetura ainda é um fator a se considerar. O alto custo computacional havia sido detectado no experimento preliminar realizado, antes do treinamento da metodologia proposta, e apesar do número reduzido de canais ter diminuído o tempo de processamento, ainda é um custo computacional considerável da arquitetura utilizada. O tempo de treinamento poderia ser reduzido pela sincronização com outras *GPUs*, dividindo os processos entre cada instância e utilizando a técnica de agregação manual de gradientes para o cálculo dos pesos.

Na análise qualitativa a metodologia proposta também apresentou melhores resultados

no modelo *HInDSM*, com baixo ruído e alta correspondência com a referência. Apesar disto, em ambos os modelos há erros na segmentação de edificações, que corresponde ao visto nas métricas avaliadas. Os principais pontos identificados foram a dificuldade da arquitetura com objetos urbanos sem altura, como canteiros de avenidas e quadras esportivas. Ao comparar os modelos, o modelo *HInDSM* conseguiu reduzir os problemas mencionados, mas não extinguiu-os. Não obstante, o modelo *HInDSM* tem mais regularidade nas edificações.

Os resultados produzidos poderiam ser aplicados na atualização rápida e constante de informações urbanas requeridas pelos gestores públicos. De fato, o tempo de processamento ainda é um fator determinante no treinamento das arquiteturas de *deep learning*, que devem ser reduzidos com o aprimoramento das técnicas e métodos, e a maior disponibilidade de *hardware* eficientes em processos de aprendizagem profunda. Após o modelo treinado, a inferência é rápida e pode ser realizada continuamente em imagens ópticas e dados *LiDAR*, permitindo aplicações como detecção de expansão urbana, uso irregular do solo, atualização de atributos e outras aplicações. Além do meio urbano, trabalhos futuros podem verificar a aplicação da metodologia na melhoria dos modelos de *deep learning* na detecção de desmatamento e pontos de fogo em áreas de proteção ambiental.

Conclui-se que a acurácia e o desempenho na segmentação semântica de edificações em meio urbano, por *deep learning*, foi aprimorado pela combinação de imagens ópticas e dados LiDAR, validando a hipótese inicialmente colocada para esta pesquisa.

REFERÊNCIAS

AHMED, N.; MAHBUB, R. B.; RAHMAN, R. M. Learning to extract buildings from ultra-high-resolution drone images and noisy labels. *International Journal of Remote Sensing*, Taylor & Francis, v. 41, n. 21, p. 8216–8237, 2020. ISSN 0143-1161. DOI: https://doi.org/10.1080/01431161.2020.1763496.

AMORIM, A.; PELEGRINA, M. A.; JULIÃO, R. P. *Cadastro e gestão territorial: uma visão luso-brasileira para a implementação de sistemas de informação cadastral nos municípios*. São Paulo: Editora Unesp Digital, 2018. ISBN 9788595462823.

APACHE-MXNET. *MXNet*. 2022. Disponível em: <https://mxnet.apache.org/versions/1.5.0/>. Acesso em: 2022-03-08.

AWRANGJEB, M.; RAVANBAKHSH, M.; FRASER, C. S. Automatic detection of residential buildings using LIDAR data and multispectral imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 65, n. 5, p. 457–467, 2010. ISSN 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2010.06.001.

BADRINARAYANAN, V.; KENDALL, A.; CIPOLLA, R. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 39, n. 12, p. 2481–2495, dez. 2017. ISSN 1939-3539. DOI: https://doi.org/10.1109/TPAMI.2016.2644615.

BERALDIN, J. A.; BLAIS, F.; LOHR, U. Laser scanning technology. In: VOSSELMAN, G.; MAAS, H.-G. (Org.). *Airborne and Terrestrial Laser Scanning*. 1. ed. Dunbeath: CRC Press, 2010. p. 1–39. ISBN 978-1-4398-2798-7.

BRENNER, C. Building extraction. In: VOSSELMAN, G.; MAAS, H.-G. (Org.). *Airborne and Terrestrial Laser Scanning*. 1. ed. Dunbeath: CRC Press, 2010. p. 169–207. ISBN 978-1-4398-2798-7.

BRIESE, C. Extraction of digital terrain models. In: VOSSELMAN, G.; MAAS, H.-G. (Org.). *Airborne and Terrestrial Laser Scanning*. 1. ed. Dunbeath: CRC Press, 2010. p. 135–163. ISBN 978-1-4398-2798-7.

BUJÁN, S. et al. Land use classification from lidar data and ortho-images in a rural area. *The Photogrammetric Record*, v. 27, n. 140, p. 401–422, 2012. DOI: https://doi.org/10.1111/j.1477-9730.2012.00698.x.

CASTREJON, L. et al. Annotating Object Instances with a Polygon-RNN. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, 2017. p. 4485–4493. ISBN 978-1-5386-0457-1. DOI: https://doi.org/10.1109/CVPR.2017.477.

CHEN, L.-C. et al. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In: FERRARI, V. et al. (Ed.). *Computer Vision – ECCV 2018*. Cham: Springer International Publishing, 2018. v. 11211, p. 833–851. ISBN 978-3-030-01233-5. DOI: https://doi.org/10.1007/978-3-030-01234-2_49.

CHENG, D. et al. DARNet: Deep Active Ray Network for Building Segmentation. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2019. p. 7423–7431. DOI: https://doi.org/10.1109/CVPR.2019.00761.

DENG, L.; YU, D. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, v. 7, n. 3–4, p. 197–387, 2014. ISSN 1932-8346. DOI: https://doi.org/10.1561/2000000039.

DIAKOGIANNIS, F. I. et al. ResUNet-a: A deep learning framework for semantic segmentation of remotely sensed data. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 162, p. 94–114, abr. 2020. ISSN 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2020.01.013.

EL-SHEIMY, N.; VALEO, C.; HABIB, A. *Digital terrain modeling: acquisition, manipulation, and applications*. Boston: Artech House, 2005. ISBN 978-1-58053-921-0.

ETTEN, A. V. City-scale road extraction from satellite imagery v2: Road speeds and travel times. In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. [S.l.: s.n.], 2020. p. 1775–1784. ISSN 2642-9381. DOI: https://doi.org/10.1109/WACV45572.2020.9093593.

FREUDENBERG, M. et al. Large scale palm tree detection in high resolution satellite images using U-Net. *Remote Sensing*, Multidisciplinary Digital Publishing Institute (MDPI), v. 11, n. 3, p. 312, 2019. DOI: https://doi.org/10.3390/rs11030312.

GONZALEZ, R. C.; WOODS, R. E. *Digital image processing*. 3. ed. Upper Saddle River, N.J: Prentice Hall, 2008. ISBN 978-0-13-168728-8.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, Massachusetts: The MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>. Acesso em: 20 abr. 2021.

HE, K. et al. Mask R-CNN. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2017. p. 2980–2988. ISSN 2380-7504. DOI: https://doi.org/10.1109/ICCV.2017.322.

HE, K. et al. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 2016. p. 770–778. ISBN 978-1-4673-8851-1. DOI: https://doi.org/10.1109/CVPR.2016.90.

HUANG, J. et al. Automatic building extraction from high-resolution aerial images and LiDAR data using gated residual refinement network. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 151, p. 91–105, maio 2019. ISSN 09242716. DOI: https://doi.org/10.1016/j.isprsjprs.2019.02.019.

INTEL. *2D Max Pooling Forward Layer*. 2017. Disponível em: <https://software.intel.com/sites/products/documentation/doclib/daal/daal-user-and-reference-guides/daal_prog_guide/GUID-CCB814DD-945A-46DD-989A-8BC39D2D01CA.htm>. Acesso em: 15 out. 2021.

IOFFE, S.; SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. DOI: http://doi.org/10.48550/ARXIV.1502.03167.

ISPRS-WG-III/4. *2D Semantic Labeling*. 2018. Disponível em: <https://www2.isprs.org/commissions/comm2/wg4/benchmark/semantic-labeling>. Acesso em: 20 abr. 2021.

JAIN, A. K. *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall, 1989. (Prentice Hall information and system sciences series). ISBN 978-0-13-336165-0.

JIAO, C. et al. Burn image segmentation based on Mask Regions with Convolutional Neural Network deep learning framework: more accurate and more convenient. *Burns and Trauma*, v. 7, 02 2019. ISSN 2321-3876. DOI: https://doi.org/10.1186/s41038-018-0137-9.

JUPYTER. *JupyterNotebook*. 2022. Disponível em: <https://jupyter.org>. Acesso em: 2022-03-08.

KANG, W. et al. EU-Net: An Efficient Fully Convolutional Network for Building Extraction from Optical Remote Sensing Images. *Remote Sensing*, v. 11, n. 23, p. 2813, nov. 2019. ISSN 2072-4292. DOI: https://doi.org/10.3390/rs11232813.

KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995. v. 2, p. 1137–1143. ISBN 1558603638. DOI: https://doi.org/10.5555/1643031.1643047.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, v. 521, n. 7553, p. 436–444, 2015. ISSN 1476-4687. DOI: https://doi.org/10.1038/nature14539.

LI, Z.; WEGNER, J. D.; LUCCHI, A. Topological Map Extraction From Overhead Images. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. Seoul, Korea (South): IEEE, 2019. p. 1715–1724. ISBN 978-1-72814-803-8. DOI: https://doi.org/10.1109/ICCV.2019.00180.

LI, Z.; ZHU, Q.; GOLD, C. *Digital terrain modeling: principles and methodology*. New York: CRC Press, 2005. ISBN 978-0-415-32462-5.

MENESES, P. R. Modelos de cores aplicados às imagens. In: MENESES, P. R.; ALMEIDA, T. d. (Org.). *Introdução ao processamento de imagens de sensoriamento remoto*. Brasília: Universidade de Brasília e CNPq, 2012. p. 121–134.

MEYER, L.; LEMARCHAND, F.; SIDIROPOULOS, P. A deep learning architecture for batch-mode fully automated field boundary detection. *ISPRS Journal of Photogrammetry and Remote Sensing*, XLIII-B3-2020, p. 1009–1016, 2020. ISSN 2194-9034. DOI: https://doi.org/10.5194/isprs-archives-XLIII-B3-2020-1009-2020.

MILLER, C. L.; LAFLAMME, R. A. *The Digital Terrain Model: Theory & Application*. [S.l.]: MIT Photogrammetry Laboratory, 1958.

MIROSŁAW-ŚWIąTEK, D. et al. Developing an algorithm for enhancement of a digital terrain model for a densely vegetated floodplain wetland. *Journal of Applied Remote Sensing*, SPIE, v. 10, n. 3, p. 1 – 16, 2016. DOI: https://doi.org/10.1117/1.JRS.10.036013.

MISHRA, D. *Transposed Convolution Demystified*. 2020. Disponível em: <https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba>. Acesso em: 2021-06-06.

MOUJAHID, A. *A Practical Introduction to Deep Learning with Caffe and Python // Adil Moujahid // Data Analytics and more*. 2016. Disponível em: <http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/>. Acesso em: 2022-03-08.

NASCIMENTO, G. A. G. d. *Verificação da Aplicabilidade de Dados Obtidos por Sistema LASER Batimétrico Aerotransportado à Cartografia Náutica*. Dissertação (Mestrado em Ciências Cartográficas) — Universidade Estadual Paulista (Unesp), Faculdade de Ciências e Tecnologia, Presidente Prudente, 2019. Disponível em: <https://repositorio.unesp.br/handle/11449/181407>. Acesso em: 20 abr. 2021.

PEDRINI, H.; SCHWARTZ, W. R. *Análise de imagens digitais princípios, algoritmos e aplicações*. São Paulo: Thomson Learning, 2008. OCLC: 319215118. ISBN 978-85-221-0595-3.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted Intervention – (MICCAI)*. Cham: Springer International Publishing, 2015. p. 234–241. ISBN 978-3-319-24574-4. DOI: https://doi.org/10.1007/978-3-319-24574-4_28.

SANTOS, R. C. dos; GALO, M.; CARRILHO, A. C. Building boundary extraction from lidar data using a local estimated parameter for alpha shape algorithm. *The Int. Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, ISPRS TC I Mid-term Symposium "Innovative Sensing - From Sensors to Methods and Applications"*, V.XLII-1, p. 127–132, 2018. ISSN 2194-9034. DOI: https://doi.org/10.5194/isprs-archives-XLII-1-127-2018.

SANTOS, R. C. dos; GALO, M.; HABIB, A. F. Regularization of Building Roof Boundaries from Airborne LiDAR Data Using an Iterative CD-Spline. *Remote Sensing*, v. 12, n. 12, p. 1904, jun. 2020. ISSN 2072-4292. DOI: https://doi.org/10.3390/rs12121904.

SCIKIT-LEARN. *scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation*. 2022. Disponível em: <https://scikit-learn.org/stable/>. Acesso em: 2022-03-08.

SHAN, J.; TOTH, C. K. *Topographic laser ranging and scanning: principles and processing*. 1. ed. Boca Raton: CRC Press, 2008. ISBN 978-1-4200-5142-1.

SHI, Y.; LI, Q.; ZHU, X. X. Building footprint generation using improved generative adversarial networks. *IEEE Geoscience and Remote Sensing Letters*, v. 16, n. 4, p. 603–607, 2019. ISSN 1545-598X, 1558-0571. DOI: https://doi.org/10.1109/LGRS.2018.2878486.

SOHN, G.; DOWMAN, I. Data fusion of high-resolution satellite imagery and LiDAR data for automatic building extraction. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 62, n. 1, p. 43–63, 2007. ISSN 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2007.01.001.

SOKOLOVA, M.; JAPKOWICZ, N.; SZPAKOWICZ, S. Beyond accuracy, F-Score and ROC: A family of discriminant measures for performance evaluation. In: *AI 2006: Advances in Artificial Intelligence*. Berlin: Springer, 2006. p. 1015–1021. ISBN 978-3-540-49788-2. DOI: https://doi.org/10.1007/11941439_114.

TRANSFER, K. *PyTorch K-Fold Cross-Validation using Dataloader and Sklearn*. 2021. Disponível em: <https://androidkt.com/pytorch-k-fold-cross-validation-using-dataloader-and-sklearn/>. Acesso em: 01 set. 2021.

WANG, Y.; ZHANG, D.; DAI, G. Classification of high resolution satellite images using improved U-Net. *International Journal of Applied Mathematics and Computer Science*, v. 30, n. 3, p. 399–413, 2020. DOI: https://doi.org/10.34768/AMCS-2020-0030.

WEHR, A.; LOHR, U. Airborne laser scanning—an introduction and overview. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 54, n. 2-3, p. 68–82, jul. 1999. ISSN 09242716. DOI: https://doi.org/10.1016/S0924-2716(99)00011-8.

WEI, S.; JI, S.; LU, M. Toward Automatic Building Footprint Delineation From Aerial Images Using CNN and Regularization. *IEEE Transactions on Geoscience and Remote Sensing*, v. 58, n. 3, p. 2178–2189, mar. 2020. ISSN 0196-2892, 1558-0644. DOI: https://doi.org/10.1109/TGRS.2019.2954461.

WILKINSON, G. G. Results and implications of a study of fifteen years of satellite image classification experiments. *IEEE Transactions on Geoscience and Remote Sensing*, v. 43, n. 3, p. 433–440, 2005. ISSN 1558-0644. DOI: https://doi.org/10.1109/TGRS.2004.837325.

XU, B. et al. Livestock classification and counting in quadcopter aerial images using Mask R-CNN. *International Journal of Remote Sensing*, Taylor & Francis, v. 41, n. 21, p. 8121–8142, 2020. ISSN 0143-1161. DOI: https://doi.org/10.1080/01431161.2020.1734245.

XU, Y. et al. Building extraction in very high resolution remote sensing imagery using deep learning and guided filters. *Remote Sensing*, Multidisciplinary Digital Publishing Institute (MDPI), v. 10, n. 1, p. 144, 2018. DOI: https://doi.org/10.3390/rs10010144.

YEKEEN, S. T.; BALOGUN, A.; YUSOF, K. B. W. A novel deep learning instance segmentation model for automated marine oil spill detection. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 167, p. 190–200, 2020. ISSN 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2020.07.011.

YUAN, J. Learning building extraction in aerial scenes with convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 40, n. 11, p. 2793–2798, 2018. ISSN 1939-3539. DOI: https://doi.org/10.1109/TPAMI.2017.2750680.

ZHANG, Z.; LIU, Q.; WANG, Y. Road extraction by deep residual U-Net. *IEEE Geoscience and Remote Sensing Letters*, v. 15, n. 5, p. 749–753, 2018. ISSN 1558-0571. DOI: https://doi.org/10.1109/LGRS.2018.2802944.

ZHENG, X. et al. Parsing very high resolution urban scene images by learning deep ConvNets with edge-aware loss. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 170, p. 15–28, dez. 2020. ISSN 09242716. DOI: https://doi.org/10.1016/j.isprsjprs.2020.09.019.

Apêndices

APÊNDICE A – EXPERIMENTO PRELIMINAR

O experimento preliminar teve por objetivo identificar, com a execução da segmentação semântica prévia de um *dataset* pela arquitetura de *deep learning ResUNet-a*, os processos envolvidos na segmentação de edificações em imagens ópticas, analisando as limitações e procedimentos na preparação dos dados, treinamento do modelo e predição, incluindo a avaliação estatística dos resultados. Nesse sentido, este experimento preliminar fornece subsídios importantes para a definição da metodologia final considerada neste trabalho.

A.1 Metodologia do experimento preliminar

Para a execução deste experimento preliminar foram usados os seguintes materiais, além dos já descritos na seção de metodologia:

- Algoritmo da rede *ResUNet-a*;
- *Python* 3.7 com as bibliotecas: *MXNet* 1.5.1 para *CUDA* 9.2, *Pillow* 8.2.0, *pathos* 0.2.8, pandas 1.3.0, *Jupyter* 6.4.0, *imageio* 2.9.0, *OpenCV* 4.1.0.25, *Numpy* 1.17.3, *glob2* 0.4.1, *multiprocess* 0.70.12.2 e *rasterio* 0.32.0;
- Drivers *NVIDIA 471.41* e drivers CUDA com a biblioteca cuDNN 7.6.5;
- Computador equipado com uma *GPU Nvidia GTX 960* com *4 GB* de memória *VRAM* (*video random access memory*).

Todos os processos executados na metodologia seguiram os estudos de Diakogiannis et al. (2020) visando compatibilizar a análise dos resultados e entendimento de todos os processos da arquitetura *ResUNet-a*.

A.1.1 Preparação dos dados

Devido às condições de *hardware* disponíveis, o conjunto de dados teve de ser preparado visando o melhor desempenho computacional na execução da rede. Como a cada processo uma quantidade de memória é ocupada para armazenamento dos *tiles*, a sua dimensão definirá a quantidade de imagens processadas simultaneamente pela *GPU*, aumentando ou diminuindo a velocidade de execução. Dessa forma, as dimensões do *dataset* influenciaram no tempo de processamento. No caso, os dados possuem imagens com resolução (ou GDS – Ground Sample Distance) equivalente a 0,05 m, com dimensão de 6000 x 6000 pixeis. Para esse experimento, os dados usados foram: imagens R-G-B-NIR, MDS normalizado e rótulos com 6 classes (*background*, *car*, *tree*, *lowveg*, *building* e *imsurf*). Os processos de aprofundamento nas camadas com

esses dados exigiram um *hardware* com memória VRAM superior a 4 GB para a execução paralela dos processos. Nesse caso, identificou-se que para 4 GB de memória em GPU, a dimensão máxima nos processos com dois *tiles* simultâneos seria de 256 x 256 pixeis.

Um processo citado por Diakogiannis et al. (2020), que reduz a carga de dados na rede e pode melhorar o desempenho, é a degradação da resolução das imagens, que permite a introdução de mais informações dos rótulos em um único *tile*. Visando reduzir o tempo de processo neste *dataset*, um segundo conjunto de dados foi degradado no *software Quantum GIS* de modo que o GSD passasse de 0,05 m para 0,10 m, representando uma redução de 4x em termos de número de pixeis, reduzindo as dimensões de cada imagem para 3000 x 3000 *pixels*.

Com a dimensão definida, as imagens contendo as bandas R, G, B e NIR foram recortadas. Paralelamente, o mesmo recorte foi definido no MDSn e, por fim, combinando-os em um único *tile*, o mesmo processo foi executado para os rótulos. Todo o procedimento foi feito por algoritmos em *Python* fornecidos por Diakogiannis et al. (2020), usando a biblioteca *rasterio*, e salvos em formato *.npy* (para posterior leitura usando a biblioteca *NumPy*). Diferentes conjuntos de dados a partir do *dataset* de *Potsdam* foram gerados para o estudo, objetivando verificar a capacidade computacional, desempenho em treinamento e segmentação dos dados, que serão especificados nos próximos parágrafos. A Tabela A.1 apresenta algumas informações dos dados utilizados nestes treinamentos.

Tabela A.1 – Conjunto de dados preparados para o treinamento.

Conjunto	**Dimensão dos *tiles* (pixel)**	**Espaço alocado**	**Quantidade de *tiles* para treinamento**	**Quantidade de *tiles* para validação**
Potsdam256_4	256x256	54.4GB	9.720	1.872
Potsdam256_1	256x256	228GB	41.760	6.816
Potsdam128_4	128x128	57GB	41.760	6.816

O primeiro conjunto recortado resultou em 54,4 GB de dados, com 10% dos dados reservados aleatoriamente para validação, armazenando 9.720 *tiles* contendo as bandas R-G-B-NIR-MDSn e 9.720 *tiles* de rótulos usados no treinamento, e 1.872 *tiles* utilizados na validação. Estes dados obtidos a partir do *dataset* de *Potsdam* foram denominados *Potsdam256_4*.

Um segundo conjunto foi preparado usando o mesmo *dataset* para recorte sem degradação, com fins de avaliação do desempenho computacional mantendo a dimensão dos *tiles* (256 x 256), resultando em 228 GB de dados, com 41.760 *tiles* de treinamento, 41.760 *tiles* de rótulos e 6.816 *tiles* de validação. Estes dados obtidos a partir do *dataset* de *Potsdam* foram denominados *Potsdam256_1*.

Um terceiro conjunto foi preparado a partir do *dataset* degradado, com redução da dimensão dos *tiles* para 128 x 128 pixels. A quantidade de *tiles* para treinamento foi 41.760, sendo 6.816 utilizados na validação. Estes dados obtidos a partir do *dataset* de *Potsdam* foram denominados *Potsdam128_4*.

A.1.2 Treinamento do modelo de parâmetros da rede

Para os processos de treinamento e predição, a biblioteca *Apache MXNet 1.5.1* (APACHE-MXNET, 2022) foi utilizada para dar suporte às operações, com compatibilização das bibliotecas *CUDNN*, *drivers CUDA* e *NVIDIA* na versão 9.2. A *MXNet* é uma biblioteca altamente flexível, que permite treinamento rápido e construção de redes de *deep learning* com uso de *GPUs* e *CPUs* sincronizados, em múltiplas linguagens. A escolha da *MXNet* é motivada pela compatibilidade dos algoritmos da *ResUNet-a* com essa biblioteca. Inicialmente, as bibliotecas foram importadas no ambiente *Jupyter Notebook* (JUPYTER, 2022), incluindo as relacionadas à *ResUNet-a d6*.

Os *tiles* que serão processados são informados por um arquivo em *batch* (ou lote). Estes *tiles* serão vistos simultaneamente pela rede, sendo que o número de *tiles* que deve ser processado depende da configuração do *hardware* que será utilizado. Como o treinamento ocorreu em uma máquina com 4 GB de *VRAM*, os testes demonstraram a capacidade máxima do *batch* de 2 *tiles* para o conjunto de dados com dimensão de 256 x 256 pixels. De fato, para aumentar a quantidade de *tiles* processados simultaneamente, é necessário reduzir sua dimensão para 128 x 128. A consequência dessa redução é o aumento proporcional de *tiles* para processamento, não apresentando melhoras significativas no tempo de treinamento. Destaca-se que, ao usar um *batch* pequeno, não é possível garantir uma convergência global adequada dos parâmetros, devido ao algoritmo não ter acessado todo o *dataset*. Dessa forma, ter um lote pequeno pode impactar na taxa de aprendizagem que a rede terá. Para o *dataset Potsdam128_4*, o *batch* pode ser definido em 8 *tiles*, 4 vezes maior que os conjuntos anteriores, devido a ter *tiles* com dimensão reduzida na mesma proporção, ocupando o mesmo espaço. Outra medida que visa garantir o desempenho computacional foi ajustar a execução das iterações pela biblioteca *MXNet*, que por padrão realiza iterações paralelas, mas que teve de ser modificada para iterações sequenciais, ou seja, somente iniciar a próxima iteração quando a anterior estiver finalizada.

Para o treinamento, iniciou-se o carregamento dos pares de *tiles* na memória, conduzido de forma randomizada. Após seu carregamento, cada *tile* passa por um algoritmo para aplicação de parâmetros de normalização e transformação. A normalização decorre da necessidade de ajustar todas as imagens a um intervalo de ocorrência comum, garantindo que os gradientes sejam controlados e estejam no mesmo intervalo. As transformações permitem que a rede visualize o mesmo *tile* em diferentes posições, cores e ângulos, aumentando a quantidade de informação fornecida para o treinamento do modelo.

Os valores de normalização são encontrados realizando previamente o cálculo da média e o desvio-padrão de todo o *dataset* usado no processamento, para cada canal globalmente. Dessa forma, os *tiles* são normalizados em cada canal, na seguinte sequência: R-G-B-IR-MDSn.

A rotação ocorre por uma matriz de rotação no momento em que o *tile* é inserido na rede, partindo do centro dos *tiles* (0 – 256), com aproximação e afastamento (0,25 e 1,25), rotação em duas posições (-85º e +85º) e transformação para o espaço de cores HSV (*hue*, *saturation* e

value). Todos os parâmetros — de normalização e transformação — são os mesmos definidos no trabalho de Diakogiannis et al. (2020), com o ajuste do centro dos *tiles* quando usado o conjunto *Potsdam128_4* com valor de 0 a 128 pixeis.

Para início do treinamento, a rede é iniciada com a definição de dois parâmetros: tamanho do filtro inicial (do módulo de convolução e *ResBlock*) e o número de classes do *dataset* (seis classes). O tamanho do filtro é dobrado conforme a profundidade da rede, e, como a arquitetura usada possui 6 camadas de profundidade, ao iniciar o treinamento com um filtro de dimensão 32x32, chega-se à sexta camada com um filtro de 1024x1024.

As iterações se iniciam, e os *tiles* são copiados para a *VRAM* e processados pela arquitetura. Ao final da primeira iteração, com os primeiros *tiles*, a função *Tanimoto* quantifica o custo de usar os parâmetros encontrados e, consequentemente, ajusta os parâmetros para a próxima iteração. O valor médio da função custo de cada lote é armazenado e, ao final das épocas, o custo total do treinamento é calculado. Em cada época, todo o *dataset* é observado pela rede uma única vez, e o processo pode ser finalizado quando atingido o número máximo de épocas definido pelo usuário ou pela definição de um critério de parada.

Para o conjunto *Potsdam256_4*, a quantidade de épocas foi definida em 30, para o conjunto *Potsdam256_1* foram definidas apenas 15 épocas, e para o conjunto *Potsdam128_4* foram definidas 60 épocas. A determinação do número de épocas considerou o tempo de processamento computacional em cada *dataset*, sendo iguais para os dois primeiros conjuntos e o dobro para o último. Os conjuntos utilizaram os mesmos parâmetros de filtro inicial e número de classes, distinguindo-se apenas no tamanho do *batch* — 2 *tiles* para os conjuntos de dados *Potsdam256_4* e *Potsdam256_1*, e 8 *tiles* para o conjunto *Potsdam128_4*.

Embora seja possível calcular as métricas relacionadas em cada iteração, devido ao alto custo computacional e à potencial extrapolação de memória, optou-se por não obter as métricas de avaliação do modelo durante o treinamento, realizando essa avaliação apenas ao final. A única métrica estimada durante o treinamento foi obtida para o modelo com o conjunto *Potsdam128_4*, no qual a função custo retornou seus valores para a construção de um gráfico de evolução do aprendizado ao longo das épocas. Mesmo não sendo armazenada nos *datasets* anteriores, por representar uma etapa que reduziria a velocidade do treinamento, a função custo esteve em execução durante todas as iterações para o ajuste dos parâmetros.

A.1.3 Validação dos modelos

Ao realizar a validação dos resultados, foi utilizado o conjunto de dados reservado exclusivamente para essa finalidade, composto por 10% dos dados que não foram utilizados anteriormente pela rede durante o treinamento. Nesse processo, o modelo de parâmetros treinado foi aplicado a esse conjunto de dados para realizar a predição dos resultados.

O procedimento de validação seguiu uma abordagem semelhante ao treinamento. Inicial-

mente, os dados de validação foram carregados na memória e processados pela arquitetura em pequenos lotes, resultando em um conjunto de *tiles* preditos. Dentre os produtos gerados pela *ResUNet-a*, que incluem segmentação semântica com as seis classes, predição das bordas, mapa de distâncias inferido e imagem reconstruída, apenas a segmentação semântica foi considerada para a avaliação das métricas. Os demais itens foram empregados para suporte às análises, fornecendo percepções sobre o grau de aprendizado alcançado pela rede.

A avaliação dos resultados envolve a construção de uma matriz de confusão com múltiplas classes, além de uma verificação pixel-a-pixel. Para cada um dos modelos de parâmetros, essas análises são utilizadas para calcular métricas como *precision*, *recall*, *F1-Score*, *accuracy* e *Matthews Correlation Coefficient (MCC)*. Como os dados de validação são provenientes do mesmo *dataset* sendo escolhidos de forma randômica, sem terem sido utilizados pela rede no treinamento, essa avaliação permite verificar a capacidade do modelo em segmentar um conjunto inédito com características urbanas semelhantes.

As métricas foram recolhidas usando a biblioteca em *Python scikit-learn* (SCIKIT-LEARN, 2022), com o armazenamento para múltiplas classes, comparando-as com os rótulos verdadeiros. A biblioteca possui um módulo dedicado ao cálculo das métricas chamado *sklearn.metrics*, com uma seção de avaliação de classificações. Para avaliação dos resultados, o módulo *sklearn.metrics.confusion_matrix* foi escolhido por gerar uma matriz de confusão de múltiplas classes, definindo previamente os rótulos. Com a matriz de confusão construída, todas as métricas foram calculadas e avaliadas, sendo apresentadas na seção de resultados deste estudo.

A.2 Resultados e análise do experimento preliminar

Nesta seção, são apresentados os resultados do experimento preliminar realizado com os diferentes modelos gerados pela *ResUNet-a d6*, e respectivos conjuntos de dados e parâmetros de treinamento, buscando compreender a influência das modificações propostas na qualidade do treinamento.

A rede testada no experimento possui diversos módulos e etapas que aumentam o tempo de processamento dos *datasets*, ao comparar com outras arquiteturas, como a U-Net. Dessa forma, as escolhas envolvidas na preparação dos *datasets* e parâmetros de treinamento têm alto impacto no tempo de operação da rede e qualidade dos resultados.

No primeiro *dataset* (*Potsdam256_4*), a opção pela degradação das imagens, modificando o GSD de 0,05 m para 0,10 m de *GSD* (*Ground Sample Distance*), reduz o conjunto de dados de 228 GB (como possui o *Potsdam256_1*) para 54,4 GB, permitindo que a arquitetura processe o *dataset* mais rapidamente. Apesar da perda de informações envolvida no processo de degradação das imagens, a modificação permite maior contexto para a rede quando mantida a dimensão dos *tiles*, uma vez que cada *tile* corresponderá a uma área maior do conjunto de dados. Quando comparado com o segundo *dataset* (*Potsdam256_1*) sem degradação, a rede necessita de mais

tempo para percorrer o conjunto. Nos dois casos, a limitação do *hardware* impede que mais de dois *tiles* sejam processados simultaneamente pela rede. Nesse sentido, o terceiro conjunto (*Potsdam128_4*) foi gerado pela combinação da degradação das imagens com a redução da dimensão de cada *tile* para 128 x 128 pixels, permitindo à rede percorrer o conjunto em menos tempo, com 8 *tiles* simultâneos.

O tempo de execução do *dataset Potsdam256_4* foi de 6.500 segundos para cada época (resultando num total de 54 horas), no *Potsdam256_1* de 12.000 segundos para cada época (50 horas), e no *Potsdam128_1* foi de 5.600 segundos para cada época (93,34 horas), com variações de ± 100 segundos em cada iteração. Apesar da proximidade de tempo entre o primeiro e o segundo conjunto, é importante destacar que o segundo permite que a rede tenha mais contexto inicial devido ao *batch* ser maior.

Outro fator de destaque e de grande impacto no desempenho computacional é a baixa quantidade de memória *VRAM* disponível. Para permitir o condicionamento dos *tiles* na memória, a biblioteca *MXNET* teve parâmetros alterados para não realizar iterações paralelas, e sim sequenciais, garantindo que apenas iniciaria a próxima iteração quando a anterior estivesse finalizada. Caso contrário, a rede tentaria alocar mais *tiles* na memória, extrapolando a quantidade disponível.

Com a matriz de confusão do conjunto *Potsdam256_4* (Tabela A.2) foram calculadas as métricas para cada classe (Tabela A.4). Neste conjunto, os melhores resultados foram relacionados à classe de edificações (*building*) com um *F1-Score* de 91.319%, e o pior resultado relacionado à predição dos fundos das imagens (*background*) de 16.629% de *F1-Score*. Na matriz do conjunto *Potsdam256_1* (Tabela A.3 e Tabela A.5), a mesma tendência de melhores resultados para a classe edificações (*building*) foi observada, com *F1-Score* de 91.697%, e baixo *F1-Score* na segmentação de árvores (*tree*) (de 73.28%), sendo identificada nesta análise a incapacidade do modelo de predizer os fundos das imagens (*background*). Apesar de não significativas, as diferenças nas métricas do primeiro modelo ao ser comparado com o segundo podem ser interpretada como resultado do aumento do número de épocas e do uso de *tiles* com maior contexto. Essas alterações não melhoraram a segmentação, mas garantiram a predição do fundo das imagens (*background*).

Pela quantidade de épocas aplicadas nesse estudo, os dois conjuntos não atingiram a mesma qualidade nos resultados vistos por Diakogiannis et al. (2020). Supõe-se que esse resultado se deve ao baixo número de épocas definido para treinamento desses modelos. Apesar disso, as tendências encontradas — como melhor segmentação de edificações e baixa segmentação em fundos de imagens (*background*) — foram as mesmas do estudo dos autores citados.

Tabela A.2 – Matriz de confusão do conjunto *Potsdam256_4*.

		Predicted						
	Class	***Backg.***	***ImSurf***	***Car***	***Building***	***LowVeg***	***Tree***	***All***
Ground True	***Backg.***	595525	2124793	40917	2916250	617822	119175	6414482
	ImSurf	39133	32869137	137780	790491	2135762	397900	36370203
	Car	14636	259257	2000161	19867	10811	20706	2325438
	Building	18896	992439	14889	31125183	207915	306487	32665809
	LowVeg	60818	1561268	6809	478565	23048515	1564299	26720274
	Tree	19061	1173365	67533	171584	4656170	12099473	18187186
	All	748069	38980259	2268089	35501940	30676995	14508040	122683392

Tabela A.3 – Matriz de confusão do conjunto *Potsdam256_1*.

		Predicted						
		Backg.	***ImSurf***	***Car***	***Building***	**LowVeg**	***Tree***	***All***
Ground True	***Backg.***	0	9824958	115611	8588378	3048460	689517	22266924
	ImSurf	0	117825456	595089	2953219	11387386	2518967	135280117
	Car	0	1041797	7299674	235402	131175	118496	8826544
	Building	0	3374467	28196	111539631	2369586	974377	118286257
	LowVeg	0	3267935	17822	1067505	88011081	4650294	97014637
	Tree	0	2660875	196808	608836	18776405	42775973	65018897
	All	0	137995488	8253200	124992971	123724093	51727624	446693376

Tabela A.4 – Métricas do conjunto *Potsdam256_4* em porcentagem.

	Backg.	***ImSurf***	***Car***	***Building***	***LowVeg***	***Tree***
Precision	79.608	84.323	88.187	87.672	75.133	83.398
Recall	9.284	90.374	86.012	95.284	86.259	66.527
Accuracy	95.133	92.165	99.516	95.177	90.789	93.075
F1-Score	16.629	87.243	87.086	91.319	80.312	74.014
MCC	26.172	81.702	86.847	88.134	74.640	70.673

Tabela A.5 – Métricas do conjunto *Potsdam256_1* em porcentagem.

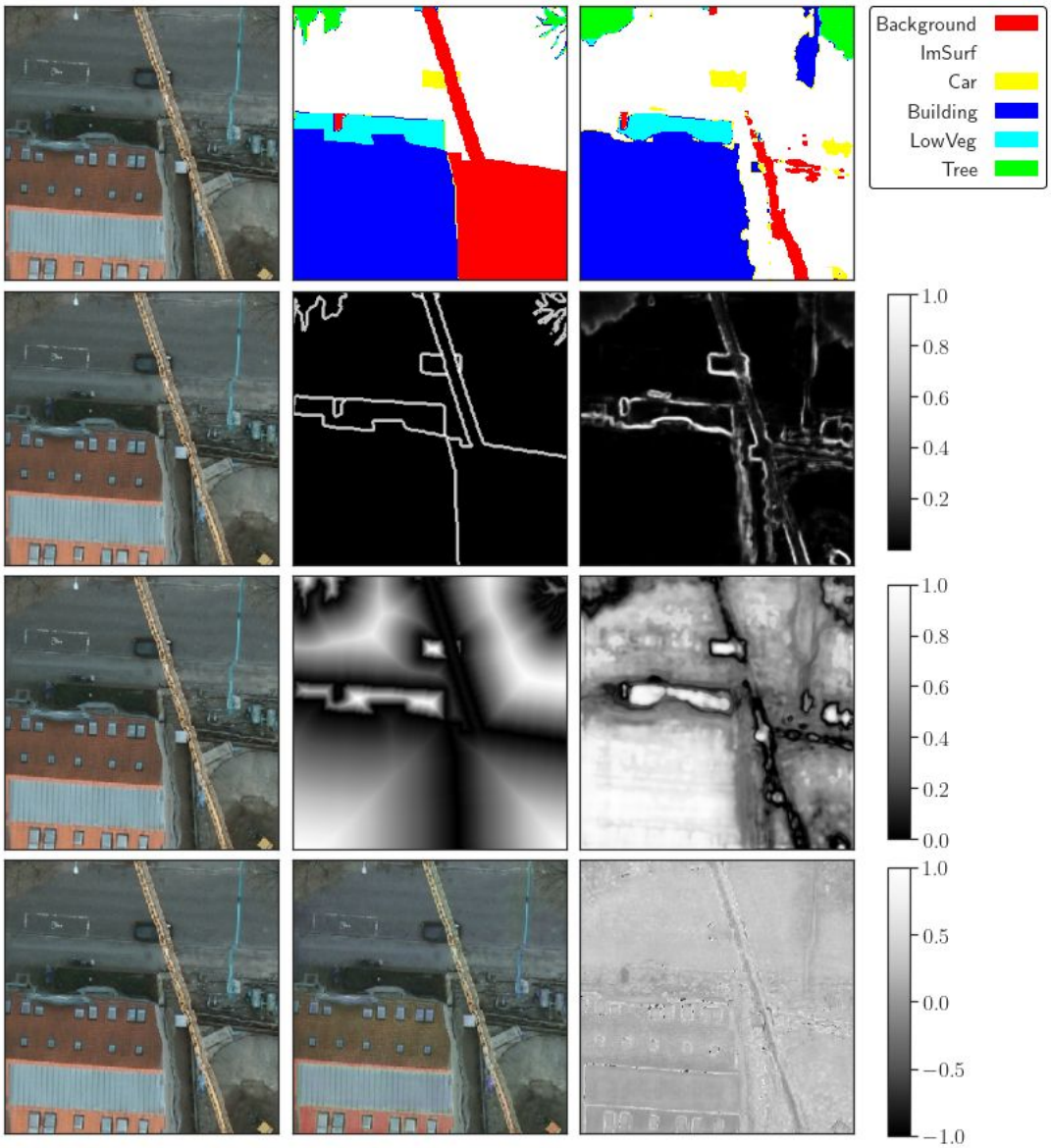
	Backg.	***ImSurf***	***Car***	***Building***	***LowVeg***	***Tree***
Precision	0	85.384	88.447	89.237	71.135	82.695
Recall	0	87.097	82.701	94.296	90.719	65.790
Accuracy	0	91.577	99.445	95.478	89.989	93.017
F1-Score	0	86.232	85.478	91.697	79.742	73.280
MCC	0	80.174	85.245	88.656	74.178	69.923

Pelas Figuras A.1 e A.2 podem ser observados os *tiles* preditos pelos modelos treinados. Nesses *tiles* foi possível identificar a segmentação das classes na imagem, os contornos (ou bordas) dos objetos, o mapa de distâncias inferido e a imagem predita.

Nas Figuras A.1 a A.3 têm-se as predições realizadas. Como essas predições são usadas para a segmentação, consegue-se perceber que as classes *tree* e *lowVeg* possuem um refinamento baixo em bordas e no mapa de distâncias, correspondendo às estatísticas calculadas, devido à

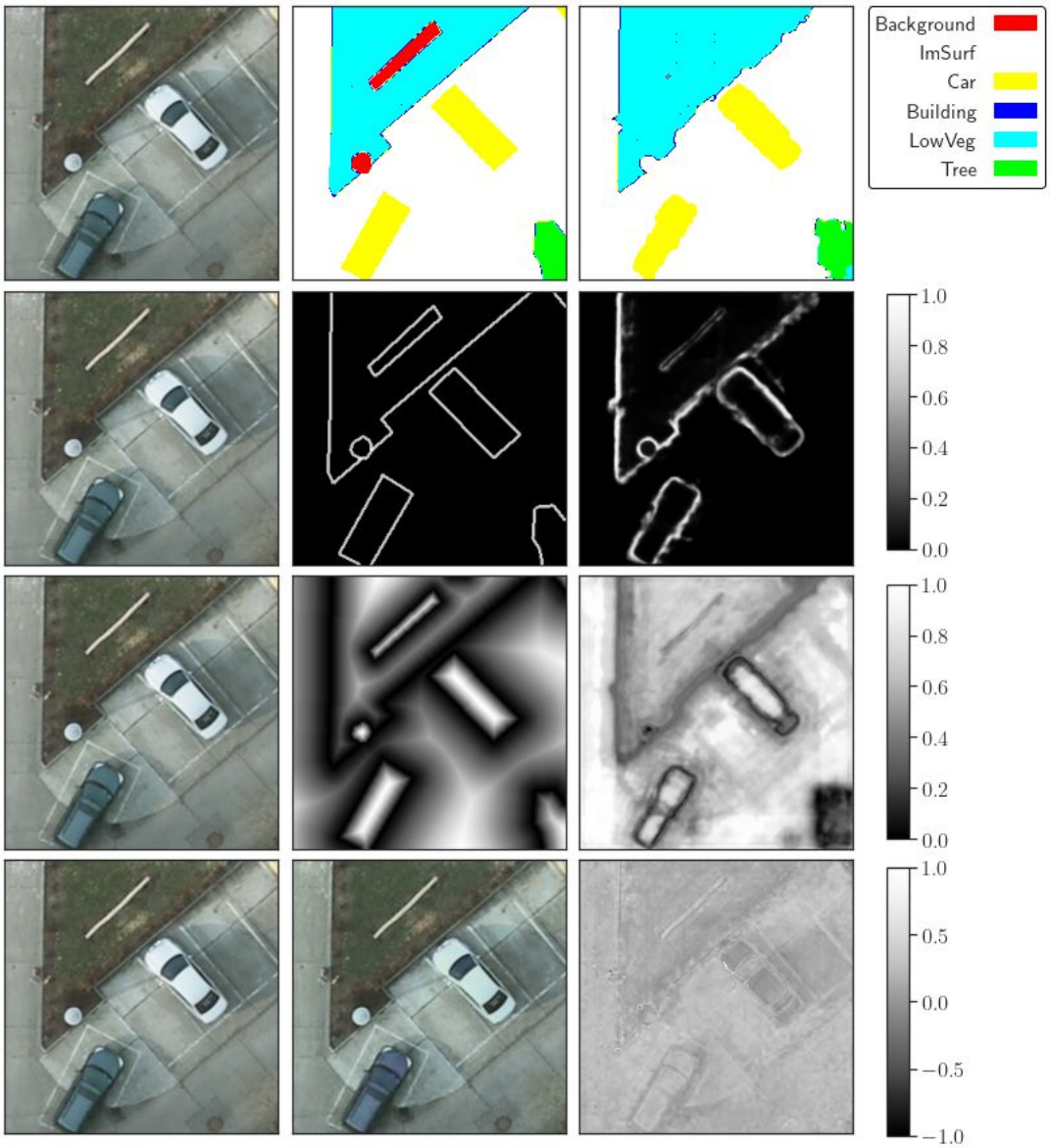
arquitetura priorizar bordas regulares ao segmentar. Segundo Diakogiannis et al. (2020) esse fenômeno é minimizado na segunda versão da arquitetura (*ResUNet-a d7v2*), que não teve seu código disponibilizado pelos autores. Além disso, é possível identificar visualmente que a rede não foi capaz de segmentar a classe *background*.

Figura A.1 – Resultado da predição no modelo *Potsdam256_4*, sendo a primeira linha composta pelo *tile*, rótulo e predição. Na segunda linha é visto o *tile*, as bordas e a predição das bordas. Na terceira linha são apresentadas as distâncias das bordas, e a inferência das distâncias das bordas; e na quarta linha são mostrados o primeiro *tile*, o *tile* reconstruído, e o *tile* em HSI.



O terceiro modelo de parâmetros, por usar o dobro de épocas dos modelos anteriores, teve predição aproximada às encontradas por Diakogiannis et al. (2020), com *F1-Score* de 96% para edificações. Os piores resultados são relacionados à classe *background*, com *F1-Score* de 70,6%, seguindo a mesma tendência já citada dos modelos anteriores. A matriz de confusão pode ser observada na Tabela A.6 e as métricas apresentadas na Tabela A.7. Em uma análise

Figura A.2 – Resultado da predição no modelo *Potsdam256_1*, sendo a primeira linha composta pelo *tile*, rótulo e predição. Na segunda linha é visto o *tile*, as bordas e a predição das bordas. Na terceira linha são apresentadas as distâncias das bordas, e a inferência das distâncias das bordas; e na quarta linha são mostrados o primeiro *tile*, o *tile* reconstruído, e o *tile* em HSI.



visual pela Figura A.3, é possível perceber maior similaridade entre a predição dos contornos e do mapa de distâncias do que a vista nos modelos anteriores, com maiores dificuldades em árvores (*tree*) e vegetação baixa (*lowveg*).

Figura A.3 – Resultado do modelo *Potsdam128_4*, sendo a primeira linha composta pelo *tile*, rótulo e predição. Na segunda linha é visto o *tile*, as bordas e a predição das bordas. Na terceira são apresentadas as distâncias das bordas, e a inferência das distâncias das bordas; na quarta linha são mostrados o primeiro *tile*, o *tile* reconstruído, e o *tile* em HSI.

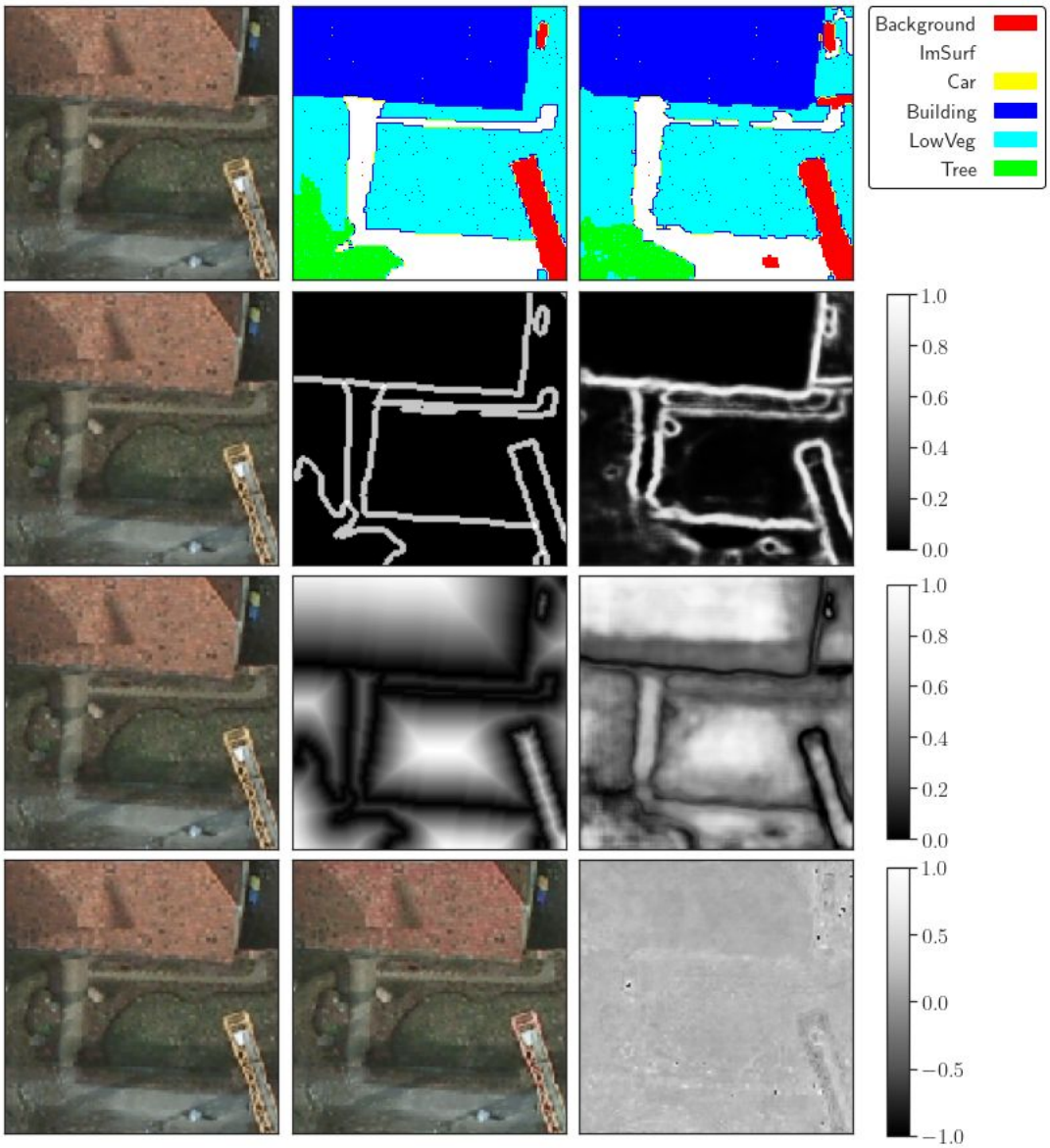


Tabela A.6 – Matriz de confusão do conjunto *Potsdam128_4*.

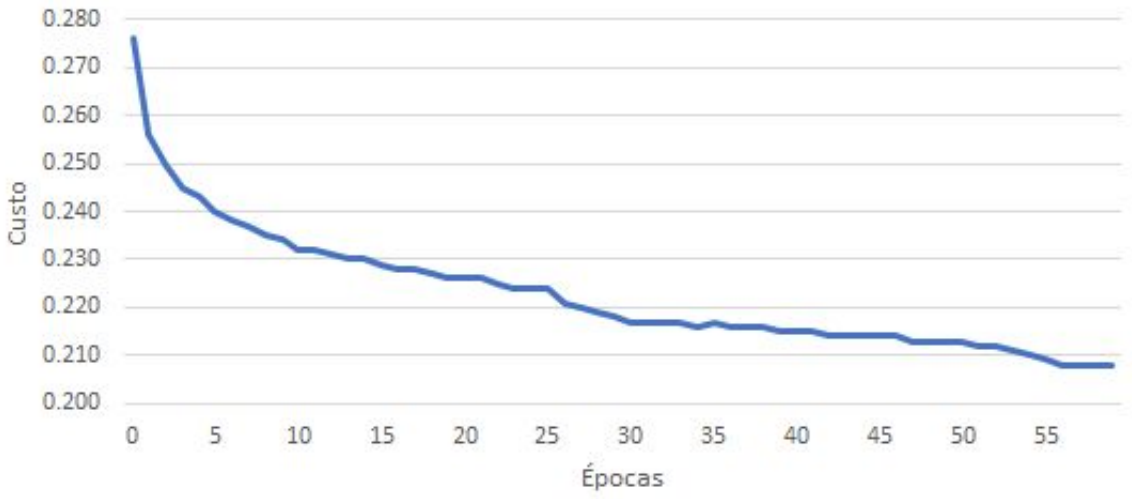
					Predicted			
		Backg.	***ImSurf***	***Car***	***Building***	***LowVeg***	***Tree***	***All***
Ground True	**Backg.**	3281029	1087918	32205	494586	492615	178876	5567229
	ImSurf	295738	30836487	165182	510440	1253393	759515	33820755
	Car	14172	131330	1976571	28780	5692	51125	2207670
	Building	49906	457681	2235	28721372	212282	125717	29569193
	LowVeg	51060	1438902	6602	315287	20615684	1826304	24253839
	Tree	30368	604666	49202	147508	2222723	13200191	16254658
	All	3722273	34556984	2231997	30217973	24802389	16141728	111673344

Tabela A.7 – Métricas do conjunto *Potsdam128_4* em porcentagem.

	Backg.	***ImSurf***	***Car***	***Building***	***LowVeg***	***Tree***
Precision	88.146	89.234	88.556	95.047	83.120	81.777
Recall	58.935	91.176	89.532	97.133	85.000	81.209
Accuracy	97.558	93.996	99.564	97.901	92.993	94.631
F1-Score	70.640	90.195	89.041	96.079	84.049	81.492
MCC	70.953	85.879	88.821	94.656	79.569	78.352

Com isso, conclui-se que o modelo *Potsdam128_4* garantiu melhores resultados ao ser feito com uma quantidade de épocas maior que os modelos anteriores, além de ter uma velocidade superior pela degradação das imagens e redução do tamanho dos *tiles*. Sem essas modificações, o modelo *Potsdam256_1* necessitaria de algo em torno de quatro vezes mais tempo de processamento entre as épocas do que foi observado no *Potsdam128_4*. Pelo gráfico da função custo (Figura A.4) verifica-se que a tendência de evolução do modelo ainda não havia sido atingida com 60 épocas, o que indica que mais épocas seriam necessárias até sua estabilização.

Figura A.4 – Evolução da função custo *Tanimoto with dual* pelo número de épocas.



Conclui-se que diversos conceitos relacionados à segmentação de edificações por *deep learning* foram vistos com a execução desses experimentos preliminares, e devido à complexidade de módulos e processos, a execução do experimento com a rede *ResUNet-a* permitiu identificar as dificuldades nos processos recorrentes em outras redes, e oferecerá embasamento para as próximas análises que serão realizadas.

O desempenho computacional demonstrou que, apesar da dimensão reduzida dos *tiles* usados no estudo, as camadas de aprofundamento envolvidas em uma rede de *deep learning* resultam em um alto custo de processamento e consequente alto uso de *hardware*. Para o *hardware* utilizado, houve a necessidade de alteração do funcionamento da biblioteca *MXNET* para realizar operações sequenciais, e não paralelas, evitando exceder a memória disponível. Desse modo, as tarefas executadas foram, consequentemente, mais lentas.

O ajuste no tamanho do lote e dos *tiles* impactaram no desempenho computacional e, consequentemente, no desempenho da segmentação semântica. Por permitir maior contexto para a rede, o modelo com mais épocas, menor dimensão de *tiles* e maior *batch* resultou na melhor segmentação dentre as configurações analisadas. Dessa forma, as deficiências relacionadas ao *hardware* disponível são parcialmente superadas, ainda persistindo o aspecto crítico do tempo de execução dos processos. Cabe destacar que, essas análises somente foram possíveis devido à construção de três modelos de parâmetros baseados em três conjuntos de dados distintos executados no experimento preliminar. Nas análises dos resultados da segmentação semântica de edificações, verificou-se que o terceiro modelo resultou nas melhores métricas, com *F1-Score* de 96% na segmentação de edificações, com 60 épocas de treinamento e tempo de execução de 4 dias. Portanto, apesar do maior tempo em treinamento, a rede atingiu resultados satisfatórios para uma segmentação semântica de edificações, que este o objeto de interesse. Também tem-se que a evolução do treinamento, analisando a função custo, demostra que a rede não havia alcançado sua estabilização, e necessitaria mais épocas para alcançar o máximo do aprendizado. Dessa forma, aumentar o número de épocas traria maior tempo de operação, mas garantiria a estabilização da função custo e do aprendizado em seu maior nível.

A metodologia de treinamento e avaliação dos resultados aplicados nestes experimentos correspondeu ao que é comumente visto em outros estudos, permitindo a comparação dos resultados obtidos. Com isso, o estudo poderia ser reproduzido com outros *datasets*, no mesmo ambiente *python* e com as bibliotecas requeridas para a realização dos processamentos realizados, além de permitir a inserção de diferentes composições de imagens e MDSn. Deste modo, este estudo preliminar trouxe embasamento para a condução da metodologia proposta e adotada neste trabalho.

APÊNDICE B – ALGORITMOS/CÓDIGOS

Neste anexo são apresentados os algoritmos desenvolvidos e utilizados neste estudo, na linguagem Python. As bibliotecas necessárias para a execução destes algoritmos estão declaradas no início de cada conjunto de códigos.

```
import cv2
import numpy as np
import math
import tkinter as tk
from tkinter import filedialog
import os
from tifffile import imsave, imread

root = tk.Tk()
root.withdraw()
img_path = filedialog.askopenfilenames(initialdir = "")
dsm_path = filedialog.askopenfilenames(initialdir = "")

def RGB_TO_HSI(img,dsm):
    #Conversao para o espaco 0-1
    bgr = img.astype(np.float32)/255

    #Importacao do MDSn
    ndsm = dsm.astype(np.float32)

    #Divisao da imagem
    blue = bgr[:,:,0]
    green = bgr[:,:,1]
    red = bgr[:,:,2]
    dsm_n = ndsm[:,:,0]

    #Declaracao da dimensao das imagens (usando a dimensao do canal red)
    hue = np.copy(red)
    n = np.copy(red)
    d = np.copy(red)

    #Calculo da Intensidade
    intensity = (red + blue + green)/3

    #Calculo do Hue
    for i in range(0, blue.shape[0]):
        for j in range(0, blue.shape[1]):
            d[i][j] = (math.sqrt((red[i][j] -  green[i][j])**2 + ((red[i][j
    ] - blue[i][j])*(green[i][j] - blue[i][j]))))
```

```
            if d[i][j] == 0:
                hue[i][j] = 0
            else:
                n[i][j] = ((1/2)*((red[i][j] - green[i][j]) + (red[i][j] -
    blue[i][j])))
                hue[i][j] = (180/math.pi)*math.acos(n[i][j]/d[i][j])
                if blue[i][j] <= green[i][j]:
                    hue[i][j] = hue[i][j]/360
                else:
                    hue[i][j] = (360 - hue[i][j])/360

    #Conversao para o espaco 0-255
    hue255 = hue*255
    intensity255 = intensity*255

    #Uniao dos canais na ordem hue, intensidade e MDSn
    hsi = cv2.merge([hue255, intensity255, dsm_n])

    return hsi

#Importacao, leitura e gravacao das imagens
for file_img,file_dsm in zip(img_path,dsm_path):
    print(file_img)
    print(file_dsm)
    img = imread(file_img)
    dsm = cv2.imread(file_dsm)
    hsi = RGB_TO_HSI(img,dsm)
    dirname = '../2_Ortho_RGB'
    name = file_img
    hsi = hsi.astype(np.uint8)
    imsave(os.path.join(dirname, name), hsi)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Código B.1 – Código destinado à preparação do *dataset HInDSM*.

```
import cv2
import tkinter as tk
from tkinter import filedialog
import os
import numpy as np
from tifffile import imread

root = tk.Tk()
root.withdraw()
img_path = filedialog.askopenfilenames(initialdir = "")
print(len(img_path))
```

```
12
13 hc, ic, dc = 0,0,0
14 for file in img_path:
15     print(file)
16     img = imread(file)
17     hidsm = img.astype(np.float32)
18
19     hue = hidsm[:,:,0]
20     intensity = hidsm[:,:,1]
21     dsm_n = hidsm[:,:,2]
22
23     if file == img_path[0]:
24         h_total = np.copy(hue)
25         i_total = np.copy(intensity)
26         d_total = np.copy(dsm_n)
27     else:
28         h_total = np.concatenate((h_total,hue), axis=0)
29         i_total = np.concatenate((i_total,intensity), axis=0)
30         d_total = np.concatenate((d_total,dsm_n), axis=0)
31
32 hc_ac = np.mean(h_total)
33 ic_ac = np.mean(i_total)
34 dc_ac = np.mean(d_total)
35 print("Media total hue %.15f, Media total intensidade %.15f, Media total
       dsm %.15f" %(hc_ac,ic_ac,dc_ac))
36
37 h_std = np.std(h_total)
38 i_std = np.std(i_total)
39 d_std = np.std(d_total)
40 print("Desvio total hue %.15f, Desvio total intensidade %.15f, Desvio total
        dsm %.15f" %(h_std, i_std, d_std))
```

Código B.2 – Cálculo dos parâmetros de normalização.

```
1 for p in net.collect_params().values():
2     p.grad_req = 'add'
```

Código B.3 – Aplicação da função de agregação de gradientes.

```
1 for p in net.collect_params().values():
2     p.zero_grad()
```

Código B.4 – Aplicação da função visando inicializar a lista de gradientes com o valor nulo.

```
1 for epoch in range(16): #Numero de epocas
2     train_loss = 0. #Funcao custo definida em 0
3
4     for imgs, masks in datagen: #Datagen possui 265 tiles de imagens e
      mascaras
```

```
5
6         #Dividindo o datagen em 44 partes
7         data_split = np.array_split(imgs,44)
8         label_split = np.array_split(masks,44)
9
10            for j in range(len(data_split)):
11            #Carregamento de cada parte do datagen dividido
12                img  = nd.array(data_split[j],ctx)
13                mask = nd.array(label_split[j],ctx)
14
15                with autograd.record():
16                    ListOfPredictions = net(img) #treinamento dos dados
17                    loss = Loss(ListOfPredictions,mask) #calculo da funcao
    custo
18                loss.backward() #computa o custo
19
20            trainer.step(batch_size) #avanca para a proxima parte do
    treinamento
21
22            #zerar a lista de gradientes
23            for p in net.collect_params().values():
24                p.zero_grad()
```

Código B.5 – Código utilizado para a execução da agregação manual de gradientes.