Universidade Estadual de Londrina

MAURICIO KENDI YUI

AVALIAÇÃO DE APRENDIZAGEM LBP PARA SEGMENTAÇÃO SEMÂNTICA DE PAVIMENTAÇÃO ASFÁLTICA BASEADA EM U-NET

LONDRINA
2024

MAURICIO KENDI YUI

AVALIAÇÃO DE APRENDIZAGEM LBP PARA SEGMENTAÇÃO SEMÂNTICA DE PAVIMENTAÇÃO ASFÁLTICA BASEADA EM U-NET

Dissertação apresentada ao Programa de Mestrado em Engenharia Elétrica da Universidade Estadual de Londrina para obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Prof. Dr. Leonimer Flávio de Melo

LONDRINA
2024

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

Y94a Yui, Mauricio Kendi.
AVALIAÇÃO DE APRENDIZAGEM LBP PARA SEGMENTAÇÃO SEMÂNTICA DE PAVIMENTAÇÃO ASFÁLTICA BASEADA EM U-NET / Mauricio Kendi Yui. - Londrina, 2024.
131 f. : il.

Orientador: Leonimer Flávio de Melo.
Dissertação (Mestrado em Engenharia Elétrica) - Universidade Estadual de Londrina, Centro de Tecnologia e Urbanismo, Programa de Pós-Graduação em Engenharia Elétrica, 2024.
Inclui bibliografia.

1. Aprendizagem profunda - Tese. 2. Visão computacional - Tese. 3. Direção autônoma - Tese. 4. Local binary pattern - Tese. I. Melo, Leonimer Flávio de. II. Universidade Estadual de Londrina. Centro de Tecnologia e Urbanismo. Programa de Pós-Graduação em Engenharia Elétrica. III. Título.

CDU 62

MAURICIO KENDI YUI

AVALIAÇÃO DE APRENDIZAGEM LBP PARA SEGMENTAÇÃO SEMÂNTICA DE PAVIMENTAÇÃO ASFÁLTICA BASEADA EM U-NET

Dissertação apresentada ao Programa de Mestrado em Engenharia Elétrica da Universidade Estadual de Londrina para obtenção do título de Mestre em Engenharia Elétrica.

BANCA EXAMINADORA

Orientador: Prof. Dr. Leonimer Flávio de Melo
Universidade Estadual de Londrina – UEL

Prof. Dra. Maria Bernadete de Morais França
Universidade Estadual de Londrina – UEL

Prof. Dr. Daniel Strufali Batista
Universidade Estadual de Londrina – UEL

Prof. Dr. Wesley Angelino de Souza
Universidade Tecnológica Federal do Paraná – UTFPR

Londrina, 30 de julho de 2024.

A Deus.

AGRADECIMENTOS

Ao meu orientador Prof. Dr. Leonimer Flávio de Melo.

À todo corpo docente do Departamento de Engenharia Elétrica da UEL.

À empresa Orion Gaming.

Aos meus pais.

YUI, M. K.. **Avaliação de Aprendizagem LBP para Segmentação Semântica de Pavimentação Asfáltica Baseada em U-Net**. 2024. 143f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Estadual de Londrina – UEL, Londrina, 2024.

RESUMO

A direção autônoma emerge naturalmente do notável avanço demonstrado pela Inteligência Artificial (IA) nos últimos anos, impulsionada, em grande parte, pelo desenvolvimento contínuo de recursos de *hardware* e *software*. Nesse contexto, este trabalho propõe a aplicação de ferramentas de IA, mais especificamente no domínio da Aprendizagem Profunda, para realizar a segmentação semântica de pavimentações asfálticas em imagens rasterizadas. A abordagem adotada utiliza a rede neural *U-Net* para alcançar esse objetivo, enquanto o algoritmo *Local Binary Pattern* (LBP) é investigado devido à sua vantagem declarada de invariância luminosa. Este trabalho contribui ao validar que a integração do algoritmo LBP pode aprimorar o desempenho da rede neural *U-Net*, especialmente em condições não previstas na base de dados de treinamento original.

Palavras-chave: inteligência artificial, aprendizagem de máquina, aprendizagem profunda, redes neurais artificiais, visão computacional, direção autônoma, *Local Binary Pattern* (LBP).

YUI, M. K.. **Evaluation of LPB Learning for U-Net Based Road Semantic Segmentation**. 2024. 143p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina, 2024.

ABSTRACT

Autonomous driving emerges naturally from the remarkable advancement demonstrated by Artificial Intelligence (AI) in recent years, driven, in large part, by the continuous development of hardware and software resources. In this context, this works proposes the application of AI tools, more specifically in the domain of Deep Learning, to perform the semantic segmentation of asphalt paving in raster images. The adopted approach uses the U-Net neural network to achieve this objective, while the Local Binary Pattern (LBP) algorithm is investigated due to its declared advantage of light invariance. This work contributes by validating that the integration of the LBP algorithm can improve the performance of the U-Net neural network, especially in conditions not foreseen in the original training database.

Keywords: artificial intelligence, machine learning, deep learning, artificial neural networks, computer vision, autonomous driving, *Local Binary Pattern* (LBP)

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

ACDC	*Adverse Conditions Dataset with Correspondences*
ADAM	*Adaptive Moment Estimation*
ANN	*Artificial Neural Network*
CNN	*Convolutional Neural Network*
DLERS	*Deep LBP-Enriched Real-time Segmentation*
FCN	*Fully Convolutional Network*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
LBP	*Local Binary Pattern*
MSE	*Mean Squared Error*
PSPNet	*Pyramid Scene Parsing Network*
ReLU	*Rectified Linear Unit*
ResUNet	*Residual U-Net*
RNA	Redes Neurais Artificiais
SSP	*Spatial Pyramid Polling*
Tanh	Tangente Hiperbólica
VGG	*Visual Geometry Group*

SUMÁRIO

1 INTRODUÇÃO

É amplamente reconhecido que o marco inicial no campo da Inteligência Artificial (IA) remonta a 1943, com a publicação do primeiro artigo, em (MCCULLOCH; PITTS, 1943), descrevendo o modelo matemático de um rede neural. Esse modelo ofereceu um método para representar as funcionalidades cerebrais de maneira abstrata, demonstrando que a unidade fundamental, o neurônio, poderia ser interligada em uma rede para fornecer uma capacidade computacional significativa.

No entanto, apenas nos últimos anos foi observado um desenvolvimento acelerado da IA. Isso foi possível graças aos avanços tecnológicos tanto na área de *hardware*, com processadores de velocidades de *clock* cada vez mais altas, processamento distribuído em núcleos paralelos, maior disponibilização de memória e bancos de dados imensos, quanto na área de *software*, com sistemas operacionais, linguagens de programação e ferramentas que facilitam o desenvolvimento, depuração e teste de códigos (RUSSELL; NORVIG, 2010).

Em Russell e Norvig (2010), define-se que a IA, por meio de suas Redes Neurais Artificiais (RNAs), busca modelar o pensamento humano para que um programa possa gerar uma saída racional. Isto é, gerar a melhor saída possível ou, quando incertezas são introduzidas por variações no ambiente de operação, ajustar-se para obter o melhor resultado esperado. Essa capacidade de adaptação possibilita que diversas tarefas realizadas por pessoas sejam automatizadas, delegando a tomada de decisão a uma RNA. Dentre essas aplicações, a direção autônoma de veículos tem ganhado destaque e relevância significativos.

Dirigir é uma tarefa que deve levar em consideração uma quantidade muito grande de informações antes que uma decisão segura possa ser tomada. Conforme apresentado em Aoto, Wada e Numata (2018), para que um veículo seja considerado autônomo, inúmeros obstáculos precisam ser superados: detecção da pavimentação asfáltica, identificação e interpretação de semáforos e sinais de trânsito, localização de obstáculos e pessoas, planejamento de rotas e posicionamento. Cada um desses desafios apresenta complexidade suficiente para justificar um estudo independente. Este trabalho, portanto, concentra-se no problema específico da detecção de pavimentações asfálticas.

A identificação de pavimentações asfálticas pode ser realizada através do processamento de imagens rasterizadas. Câmeras digitais convencionais são comuns e de fácil acesso, e podem ser usadas para fazer a captura dessas imagens. Posteriormente, as imagens são submetidas a um método que deve delimitar, isolar e classificar os objetos na imagem. Isso pode ser alcançado por meio da segmentação semântica, uma técnica que

gera uma máscara de segmentação. Essa máscara atribui uma classe ou rótulo semântico a cada *pixel* pertencente a uma região segmentada (GREESHMA, 2020). De forma inerente a este trabalho, a segmentação semântica de pavimentações asfálticas consiste na identificação dos *pixels* na imagem que correspondem a essa classe específica. De forma geral, a segmentação semântica assume um papel fundamental em aplicações que demandam uma compreensão sofisticada e contextualizada do conteúdo visual.

Diante do exposto, este trabalho propõe a utilização de uma *U-Net*, um modelo específico de RNA, para realizar a segmentação semântica de pavimentações asfálticas em imagens rasterizadas. A contribuição deste trabalho reside, no entanto, na forma diferenciada de treinamento da *U-Net*.

Conforme mencionado em Goodfellow, Bengio e Courville (2016), as RNAs buscam por padrões nos dados recebidos para gerar a saída. Atentando-se a essa característica, observou-se que o algoritmo *Local Binary Pattern* (LBP) (OJALA; PIETIKäINEN; HARWOOD, 1996), gera mapas de texturas que ressaltam os padrões estruturais em imagens rasterizadas, atenuando, ao mesmo tempo, diferenças nos níveis de iluminação. A aplicação do algoritmo LBP no método de treinamento contribui para a melhora da precisão da *U-Net* em cenários nos quais o modelo não foi treinado para operar.

1.1 Escopo e Justificativa

O neurônio artificial é o elemento fundamental da Aprendizagem Profunda, uma especialização da Aprendizagem de Máquina, que, por sua vez, é uma sub-área da Inteligência Artificial (IA). A organização de vários neurônios em camadas e sua interligação formam uma rede neural artificial (RNA), conferindo a essa estrutura a capacidade de análise abstrata e generalista.

Reconhecendo a versatilidade das RNAs, este trabalho propõe um método para a segmentação semântica de pavimentações asfálticas por meio da arquitetura *U-Net*, um modelo específico de RNA. A *U-Net* tem atraído considerável atenção na segmentação semântica de objetos em imagens desde a sua primeira publicação em Ronneberger, P.Fischer e Brox (2015). Além do bom desempenho, a base bibliográfica é ampla e existem bibliotecas de *software* de código aberto para facilitar a implementação. O trabalho contribui para o aprimoramento da precisão da rede ao integrar as vantagens do descritor de texturas *Local Binary Pattern* (LBP) à base de dados de treinamento.

Ao aplicar o descritor de texturas LBP a uma imagem, o resultado gerado é conhecido como mapa de texturas LBP, comum em tarefas de classificação de texturas através de histogramas. No entanto, o mapa de texturas LBP destaca características importantes das imagens (ZHANG et al., 2017) (WU; ZHOU; LI, 2022), além de atenuar os efeitos de iluminação, conforme descrito na publicação original do algoritmo LBP em Ojala, Pie-

tikäinen e Harwood (1996), em que se declara que as características extraídas demonstram invariância à translação de níveis de cinza. A evidenciação de padrões estruturais contidos na imagem auxilia no processo de treinamento e contribui para melhorar a resposta da rede, especialmente em situações de baixa luminosidade que podem não estar incluídas no conjunto de dados de treinamento. Consequentemente, a confiabilidade é aumentada, beneficiando sistemas como os de navegação autônoma que estão expostos a condições climáticas adversas.

A compreensão do ambiente é um requisito fundamental para que um veículo alcance a autonomia de direção. Essa capacidade cognitiva pode ser atingida pela segmentação semântica de imagens rasterizadas, obtidas por câmeras digitais convencionais.

Novamente, devido à enorme variedade de cenários possíveis, a segmentação semântica apresenta-se como mais uma tarefa que pode ser melhor executada se for confiada a uma rede neural. Certos modelos de redes neurais são capazes de realizar a segmentação semântica necessária para essa aplicação. Geralmente, durante a fase de filtragem convolucional e agrupamento na primeira metade da rede, as características-chave dos objetos, como formato, cores, texturas, e outros padrões, são identificadas e extraídas. Posteriormente, no processo de convolução transposta e na aplicação de novas convoluções na metade de saída da rede, uma máscara é gerada, contendo a classificação semântica de cada *pixel* na imagem original. Diferente das convoluções de entrada cujo papel é filtrar as características marcantes, as convoluções na saida contribuem para a construção da imagem final, associando os padrões encontrados às classes respectivas (RONNEBERGER; P.FISCHER; BROX, 2015).

Neste momento, é importante lembrar da propriedade de flexibilidade das redes neurais (RUSSELL; NORVIG, 2010). Conforme discutido, um mesmo modelo oferece a possibilidade de impactar em diversos outros setores econômicos, como monitoramento e segurança, diagnósticos por imagens médicas, controle de qualidade, entre outros. A distinção entre esses setores reside na base de dados utilizada durante o treinamento do modelo. Contudo, a compilação dessa base de dados representa um dos maiores desafios, demandando um investimento significativo na coleta abrangente de imagens de treinamento e na implementação de suas máscaras.

No caso específico de aplicações em veículos autônomos, algumas bases de dados foram estabelecidas e reconhecidas a ponto de criar diretrizes e referências, para o formato, a resolução das imagens e a implementação das máscaras. Essas regras podem estar sendo adotadas à medida que novas bases de dados são disponibilizadas. Essa prática não só reduz custos e acelera o desenvolvimento de novas pesquisas e projetos, como também facilita a reprodutibilidade para interesses científicos.

1.2 Objetivos

O objetivo principal é realizar a segmentação semântica de pavimentações asfálticas em imagens rasterizadas por meio de uma rede neural *U-Net*, aproveitando as vantagens do descritor de texturas LBP para melhorar a resposta final do modelo.

São propostas de objetivos secundários:

- Detalhar a estrutura da rede *U-Net*;
  - Analisar os componentes de forma isolada, com a descrição de sua funcionalidade realizada tanto de maneira individual quanto contextualizada;
- Apresentar o algoritmo *Local Binary Pattern* (LBP):
  - Examinar as caracteristicas do mapa de texturas LBP extraído de uma imagem rasterizada;
  - Destacar as vantagens do mapa de texturas LBP no treinamento de RNAs;
- Explorar as ferramentas relacionadas à manipulação de arquivos de imagens em *Python*:
  - Manipular operações simples para abrir, salvar, fechar e redimensionar;
  - Extrair o mapa de texturas LBP;
- Explorar as ferramentas e técnicas relacionadas à pesquisa em IA em *Python*;
  - Implementar em código a estrutura de uma *U-Net*;
  - Explorar bases de dados disponíveis na internet para treinamento de RNAs;
  - Validar o funcionamento de *hardwares* dedicados para aplicações com RNAs;
- Realizar ensaios para validar a declaração de invariância à escala de cinza do descritor de texturas LBP:
  - Validar a melhora de performance do modelo em condições de iluminação para os quais não foi treinado;
  - Comparar a performance do modelo com o treinamento convencional;

1.3 Descrição do Projeto

No contexto de direção autônoma, a segmentação semântica proporciona uma compreensão mais profunda do conteúdo da imagem em comparação com os métodos de segmentação tradicionais. Diversas arquiteturas foram publicadas para lidar com a tarefa de

segmentação semântica, tais como *Fully Convolutional Networks* (FNC) (LONG; SHELHAMER; DARRELL, 2015), *SegNet* (BADRINARAYANAN; KENDALL; CIPOLLA, 2016), *Pyramid Scene Parsing Network* (*PSPNet*) (ZHAO et al., 2017), entre outras. O modelo *U-Net* (RONNEBERGER; P.FISCHER; BROX, 2015) foi escolhido devido ao seu equilíbrio entre boa performance e relativa simplicidade em comparação com a maioria dos modelos mencionados, além da disponibilidade de uma base bibliográfica abrangente e bibliotecas de *software* de código aberto com suporte sólido para implementação.

Explorando o tópico relacionado a *softwares*, a biblioteca *TensorFlow* foi desenvolvida para pesquisa e desenvolvimento de redes neurais artificiais. Com seu código aberto e escrito em C++, é acessível para estudos acadêmicos, permitindo também sua aplicação comercial. A interface *Keras* foi integrada ao *TensorFlow*, facilitando seu uso com alto nível de abstração na programação em linguagem *Python*.

A plataforma *Colab* é um serviço de nuvem hospedado pela empresa *Alphabet*. A interface gráfica é exibida em formato de *notebooks* e constitui um ambiente de desenvolvimento com programação em linguagem *Python*, aplicável a diversos propósitos, como educação, Aprendizagem de Máquina e análise de dados. O uso é gratuito, no entanto, mediante algum custo, pode-se acessar mais e melhores recursos de *hardware*. Especificamente, as placas de GPUs de alta tecnologia trazem benefícios justificados pela otimização no tratamento de redes neurais artificiais, sem a necessidade de configuração. Além disso, a quantidade significativa de memória RAM dedicada tanto ao sistema operacional quanto à placa GPU é imprescindível para o processamento de bases de dados que envolvem imagens.

O estudo de redes neurais também requer uma base de dados para treinamento e testes. Embora seja possível construí-la, isso demandaria uma considerável quantidade de energia e recursos. Existem, no entanto, algumas bases de dados estabelecidas, como *Cityscapes* (CORDTS et al., 2016) e *Adverse Conditions Dataset with Correspondences* (ACDC) (SAKARIDIS; DAI; GOOL, 2021). *Cityscapes* é uma extensa base de dados com imagens em alta resolução capturadas das ruas de cidades europeias durante o dia. Já o ACDC oferece fotos com padrões e resoluções muito semelhantes às do *Cityscapes*, mas retrata cenários em condições ambientais adversas variadas.

Um diferencial estratégico trazido para este trabalho envolve o uso do algoritmo LBP, baseado em conhecimento prévio de publicações que relatam um aumento nos índices de resposta ao aplicar esse operador no treinamento de redes neurais. Esse aumento pode ser atribuído, provavelmente, à capacidade de o operador LBP gerar um mapa de texturas da imagem original, que enfatiza o contraste relevante entre os *pixels*. Esse enfoque destaca as bordas e os diferentes padrões de texturas dos objetos representados na imagem, ao mesmo tempo em que minimiza a influência da iluminação e superfícies lisas.

A biblioteca *scikit-image* (PEDREGOSA et al., 2011) é de código aberto, pode ser

acessada em linguagem *Python*, e oferece gratuitamente uma variedade de funções para o processamento de imagens. Entre as funções está incluída uma para o processamento do algoritmo LBP em imagens rasterizadas.

Com a disponibilidade de um ambiente para a implementação da *U-Net*, as bases de dados contendo imagens e máscaras necessárias para treinamento e teste, juntamente com uma função para derivar os mapas de textura LBP, torna-se possível elaborar um esquema para gerar as máscaras de segmentação semântica. Em seguida, deve-se validar o aumento na precisão do treinamento utilizando os mapas de textura LBP. Para isso, são elaborados três ensaios, em que todos utilizam a mesma arquitetura *U-Net* variando apenas o método de treinamento.

No primeiro ensaio, uma *U-Net* é treinada de duas maneiras distintas: uma utilizando imagens rasterizadas do período diurno e outra utilizando os respectivos mapas de textura LBP das imagens diurnas. Este ensaio avalia os treinamentos de duas maneiras: o primeiro teste é conduzido exclusivamente com imagens diurnas, com o objetivo de comparar equitativamente a precisão dos dois treinamentos para os cenários nos quais o modelo foi treinado. O segundo teste é realizado apenas com imagens noturnas, visando verificar a degradação da precisão dos dois treinamentos. A finalidade é comprovar que os mapas de textura LBP facilitam na identificação dos padrões e texturas pertencentes à pavimentação asfáltica mesmo em cenários não incluídos no processo de treinamento.

O segundo ensaio tem caráter de controle experimental. A *U-Net* é novamente treinada de duas formas distintas, seguindo o mesmo procedimento do primeiro experimento, mas incluindo as imagens noturnas no processo de treinamento. O propósito é validar se a precisão do modelo é aprimorada nos testes utilizando imagens noturnas para ambos os treinamentos.

O terceiro e último ensaio concatena o mapa de texturas LBP na imagem rasterizada, resultando em uma imagem com quatro camadas. São realizados dois treinamentos: o primeiro apenas com imagens de dia, e o segundo com imagens de dia e de noite. O objetivo é verificar a degradação da precisão nos testes com imagens noturnas, e também verificar se existe um aumento de precisão ao utilizar essa técnica, em relação aos experimentos anteriores.

Enfim, o plano da metodologia inclui a revisão detalhada da arquitetura *U-Net*, a integração eficiente de *softwares* e *hardwares*, a seleção da bases de dados, e a avaliação da viabilidade ao agregar um algoritmo para tratamento de imagem. Esses elementos são combinados para a implementação da U-Net. O modelo é posto à prova, sendo treinado de diferentes formas para confirmar a vantagem do método de treinamento proposto.

1.4 Descrição do Trabalho

O Capítulo 2 aborda a fundamentação teórica do projeto. Inicia-se com a apresentação dos primeiros modelos de neurônios artificiais, explorando o seu funcionamento interno, desde a soma ponderada das entradas na primeira etapa até as diversas funções de ativação na saída. Detalhes sobre o processo de treinamento por retropropagação são discutidos, sendo relevante destacar o processo automatizado de otimização dos pesos associados às entradas dos neurônios. Em seguida, é demonstrado como vários neurônios podem ser conectados e utilizados na forma de módulos para executar algum processamento, e como esses módulos são interligados para formar a rede.

Adicionalmente, a teoria relacionada ao descritor de texturas LBP é abordada, com a apresentação de suas características e vantagens de interesse para o projeto.

O Capítulo 3 examina alguns trabalhos que incorporaram o descritor de texturas LBP para treinamento de diferentes modelos de redes neurais para diversas aplicações, diferentes da proposta neste trabalho. No entanto, para efeitos de comparação, são exploradas outras publicações que abordaram o mesmo desafio de segmentação semântica de pavimentações asfálticas, utilizando tanto a arquitetura *U-Net* quanto a de outros modelos distintos.

O Capítulo 4 descreve todos os recursos, tanto de *software* quanto de *hardware*, empregados ao longo deste trabalho. Mais especificamete, a plataforma *Colab* e os recursos de *hardware* dedicados são detalhados, bem como as várias bibliotecas de código aberto, desenvolvidas com o intuito de promover a pesquisa e, inclusive, a exploração comercial dos algoritmos de IA.

O Capítulo 5 aborda os experimentos conduzidos com a rede neural *U-Net*. Mostra-se que a segmentação semântica é alcançada pelo modelo, e que o modelo pode ser treinado de diferentes formas. Testa-se a vantagem em se utilizar o descritor de texturas LBP, e confronta-se os resultados obtidos com os do mesmo modelo quando o uso convencional com apenas imagens rasterizadas é feito. Adicionalmente, neste capítulo, os resultados de todos os experimentos realizados são comparados com os publicados nos outros trabalhos mencionados no Capítulo 3.

O Capítulo 6 encerra este trabalho fazendo as considerações finais e apresentando propostas para novas tarefas.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo descreve os tópicos contidos no escopo do trabalho. A área de Aprendizado de Máquina será introduzida para abordar de maneira coesa a especialização em Aprendizagem Profunda, baseada no modelo do neurônio artificial. Nesse contexto, é necessário discorrer acerca da estrutura do neurônio artificial, desde a recepção dos valores de entrada até a determinação do valor de ativação da saída. Adicionalmente, serão apresentados detalhes sobre o processo de treinamento e otimização do neurônio artificial. Posteriormente, serão discutidas as técnicas que compõem uma *U-Net*, tais como convolução, camada de agrupamento e convolução transposta. Assim, pode-se definir a arquitetura *U-Net* adotada, esclarecendo a integração e o funcionamento conjunto das técnicas mencionadas.

Além dos métodos pertinentes à Aprendizagem de Máquina, o descritor de texturas LBP também será abordado. Este algoritmo será integrado ao processo de treinamento da *U-Net* para aprimorar a precisão do modelo em condições de iluminação adversas. Portanto, é imprescindível demonstrar seu funcionamento e suas propriedades, bem como justificar as vantagens de utilizar o mapa de texturas que ele produz.

2.1 Aprendizagem de Máquina

A Aprendizagem de Máquina é uma área do conhecimento que está inserida dentro do domínio da Inteligência Artificial (IA), e tem por objetivo a identificação automática de padrões significativos presentes em um conjunto de dados (SHALEV-SHWARTZ; BEN-DAVID, 2014). Os padrões identificados podem ainda ser utilizados para fazer previsões, tomar decisões ou realizar tarefas específicas sem instruções explícitas. Porém, em contraposição à abordagem convencional de IA, a Aprendizagem de Máquina não visa replicar de forma automatizada o comportamento inteligente humano. Seu propósito reside na exploração das vantagens e capacidades singulares dos computadores para complementar a inteligência humana, tipicamente realizando tarefas que ultrapassam as habilidades humanas.

A tecnologia fundamentada na Aprendizagem de Máquina encontra-se atualmente empregada em uma ampla gama de aplicações práticas, incluindo identificação facial em dispositivos de câmeras digitais, interpretação de comandos por voz, detecção de atividades suspeitas para prevenir fraudes em transações de cartões de crédito, e diversas outras áreas de atuação (RUSSELL; NORVIG, 2010).

O emprego da Aprendizagem de Máquina é justificado diante de problemas que exibem uma complexidade elevada e requerem uma capacidade adaptativa. Especificamente,

em tarefas que não viabilizam uma descrição detalhada de como devem ser executadas devido à complexidade dos padrões presentes. Diante disso, os métodos de Aprendizagem de Máquina possibilitam a capacidade de aprendizado e adaptação aos programas, oferecendo uma abordagem mais flexível para tratar a complexidade inerente desses problemas.

Tarefas como condução de veículos, tradução de textos, controle de qualidade e diagnósticos médicos são exemplos de atividades frequentemente realizadas por seres humanos. Embora muitas vezes as pessoas enfrentem esses desafios de forma intuitiva, as soluções geralmente exigem adaptação a cada situação e a condições imprevisíveis. Consequentemente, a implementação de um programa convencional capaz de abordar todos os cenários possíveis se tornaria extremamente difícil, dada a quantidade significativa de variáveis aleatórias envolvidas, resultando em uma necessidade excessiva de regras e exceções. Em outras palavras, programas convencionais permanecem estáticos após sua implementação, enquanto muitas tarefas estão sujeitas a mudanças ao longo do tempo.

Nesse contexto, a adaptabilidade da Aprendizagem de Máquina torna-se evidente ao permitir que os programas se ajustem dinamicamente às entradas. Exemplos de aplicações bem-sucedidas de Aprendizagem de Máquina incluem sistemas que interpretam escrita manual ou reconhecem discurso oral (RUSSELL; NORVIG, 2010).

A Aprendizagem de Máquina também pode ser utilizada para facilitar a análise de conjuntos de dados grandes e intrincados, como os dados astronômicos, previsões meteorológicas, motores de busca na internet e plataformas de comércio eletrônico (RUSSELL; NORVIG, 2010). Isso se deve à capacidade da Aprendizagem de Máquina em identificar padrões em conjuntos de dados extensos e complexos, viabilizando a identificação de características valiosas dentro desses conjuntos.

Existem muitos métodos estabelecidos na área de Aprendizagem de Máquina, como os citados em Sarker (2021b): regressão linear e logística, *Support Vector Machine* (SVM), *K-Nearest Neighbours* (KNN), árvore de decisão, K-Means, entre outros. Cada método apresenta atributos distintos que requerem avaliação prévia antes de sua aplicação em um contexto prático específico.

A Aprendizagem Profunda é uma especialização da Aprendizagem de Máquina e difere dos outros métodos citados em sua estrutura que é fundamentada no neurônio artificial. A interconexão coerente de vários neurônios artificiais forma as Redes Neurais Artificiais (RNAs), que são modelos reconhecidos pela capacidade de aprender representações complexas e pela adaptabilidade a diferentes tipos de entradas. As RNA são geralmente indicadas quando se requer a análise de imagens rasterizadas; no entanto, elas apresentam desafios significativos, como a necessidade de grandes volumes de dados e recursos computacionais substanciais para o treinamento e processamento dessas redes.

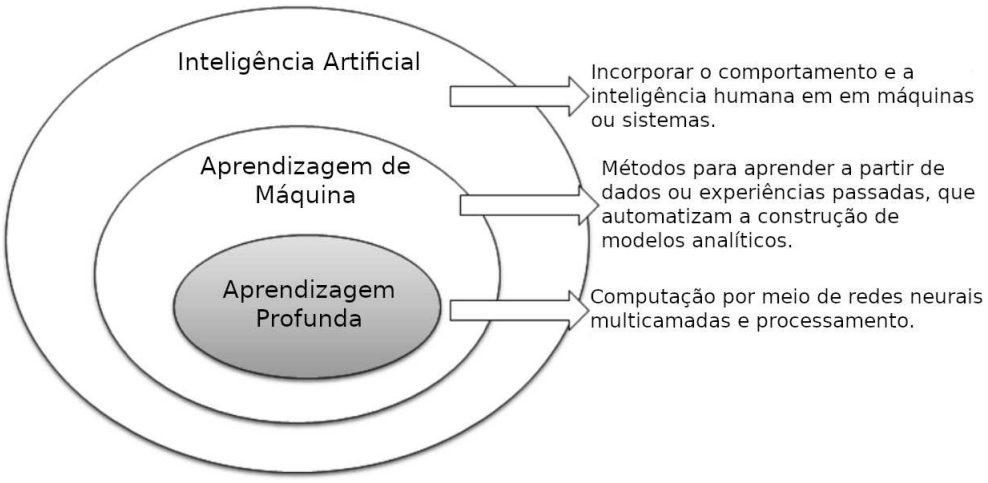
2.1.1 Redes Neurais Artificiais e Aprendizagem Profunda

O neurônio artificial, também conhecido como *perceptron*, é uma representação matemática baseada nos neurônios presentes em um cérebro biológico. Uma Rede Neural Artificial (RNA) é composta por uma coleção interconectada de neurônios artificiais, que operam de maneira entrelaçada para aprender coletivamente a partir das entradas, buscando otimizar seu resultado final (RUSSELL; NORVIG, 2010).

Cada conexão entre esses neurônios é análoga às sinapses no cérebro biológico, e tem a capacidade de transmitir sinais para outros neurônios. Um neurônio artificial recebe esses sinais, processa-os e pode enviar sinais para outros neurônios conectados. Em cada conexão, o sinal é representado por um número real, e a saída de cada neurônio é determinada por uma função não linear da soma ponderada de suas entradas. As conexões, denominadas arestas, possuem pesos que são ajustados durante o processo de aprendizado. Esses pesos aumentam ou diminuem a influência do sinal transmitido pela conexão. Além disso, os neurônios podem ter um limiar de ativação, onde um sinal só é enviado se a soma dos sinais recebidos ultrapassar esse limiar.

Os neurônios são organizados em camadas, formando uma estrutura comumente referida como "modelo", onde cada camada executa transformações específicas nos sinais provenientes da camada anterior, gerando um valor de ativação que é transmitida para a próxima camada. Esses sinais atravessam a rede, transitando da primeira camada de entrada até a última camada de saída. As RNAs são consideradas uma vertente da área de Aprendizagem de Máquina, e se a RNA incluir, duas ou mais camadas internas entre as camadas de entrada e saída, é denominada como Rede Neural Profunda, constituindo a área de Aprendizagem Profunda, conforme o diagrama da Figura 1.

Figura 1 – Diagrama das áreas de Inteligência Artificial, Aprendizagem de Máquina e Aprendizagem Profunda.



Fonte: adaptado e traduzido de Sarker (2021a).

Um modelo tem a capacidade de realizar previsões acerca de um determinado assunto com um nível específico de acurácia. Para aprimorar essa acurácia, o modelo pode ser submetido a um processo de treinamento no qual uma base de dados é utilizada para ajustar os valores dos pesos iterativamente, visando minimizar os erros observados nas previsões.

As camadas intermediárias nas redes neurais mantêm e processam informações sobre a importância das entradas, utilizando essas informações para avaliar e determinar a saída.

Os algoritmos mais populares em Aprendizagem Profunda são: *Multi-layer Perceptron* (MLP), e *Convolutional Neural Network* (CNN, ou ConvNet) (SARKER, 2021b), que serão apresentados adiante.

2.1.2 Tipos de Técnicas em Aprendizagem de Máquina

Através do processo de treinamento a detecção de padrões em um conjunto de dados é realizada. De forma geral, existem quatro grandes categorias de algoritmos em Aprendizagem de Máquina (SARKER, 2021b):

- Supervisionado: é um algoritmo que é treinado com o propósito de aprender uma função que possa mapear entradas para saídas, fazendo uso de um conjunto de pares de dados rotulados. Esse processo viabiliza a capacidade de realizar predições ou classificações em novos conjuntos de dados. Esse método é aplicado quando se têm objetivos específicos a serem alcançados com um conjunto conhecido de entradas. As tarefas mais comuns na aprendizagem supervisionada são a “classificação”, que separa os dados em categorias, e a “regressão”, que reduz os dados a uma função matemática contínua;

- Não-Supervisionado: é um algoritmo que analisa conjuntos de dados não rotulados a fim de identificar e extrair padrões, tendências e estruturas significativas para fins exploratórios. As tarefas mais comuns de aprendizado não-supervisionado incluem agrupamento dos dados, aprendizado de características, redução de dimensionalidade, descoberta de regras de associação, detecção de anomalias, entre outros;

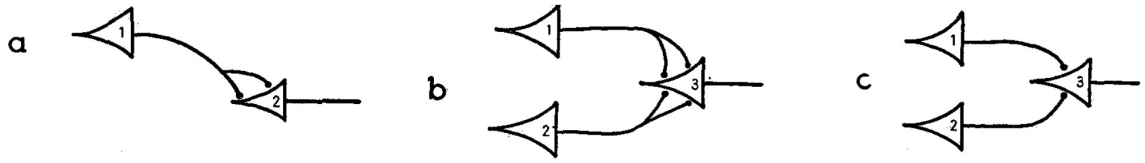
- Semi-Supervisionado: é uma combinação dos métodos supervisionado e não supervisionado pois opera tanto em dados rotulados quanto não rotulados. Tem o objetivo de fornecer um resultado melhor para predição do que o obtido apenas com os dados rotulados. Algumas áreas de aplicação do aprendizado semi-supervisionado incluem tradução automática, detecção de fraudes, classificação de textos e imagens, entre outros;

- Por Reforço: é um tipo de algoritmo de Aprendizagem de Máquina que capacita agentes de *software* e máquinas a avaliarem automaticamente o comportamento ótimo em um contexto ou ambiente específico para aprimorar sua eficiência. Baseia-se em recompensas ou penalidades para tomar ações que maximizem a recompensa ou minimizem o risco. É útil para treinar modelos em áreas como robótica, manufatura e logística;

2.1.3 O *Perceptron*

O *perceptron* é a estrutura fundamental do campo de inteligência artifical. Embora o primeiro neurônio tenha sido modelado matematicamente em McCulloch e Pitts (1943), sua funcionalidade estava limitada a entradas e saídas binárias, essencialmente executando apenas operações de lógica booleana. A Figura 2 foi retirada da publicação original, e retrata alguns exemplos de neurônios e como as conexões e operações foram representadas.

Figura 2 – Exemplos de neurônios idealizados por McCulloch e Pitts. (a) Atrasador. (b) Porta lógica OU. (c) Porta lógica E.



Fonte: McCulloch e Pitts (1943).

A Figura 2.a ilustra como os neurônios de McCulloch-Pitts podem ser ordenados para formar um circuito atrasador, cuja operação lógica é definida pela equação (2.1),

$$\text{a)} \quad N_2(t). \equiv .N_1(t-1). \tag{2.1}$$

A Figura 2.b representa o circuito booleano OU, e é definida pela equação (2.2),

$$\text{b)} \quad N_3(t). \equiv .N_1(t-1) \vee N_2(t-1), \tag{2.2}$$

enquanto a Figura 2.c representa o circuito booleano E, e é definida pela equação (2.3),

$$\text{c)} \quad N_3(t). \equiv .N_1(t-1)N_2(t-1). \tag{2.3}$$

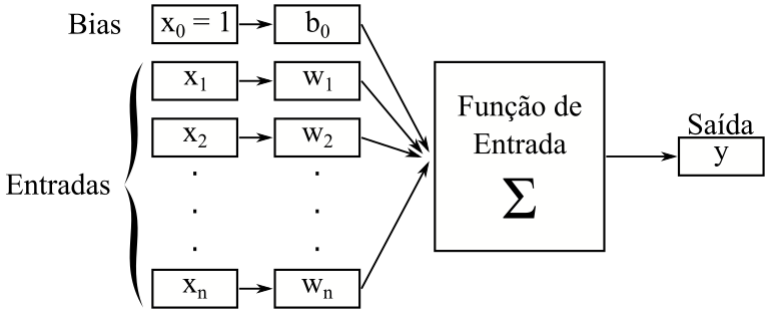
Posteriormente, no campo da neurociência e psicologia, Hebb (1949) apresentou uma nova proposta sobre o funcionamento da comunicação neuronal e sua relação com a capacidade de aprendizado. Hebb definiu a Lei de Hebb, segundo a qual a conexão entre dois neurônios se fortalece quando são ativados simultaneamente e enfraquece quando

são ativados de forma independente. Consequentemente, o modelo de McCulloch-Pitts precisava ser alterado para incorporar a nova proposta biológica.

Combinando o modelo de McCulloch-Pitts com as descobertas de Hebb, Rosenblatt (1957) introduziu uma abordagem aprimorada que resultou no desenvolvimento do modelo do *perceptron*. Esse modelo permanece como uma referência relevante até os dias atuais, em que as entradas são multiplicadas por pesos para representar a intensidade da conexão. A representação matematica pode ser representada como uma transformação de um vetor de entrada x_1, x_2, ... x_n em um sinal de saída denominado valor de ativação y.

A Figura 3 representa o modelo simplificado de *perceptron*. Nesse modelo, os pesos são representados pelo vetor w_0, w_1, ... w_n, e multiplicam as respectivas entradas. A entrada x_0 é a única que sempre tem valor fixo `1`, e é utilizado para proporcionar um valor de *bias* através do peso w_0. Os resultados de todas as multiplicações são somados para gerarem o valor de ativação y.

Figura 3 – Estrutura simplificada do *Perceptron*.

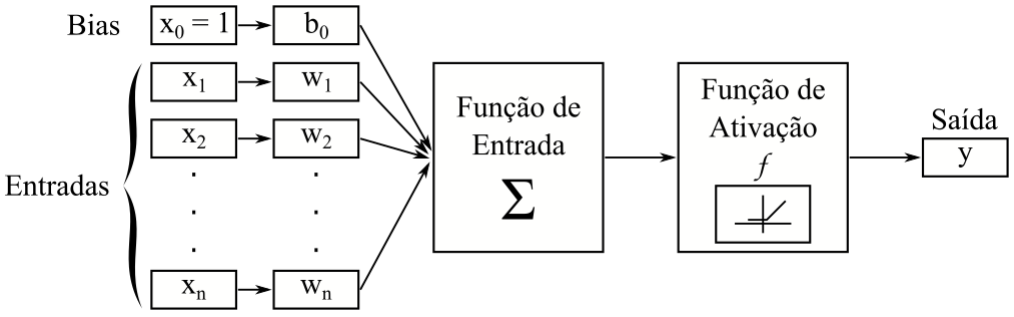


Fonte: De autoria própria.

2.1.4 Funções de Ativação

A função de ativação tem o objetido de transformar a soma ponderada das entradas de um *perceptron* em um valor de ativação na saída, e está adicionada na Figura 4.

Figura 4 – *Perceptron* com função de ativação.



Fonte: De autoria própria.

A função de ativação empregada no modelo de perceptron por Rosenblatt é definida pela equação (2.4), e consistia em uma função degrau simples,

$$f(x) = \begin{cases} 0 \text{ , se } x < limiar, \\ 1 \text{ , caso contrário} \end{cases}, \tag{2.4}$$

no qual a saída era determinada pela comparação entre a soma ponderada das entradas e um limiar predefinido. Essa função possibilitava ao *perceptron* realizar classificações binárias diretas, como a separação de pontos em duas categorias distintas, contudo, limitada apenas a dados linearmente separáveis (MINSKY; PAPERT, 1969).

Com o objetivo de capacitar uma rede a lidar eficazmente com a distribuição de dados não lineares, muitas outras funções de ativação foram estudadas. Dentre as listadas por Dubey, Singh e Chaudhuri (2021), a mais relevantes para este trabalho estão a função sigmoide, a função Tangente Hiperbólica (Tanh), a função *Rectified Linear Unit* (ReLU), e a função *Softmax*.

A presença da função de ativação representou um desafio significativo no processo de treinamento de redes neurais. O algoritmo retropropagação foi popularizado por Rumelhart, Hinton e Williams (1986) consolidou-se como um padrão na área de Aprendizagem Profunda.

O algoritmo retropropagação será discutido na Subseção 2.1.6, e é um método otimizado para ajustar os pesos de uma rede neural, visando minimizar o erro global por meio do cálculo do gradiente da função de custo. Esse cálculo é viabilizado por meio da aplicação da regra da cadeia na derivação da função de custo de saída, retropropagando o erro desde a camada de saída, pelas camadas internas até a camada de entrada. Isso possibilita a determinação do impacto de cada peso na função de custo global. O gradiente resultante é então aplicado iterativamente na atualização dos pesos, orientando-os na direção que reduz o erro de predição da rede.

O cálculo do gradiente é realizado por meio das derivadas parciais da função de custo na saída em relação a cada peso da rede neural. É importante destacar que, quanto maior for a variação encontrada (a magnitude do gradiente), maior será o ajuste aplicado, influenciando diretamente na atualização dos pesos durante o treinamento da rede neural. Esse processo é repetido várias vezes para fazer o ajuste dos pesos até que seja estabelecido um valor que minimiza a diferença entre o valor obtido e o esperado (RUMELHART; HINTON; WILLIAMS, 1986) .

2.1.4.1 Função Sigmoide

A função sigmoide é definida pela equação (2.5), conforme apresentado em Dubey, Singh e Chaudhuri (2021),

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.5}$$

Essa foi uma das primeiras funções de ativação a serem utilizadas, e possibilitou o aprendizado de padrões complexos em um conjunto de dados por uma rede formada pela organização em camadas concatenadas de vários *perceptrons* interconectados.

A função sigmoide é utilizada para transformar os valores de entrada em uma faixa de saída centrada em `0,5` e variando entre `0` e `1`. Sua diferenciabilidade em todos os pontos é uma característica relevante, embora sua derivada tenha um valor máximo de `0,25` no ponto médio da função. A reduzida amplitude da derivada da função sigmoide desempenha um papel significativo no problema da dissipação do gradiente em redes neurais profundas. Esse fenômeno pode levar a uma redução marcante na magnitude do gradiente de erro à medida que a retropropagação alcança as camadas iniciais, resultando em ajustes muito pequenos nos pesos dessas camadas. Como consequência, isso facilita a transferência de informações menos relevantes das camadas iniciais para as subsequentes durante o processo de propagação direta na rede neural.

Além disso, é importante destacar que o custo computacional associado ao uso da função sigmoide é relativamente alto devido à sua natureza exponencial, o que pode impactar significativamente no tempo de processamento em operações de grandes conjuntos de dados ou redes mais complexas.

2.1.4.2 Função Tangente Hiperbólica (Tanh)

A função Tangente Hiperbólica (Tanh) é definida pela equação (2.6), conforme apresentado em Dubey, Singh e Chaudhuri (2021),

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.6}$$

Pode-se perceber pela Figura 5 que as funções Tanh e sigmoide compartilham um formato gráfico muito semelhante, ambas de natureza exponencial, com diferenciabilidade em todos os pontos, embora apresentem a mesma desvantagem em relação ao alto custo computacional ao serem utilizadas.

No entanto, algumas características da função Tanh a tornam mais atraentes que a sigmoide. Sua amplitude varia de `-1` a `1`, centrada em `0`, proporcionando simetria. Esta propriedade simétrica é relevante, pois pode favorecer uma distribuição melhor dos pesos, facilitando a convergência durante a etapa de treinamento. Adicionalmente, a amplitude da derivada da função Tanh apresenta um pico mais elevado, o que pode reduzir os efeitos associados à dissipação do gradiente. É relevante mencionar que o custo computacional.

2.1.4.3 Função *Rectified Linear Unit* (ReLU)

A função *Rectified Linear Unit* (ReLU), definida pela equação (2.7), foi introduzida por Hahnloser et al. (2000),

$$f(x) = max(x, 0) = \begin{cases} x \text{ , se } x > 0, \\ 0 \text{ , caso contrário} \end{cases}, \tag{2.7}$$

e representa, provavelmente, uma das funções de ativação mais conhecidas e utilizadas.

Se a entrada for positiva, a saída será mantida igual a entrada, caso contrário, a saída permanecerá em zero. Essa transformação é o que permite o estabelecimento da relação de não linearidade entre os dados de entrada e saída, e o aprendizado de relações complexas, pois a presença de valores zero para entradas negativas facilita na separação de padrões e na representação de características essenciais dos dados.

A simplicidade da equação (2.7) torna o custo computacional significativamente mais baixo e, consequentemente maior velocidade de operação em relação às funções de ativação anteriores.

A função ReLU possui derivativa igual a `1` nos pontos positivos do eixo x, e igual a `0` em todos os demais. A derivativa constante e unitária soluciona o problema da dissipação do gradiente em redes profundas.

Contudo, é preciso destacar que a propriedade responsável por conferir a não linearidade desejada à função ReLU (a anulação das entradas negativas) também pode provocar um fenômeno conhecido como "problema do ReLU morto" (LU, 2020). Entradas negativas levam os pesos associados a permanecerem sem ajuste em relação às variações de erro, uma vez que, durante a propagação direta, a função de ativação anula a saída; na fase de retropropagação, o gradiente é anulado, resultando na ausência de alteração nos pesos correspondentes.

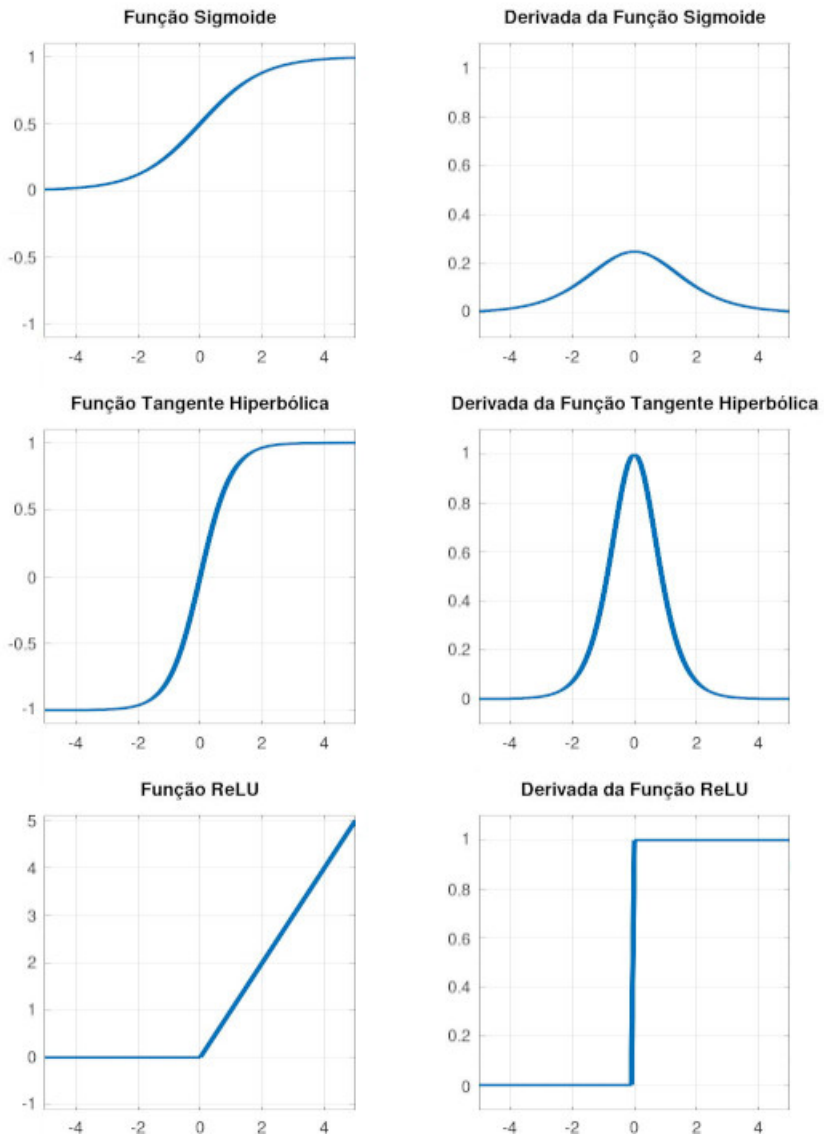
2.1.4.4 Função *Softmax*

A função *Softmax* transforma um vetor de valores reais em uma distribuição de probabilidade sobre as classes possíveis. Matematicamente, é definida pela equação (2.8), conforme apresentada em Bridle (1990),

$$f(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}}, \tag{2.8}$$

em que y é o vetor de entrada da função *Softmax*, y_i é o i-ésimo elemento do vetor, e n é o número de elementos e também o número de classes diferentes na saída.

Figura 5 – Funções de ativação e suas respectivas derivadas.



Fonte: De autoria própria.

Isso permite interpretar esses valores como probabilidades, pois o conjunto de saída foi normalizado para representar uma distribuição de probabilidade.

Geralmente, a função *Softmax* é empregada apenas como a função de ativação da camada final de uma rede neural, e apenas quando a saída contém mais de duas classes diferentes. Assim, a saída da rede ficará na forma de uma distribuição de probabilidades que representa as diversas classes de saída previstas.

2.1.4.5 Outras Funções de Ativação

Em Dubey, Singh e Chaudhuri (2021) são expostas diversas outras funções de ativação, como a *Leaky* ReLU, a *Parametric* ReLU e a *Exponential Linear Units* (ELU). Estas foram desenvolvidas com o propósito de mitigar o problema conhecido como "ReLU

morto", introduzindo algum grau adicional de complexidade na função de ativação. No entanto, a proposta deste trabalho não inclui uma análise comparativa do desempenho das diferentes funções de ativação, e considerando a quantidade de variedades existentes, optou-se por não abordar a discussão de outras funções de ativação.

2.1.5 Propagação Direta

O processo de propagação direta representa a operação direta e progressiva de uma rede neural. Essa estrutura, representada na Figura 6, é também conhecida como *Multilayer Perceptron* (MLP) e é o modelo primordial no que concerne as RNAs (GOODFELLOW; BENGIO; COURVILLE, 2016).

Nesse modelo, as entradas são recebidas pelos *perceptrons* da primeira camada, passam pela multiplicação ponderada pelos pesos, têm seus resultados somados e, por fim, são submetidas a uma função de ativação. Esse processo gera um valor de ativação, o qual é transmitido para a próxima camada de *perceptrons*. Esse processo se repete consecutivamente até atingir a última camada da rede, onde a predição é finalmente realizada.

2.1.6 Treinamento com Retropropagação

O algoritmo de retropropagação foi introduzido por Werbos e John (1974) e popularizado por Rumelhart, Hinton e Williams (1986) como um dos primeiros métodos de treinamento a demonstrar eficácia no ajuste dos pesos dos *perceptrons*. Seu principal objetivo é minimizar o erro da rede neural através de boas representações internas.

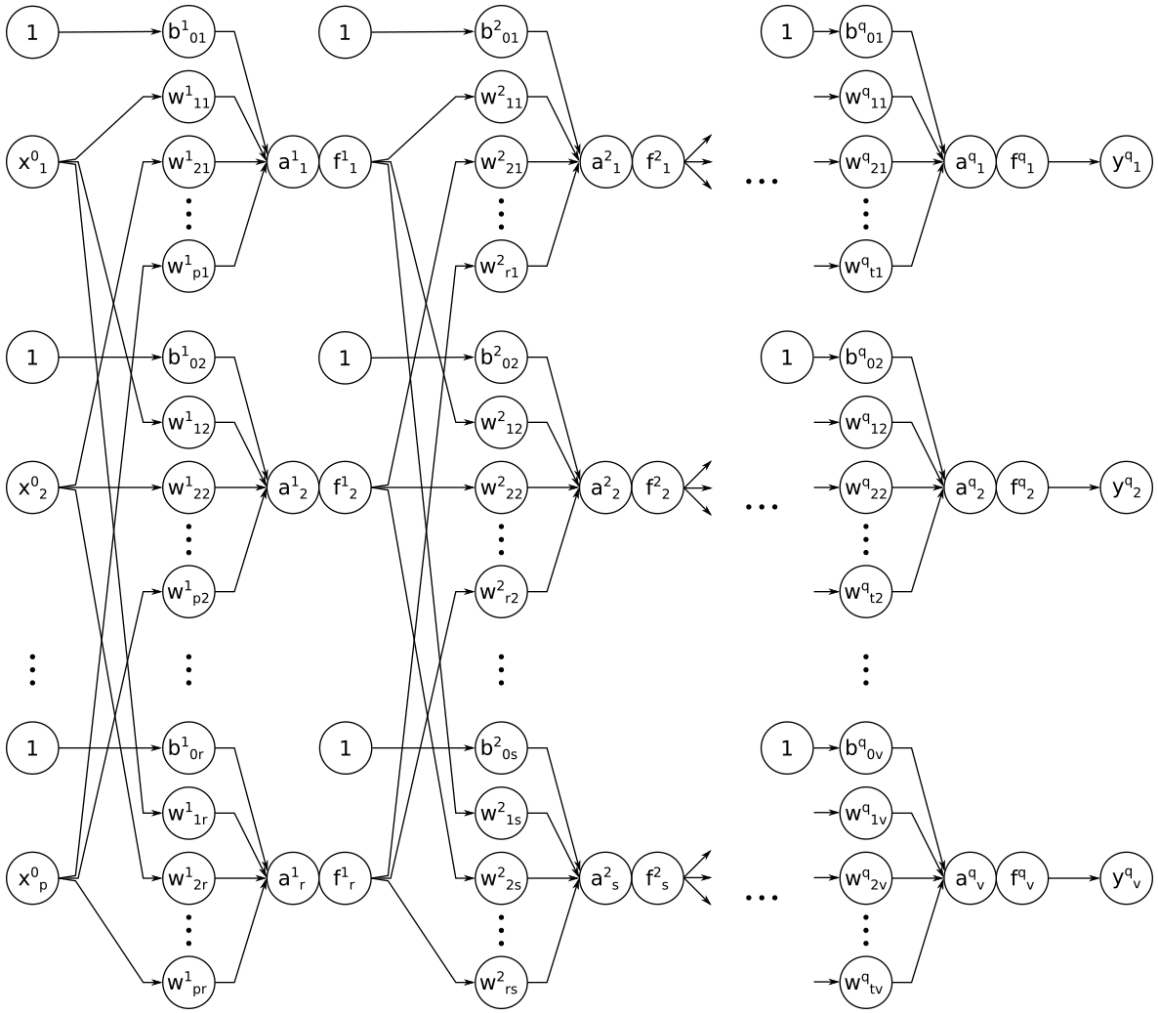
O algoritmo de gradiente descendente é um dos otimizadores mais amplamente empregados em conjunto com o algoritmo de retropropagação, devido ao seu pioneirismo e à comprovação reiterada de sua funcionalidade. O método consiste em calcular o gradiente da função de custo para estimar um valor de ajuste dinâmico e gradual nos pesos dos *perceptrons*, o qual diminui à medida que o erro se aproxima do seu mínimo.

A função de custo J é também conhecida como "função de erro" ou "função de perda", sendo a fórmula do erro quadrático médio, definida na equação (2.9), a mais tradicionalmente utilizada,

$$J(x, y, w) = \frac{1}{N} \sum_{d=1}^{N} (\hat{y_d} - y_d)^2, \tag{2.9}$$

em que x é o conjunto de entradas, y é o conjunto de saídas verdadeiras, w é o conjunto de pesos da rede, N é o número de elementos do conjunto x ou y, y_d é o valor verdadeiro, e $\hat{y_d}$ é o valor predito pela saída da rede.

Figura 6 – Estrutura genérica de uma rede neural.



Fonte: De autoria própria.

A equação (2.9) denota que a função de custo J depende do conjunto de entradas (x), do conjunto de saídas verdadeiras (y), e do conjunto de pesos (w) da rede neural (deve ser lembrado que o valor predito $\hat{y_d}$ é a saída da função de ativação $f(x_d, w_d)$).

Muitas vezes, no contexto de Aprendizagem Profunda, a equação (2.9) é modificada com a introdução de um fator constante de $\frac{1}{2}$, assumindo a forma da equação (2.10),

$$J(x, y, w) = \frac{1}{2N} \sum_{d=1}^{N} (\hat{y_d} - y_d)^2. \tag{2.10}$$

Isso é realizado com o intuito de facilitar os cálculos subsequentes, pois o fator constante é anulado durante a derivação parcial para o cálculo do gradiente (BISHOP, 2006). Essencialmente, essa escolha não prejudica o treinamento, já que o fator é incorporado à taxa de aprendizado durante a atualização dos pesos.

Neste trabalho, a notação de um peso (w) é representada pela forma dada na

equação (2.11),

$$w_{ji}^{k}, \tag{2.11}$$

em que k representa o índice da camada em que o *perceptron* está posicionado, j é o índice da saída do *perceptron* da camada anterior $(k-1)$, e i é o índice do *perceptron* na camada k.

É importante lembrar que todo *perceptron* possui um peso *bias* (b_i^k) associado que pode ser incorporado à representação da equação (2.11) pela equação (2.12),

$$w_{0i}^{k} = b_i^k. \tag{2.12}$$

Assim, o valor de ativação (a_i^k) de um *perceptron* que está na camada k pode ser definido pela equação (2.13),

$$a_i^k = b_i^k + \sum_{j=1}^{n_{k-1}} w_{ji}^k o_j^{k-1} = \sum_{j=0}^{n_{k-1}} w_{ji}^k o_j^{k-1}, \tag{2.13}$$

em que n_{k-1} é a quantidade de *perceptrons* da camada anterior $(k-1)$, e o_j^{k-1} é o valor de ativação do *perceptron* j da camada anterior $(k-1)$.

O gradiente é calculado através da derivada parcial de J em relação a cada peso da rede w_{ji}^k. Se J for expresso na forma da equação (2.10), o gradiente poderá ser, então, definido pela equação (2.14),

$$\frac{\partial J(x,y,w)}{\partial w_{ji}^k} = \frac{\partial \left(\frac{1}{2N} \sum_{d=1}^{N} (\hat{y}_d - y_d)^2 \right)}{\partial w_{ji}^k}. \tag{2.14}$$

Pela regra do múltiplo constante das derivadas, N pode ser deslocado para fora da derivada, conforme a equação (2.15),

$$\frac{\partial J(x,y,w)}{\partial w_{ji}^k} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial \left(\frac{1}{2} (\hat{y}_d - y_d)^2 \right)}{\partial w_{ji}^k}. \tag{2.15}$$

A equação (2.16) mostra uma forma comum de expressar o erro quadrático médio para um único elemento,

$$J_d = \frac{1}{2} (\hat{y}_d - y_d)^2, \tag{2.16}$$

dessa forma substituindo a equação (2.16) na equação (2.15), obtém-se a equação (2.17),

$$\frac{\partial J(x,y,w)}{\partial w_{ji}^k} = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial J_d}{\partial w_{ji}^k}. \tag{2.17}$$

A derivada parcial da função de custo J em relação aos pesos w_{ji}^k pode ser desenvolvida através da regra da cadeia, como demonstra a equação (2.18),

$$\frac{\partial J(x,y,w)}{\partial w_{ji}^k} = \frac{1}{N}\sum_{d=1}^{N}\frac{\partial J_d}{\partial a_i^k}\frac{\partial a_i^k}{\partial w_{ji}^k}, \tag{2.18}$$

em que o termo a_i^k é a função de ativação do *perceptron* i na camada k antes de ser submetida à função de ativação não linear. A decomposição pela regra da cadeia gera dois termos: o primeiro pode ser substituído em uma variável auxiliar expresso pela equação (2.19),

$$\frac{\partial J_d}{\partial a_i^k} = \delta_i^k, \tag{2.19}$$

em que o termo de erro δ_i^k representa a variação na função de custo J em resposta a uma variação em sua própria função de ativação; e o segundo termo pode ser desenvolvido de acordo com a equação (2.20),

$$\begin{aligned}\frac{\partial a_i^k}{\partial w_{ji}^k} &= \frac{\partial \sum_{l=0}^{n_{k-1}} w_{li}^k o_l^{k-1}}{\partial w_{ji}^k} \\ &= \frac{\partial w_{0i}^k o_0^{k-1}}{\partial w_{ji}^k} + \frac{\partial w_{1i}^k o_1^{k-1}}{\partial w_{ji}^k} + ... + \frac{\partial w_{ji}^k o_j^{k-1}}{\partial w_{ji}^k} + ... + \frac{\partial w_{n_{k-1}i}^k o_{n_{k-1}}^{k-1}}{\partial w_{ji}^k} \\ &= o_j^{k-1}.\end{aligned} \tag{2.20}$$

Observa-se que o resultado da derivada parcial de a_i^k em relação ao peso w_{ji}^k é diretamente o valor da saída do *perceptron* j da camada anterior $(k-1)$. Essa relação permite a otimização da implementação do código de treinamento de retropropagação pois torna desnecessário fazer o cálculo do termo da derivada parcial expresso pela equação (2.20), bastando que o valor gerado durante a fase de propagação direta seja guardado para substituir o valor quando o gradiente for calculado.

Portanto, combinando os resultados obtidos nas equações (2.19) e (2.20) na equação (2.18), obtém-se a forma geral do gradiente expresso pela equação (2.21),

$$\frac{\partial J(x,y,w)}{\partial w_{ji}^k} = \frac{1}{N}\sum_{d=1}^{N}\delta_i^k o_j^{k-1}. \tag{2.21}$$

Essa forma simplificada representa o gradiente da função de custo em relação ao peso específico w_{ji}^k ao longo de todos os elementos de treinamento N.

O gradiente em relação a um peso w_{ji}^k é dado pela média dos termos de erro δ_i^k multiplicados por o_j^{k-1}. Em termos mais simples, o gradiente de um peso é dado pelo valor de ativação o_j^{k-1} do *perceptron* que está no índice j da camada anterior $(k-1)$ multiplicado pela média das variações das funções de erro em relação à função de ativação do próprio *perceptron* que está no índice i da camada k.

2.1.6.1 O Gradiente na Camada de Saída

O treinamento por retropropagação deve começar ajustando os pesos da última camada e ir gradativamente seguindo no sentido inverso pelo caminho da propagação direta através das camadas internas até atingir os pesos da camada de entrada (BISHOP, 2006). A fórmula geral do gradiente dado pela equação (2.21) deve ser desenvolvida para a última camada q. Como o valor do termo o_j^{q-1} é conhecido, basta desenvolver o termo de erro δ_i^q definido pela equação (2.19). Isso pode ser feito substituindo-se a equação (2.16), como mostra a equação (2.22),

$$\delta_i^q = \frac{\partial J_d}{\partial a_i^q} = \frac{\partial \left(\frac{1}{2}(\hat{y}_d - y_d)^2\right)}{\partial a_i^q}. \tag{2.22}$$

É importante observar que $\hat{y}_d$ é a predição ou a saída da função de ativação. Ou seja, $\hat{y}_d$ é o resultado de uma função que depende do termo a_i^q. Para que essa relação fique clara, pode-se substituir $\hat{y}_d$ por $f(a_i^q)$, e reescrever na forma da equação (2.23),

$$\frac{\partial J_d}{\partial a_i^q} = \frac{\partial \left(\frac{1}{2}(f(a_i^q) - y_d)^2\right)}{\partial a_i^q}. \tag{2.23}$$

Aplicando-se a regra da cadeia na equação (2.23), obtém-se na equação (2.24),

$$\frac{\partial J_d}{\partial a_i^q} = \frac{\partial \left(\frac{1}{2}(f(a_i^q) - y_d)^2\right)}{\partial \left(f(a_i^q) - y_d\right)} \frac{\partial \left(f(a_i^q) - y_d\right)}{\partial a_i^q} = \left(f(a_i^q) - y_d\right) \frac{\partial f(a_i^q)}{\partial a_i^q}. \tag{2.24}$$

Deve-se observar que o valor de $f(a_i^q) = \hat{y_d}$ é o valor da saída da função de ativação do próprio *perceptron*. Por isso, é possível reverter o primeiro termo $f(a_i^q)$, da equação (2.24), por $\hat{y_d}$, conforme exibido na equação (2.25),

$$\frac{\partial J_d}{\partial a_i^q} = (\hat{y}_d - y_d)\frac{\partial f(a_i^q)}{\partial a_i^q}. \tag{2.25}$$

Assim, substituindo a equação (2.25) na equação (2.21), obtém-se finalmente a forma equação (2.26) para os gradientes dos *perceptrons* da última camada,

$$\frac{\partial J(x, y, w)}{\partial w_{ji}^q} = \frac{1}{N}\sum_{d=1}^{N} \delta_i^q o_j^{q-1} = \frac{1}{N}\sum_{d=1}^{N}((\hat{y}_d - y_d)\frac{\partial f(a_i^q)}{\partial a_i^q} o_j^{q-1}). \tag{2.26}$$

Se a ativação ReLU for utilizada, a derivada da função de ativação $\partial f(a_i^q)/\partial a_i^q$ será `1`, se $\hat{y}_d$ for positivo, e `0` caso contrário. O cálculo do gradiente na camada de saída é definido pela equação (2.27),

$$\frac{\partial J(x,y,w)}{\partial w_{ji}^q} = \begin{cases} \frac{1}{N}\sum_{d=1}^{N}((\hat{y}_d - y_d)o_j^{q-1}), \text{ se } \hat{y}_d > 0 \\ 0, \text{ caso contrário} \end{cases}. \tag{2.27}$$

2.1.6.2 O Gradiente nas Camadas Internas

A forma geral do gradiente na equação (2.21) também pode ser aplicada às camadas internas. No entanto, nas camadas internas, é necessário observar que a função de custo J_d é retropropagada, sendo dessa forma influenciada pelos valores de ambas as ativações das da camadas seguinte $(k+1)$ e atual (k). Dessa forma, faz-se necessário aplicar a regra da cadeia de acordo com a equação (2.28),

$$\delta_i^k = \frac{\partial J_d}{\partial a_i^k} = \sum_{l=1}^{n_{k+1}} \frac{\partial J_d}{\partial a_l^{k+1}} \frac{\partial a_l^{k+1}}{\partial a_i^k}. \tag{2.28}$$

A somatória na equação (2.28) indica que todos os *perceptrons* da camada seguinte $(k+1)$ contribuem para o termo de erro δ_i^k. No entando, devido ao fato de a entrada de todos os *bias* da rede terem valor fixo unitário, ou seja, não estar diretamente envolvido nas ativações das camadas anteriores pois não está conectado à saída do *perceptron* i da camada k, l não assume o valor `0` na somatória.

Atentando-se para a relação apresentada pela equação (2.19), pode-se extendê-la na forma da equação (2.29),

$$\frac{\partial J_d}{\partial a_i^{k+1}} = \delta_i^{k+1}. \tag{2.29}$$

Substituindo a equação (2.29) na equação (2.28), produz-se a equação (2.30),

$$\frac{\partial J_d}{\partial a_i^k} = \sum_{l=1}^{n_{k+1}} \delta_l^{k+1} \frac{\partial a_l^{k+1}}{\partial a_i^k}. \tag{2.30}$$

Relembrando a equação (2.13), que define o valor de ativação a_i^k, é necessário observar que o termo o_j^{k-1} também pode ser escrito sob a forma $f(a_j^{k-1})$, pois refere-se à saída da função de ativação do *perceptron* da camada anterior. Ajustando os índices de acordo com a equação (2.30), pode-se estabelecê-la na forma da equação (2.31),

$$a_l^{k+1} = \sum_{j=1}^{n_k} w_{jl}^{k+1} f(a_j^k). \tag{2.31}$$

Então a derivada parcial da equação (2.31) pode ser desenvolvida na equação (2.32),

$$\begin{aligned}\frac{\partial a_l^{k+1}}{\partial a_i^k} &= \frac{\partial \sum_{j=1}^{n_k} w_{jl}^{k+1} f(a_j^k)}{\partial a_i^k} \\ &= w_{1l}^{k+1}\frac{\partial f(a_1^k)}{\partial a_i^k} + w_{2l}^{k+1}\frac{\partial f(a_2^k)}{\partial a_i^k} + ... + w_{il}^{k+1}\frac{\partial f(a_i^k)}{\partial a_i^k} + ... + w_{n_k l}^{k+1}\frac{\partial f(a_{n_k}^k)}{\partial a_i^k} \\ &= w_{il}^{k+1}\frac{\partial f(a_i^k)}{\partial a_i^k}.\end{aligned} \tag{2.32}$$

Substituindo a equação (2.32) na equação (2.30), obtém-se a equação (2.33),

$$\begin{aligned}\frac{\partial J_d}{\partial a_i^k} &= \sum_{l=1}^{n_{k+1}} \delta_l^{k+1} w_{il}^{k+1}\frac{\partial f(a_i^k)}{\partial a_i^k} \\ &= \frac{\partial f(a_i^k)}{\partial a_i^k}\sum_{l=1}^{n_{k+1}} \delta_l^{k+1} w_{il}^{k+1}.\end{aligned} \tag{2.33}$$

Finalmente, substituindo a equação (2.33) na forma geral do gradiente da equação (2.21), obtém-se a equação (2.34), que expressa o cálculo do gradiente nas camadas internas,

$$\begin{aligned}\frac{\partial J(x,y,w)}{\partial w_{ji}^k} &= \frac{1}{N}\sum_{d=1}^{N} \delta_i^k o_j^{k-1} \\ &= \frac{1}{N}\sum_{d=1}^{N}\left(\frac{\partial f(a_i^k)}{\partial a_i^k}\left(\sum_{l=1}^{n_{k+1}} \delta_l^{k+1} w_{il}^{k+1}\right) o_j^{k-1}\right).\end{aligned} \tag{2.34}$$

2.1.6.3 Atualização dos Pesos

Na retropropagação, o valor do termo de erro δ_i^k depende dos valores dos termos de erro δ_l^{k+1} que estão na camada seguinte. Dessa forma, o erro retropropaga, da última camada até a primeira. De forma simplificada, tudo que é necessário é computar os erros da camada de saída $(\hat{y}_d - y)$. Então, os termos de erro para a camada anterior são calculados fazendo-se a soma de todos os produtos dos termos de erros δ_l^{k+1} com os pesos w_{il}^{k+1} (observe que ambos os termos estão posicionados na camada seguinte $k+1$), e escalá-los pelo fator $\partial f(a_i^k)/\partial a_i^k$, e repetir o processo até atingir a primeira camada.

Além do mais, pelo fato de os cálculos para a fase de retropropagação serem dependentes dos valores das funções de entrada a_j^k e funções de ativações o_j^k dos *perceptrons* nas camadas anterior e seguintes, todos esses valores devem ser processados antes de a fase de retropropagação começar. Assim, a fase de propagação direta precede a fase de retropropagação para cada iteração do gradiente descendente. Na fase de propagação, os valores das funções de entrada a_j^k e funções de ativações o_j^k serão armazenados para

serem usados na fase de retropropagação. Uma vez que a fase de retropropagação estiver completa, e as derivadas parciais forem conhecidas, os pesos e *bias* podem ser atualizados, de acordo com a equação (2.35),

$$w_{novo} = w_{antigo} - \eta \frac{\partial J(x, y, w)}{\partial w_{ji}^{k}}, \tag{2.35}$$

em que η é a taxa de aprendizagem definida no momento da compilação do modelo. Esse processo se repete até que um mínimo local seja encontrado ou um critério de convergência seja cumprido.

2.1.6.4 Função de Custo Entropia Cruzada

A função erro quadrático médio foi apresentada na equação (2.16) e discutida em relação ao treinamento por retropropagação, devido à sua importância histórica pioneira e sua simplicidade. No entanto, para problemas de classificação, a função de custo mais indicada é a Entropia Cruzada Binária (BCE, do inglês *Binary Cross-Entropy Loss*).

A entropia mensura o quão uniforme é uma distribuição de probabilidades. Quanto mais uniforme, maior é a entropia, enquanto uma concentração dos eventos resulta em uma entropia mais baixa.

A entropia cruzada é particularmente útil em problemas de classificação, calculando a quantidade de informação necessária - geralmente expressa em um valor de probabilidade entre `0` e `1` - para quantificar a diferença entre as distribuições de probabilidade previstas pelo modelo e as distribuições reais (WALI, 2022). Dessa forma, o modelo é penalizado de acordo com a distância em que as previsões estão dos valores reais.

A fórmula da entropia cruzada está expressa na equação (2.36),

$$J(x, y, w) = -\frac{1}{N} \sum_{d=1}^{N} y_d \log(\hat{y_d}) + (1 - y_d) \log(1 - \hat{y_d}), \tag{2.36}$$

em que N é o número total de previsões, y_i são os valores reais, e $\hat{y_i}$ os valores previstos pela rede.

2.1.6.5 Otimizador ADAM

O algoritmo *Adaptive Moment Estimation* (ADAM) apresenta a característica de ter uma taxa de aprendizado adaptativa (KINGMA; BA, 2017). Essa adaptação é calculada considerando os primeiro e segundo momentos do gradiente, o que contribui para uma convergência mais eficiente, especialmente em situações em que os gradientes são ruidosos ou esparsos.

O algoritmo mantém uma média móvel dos gradientes, chamada de primeiro momento (m) na equação (2.37), que decai exponencialmente,

$$m_{novo} = \beta_1 m_{antigo} + (1 - \beta_1)\frac{\partial J(x, y, w)}{\partial w_{ji}^k}, \tag{2.37}$$

em que β_1 é uma constante de valor `0,9` (KINGMA; BA, 2017).

Uma segunda média móvel do quadrado dos gradientes também é mantida, chamada de segundo momento (v) na equação (2.38), que também decai exponencialmente,

$$v_{novo} = \beta_2 v_{antigo} + (1 - \beta_2)\frac{\partial J(x, y, w)^2}{\partial w_{ji}^k}, \tag{2.38}$$

em que β_2 é uma constante de valor `0,999` (KINGMA; BA, 2017).

Os fatores de ajuste ($\hat{m}$ e $\hat{v}$) são calculados de acordo com a equação (2.39),

$$\begin{aligned} \hat{m} &= \frac{m}{1 - \beta_1} \\ \hat{v} &= \frac{v}{1 - \beta_2}. \end{aligned} \tag{2.39}$$

Finalmente, a atualização dos parâmetros pode ser dada pela equação (2.40),

$$w_{novo} = w_{antigo} - \frac{\eta}{\sqrt{\hat{v}} + \epsilon}\hat{m}, \tag{2.40}$$

em que ϵ é uma constante de suavização 10^{-8} para evitar divisões por zero (KINGMA; BA, 2017).

2.1.7 Operação de Convolução

A convolução é uma operação matemática que recebe duas funções de entrada e as transforma para gerar uma terceira função em sua saída (GOODFELLOW; BENGIO; COURVILLE, 2016). O processo de convolução é representada pelo operador asterisco (*), e pode ser realizada tanto de forma contínua quanto discreta.

A aplicação de filtros é um dos usos mais notáveis da operação de convolução, e podem ser empregadas tanto em funções de apenas uma dimensão, como em ondas de rádio-frequência, quanto em duas dimensões, como é o caso de imagens digitais. Esses filtros são capazes de atenuar ruídos e frequências indesejadas em sinais de rádio e de telecomunicações. De forma análoga, podem suavizar uma imagem, recorrendo ao efeito de desfoque para suavizar as altas frequências.

Considerando $f(x)$ e $g(x)$ como sinais discretos, a equação (2.41) define o processo da convolução discreta em uma dimensão,

$$(f * g)(x) = \sum_{n=-\infty}^{\infty} f(n)g(x-n). \tag{2.41}$$

Considerando duas matrizes de duas dimensões $f(x, y)$ e $g(x, y)$, a equação (2.42),

$$(f * g)(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m, n)g(x-m, y-n), \tag{2.42}$$

define a convolução discreta em duas dimensões.

Um filtro é muitas vezes referido como *kernel*, e geralmente de tamanho limitado e significativamente menor em comparação com o sinal ou a imagem que será submetido à convolução. Conforme é explicado em Goodfellow, Bengio e Courville (2016), a operação de convolução entre um *kernel* e uma imagem de entrada é realizada iterativamente por seções da imagem, cada uma com dimensões correspondentes às do *kernel*. Essa abordagem é aplicada de maneira deslizante e sucessiva sobre todos os pontos da imagem.

Embora a equação (2.42) descreva a convolução em duas dimensões, é comum em várias bibliotecas de processamento de imagens ou Aprendizagem Profunda a implementação da fórmula da correlação cruzada, conforme apresentada na equação (2.43),

$$(f * g)(x, y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(x+m, y+n)g(m, n). \tag{2.43}$$

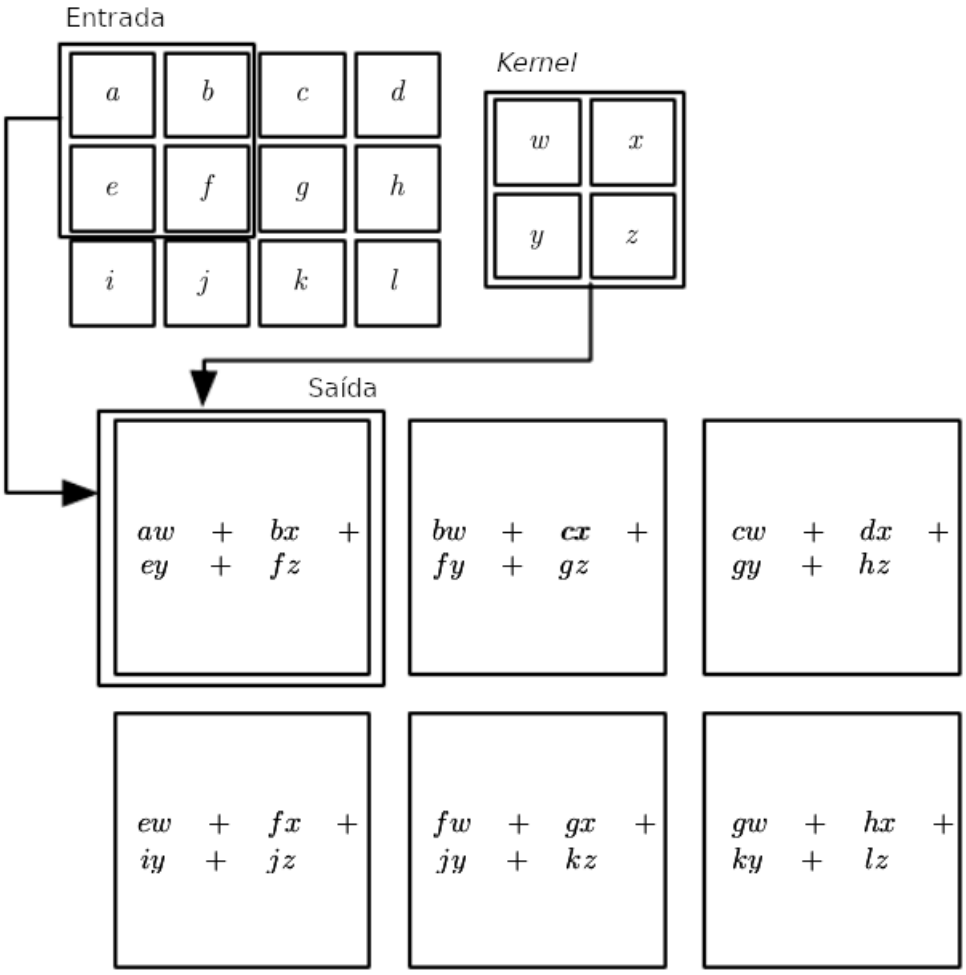
Isso ocorre devido à sua ausência de operações de subtração nos índices, e por que, na prática, o resultado é intercambiável.

Se a função $f(x, y)$ for vinculada a uma entrada $I(x, y)$, e a função $g(x, y)$ a um *kernel* $K(x, y)$ com tamanho conhecido (p x q), a equação (2.43) pode ser reescrita na equação (2.44),

$$(I * K)(x, y) = \sum_{m=0}^{p-1} \sum_{n=0}^{q-1} I(x+m, y+n)K(m, n). \tag{2.44}$$

A Figura 7 exemplifica o processo de convolução (ou mais precisamente correlação cruzada) de forma gráfica. A convolução entre a matriz de entrada e a matriz do *kernel* é realizada por meio de uma operação em que uma seção da matriz de entrada, de mesmo tamanho do *kernel*, é multiplicada elemento a elemento pelos correspondentes elementos do *kernel*. A soma dos resultados dessas multiplicações compõe a saída. A seção então desliza um índice para a direita, repetindo o processo até o final da linha. Posteriormente, a seção desliza um índice para baixo e percorre toda a linha inferior. Esse procedimento é repetido até que a seção tenha convoluído todos os elementos da matriz de entrada com o *kernel*.

Figura 7 – Exemplo de convolução em duas dimensões.



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

Na Aprendizagem Profunda, o *kernel* desempenha o papel do *perceptron*. Analogamente, cada elemento da matriz do *kernel* representa um peso associado a uma das entradas do *perceptron*. É importante lembrar que, além dos pesos presentes no *kernel*, o *perceptron* inclui um peso de *bias*, cuja entrada é sempre unitária. Cada *kernel* é treinado com o propósito de reconhecer e extrair padrões característicos dos dados de entrada.

Em implementações computacionais, existem dois parâmetros importantes que precisam ser mencionados nas funções de convolução: *padding* e *stride* (GOODFELLOW; BENGIO; COURVILLE, 2016). *Padding* se refere a linhas e colunas com valor zero que são adicionadas nas bordas da entrada. Isso é realizado com o propósito de assegurar que a saída da convolução mantenha dimensões idênticas às da entrada. *Stride* refere-se ao deslocamento do *kernel* durante a iteração do processo de convolução. Isto é, é possível fazer com que o deslocamento avance duas ou mais posições em vez de se deslocar para o espaço imediatamente à direta, mas ao custo de haver proporcionalmente uma redução do tamanho da saída.

2.1.8 Camada de Agrupamento

Após a etapa de convolução, a soma ponderada resultante é submetida à aplicação de uma função de ativação (descrita na Seção 2.1.4). Em seguida, a camada de agrupamento é empregada para realizar um processo semelhante ao de decimação, no qual uma pequena seção (geralmente retangular) de elementos tem o tamanho reduzido para fornecer uma representação mais compacta ou sumarizada dos dados (GOODFELLOW; BENGIO; COURVILLE, 2016). Essa operação é realizada por meio da aplicação de algoritmos específicos, como a seleção pelo elemento de maior valor (*max polling*), o cálculo da média estatística dos valores dos elementos dessa seção, a média ponderada baseada na distância em relação ao elemento central, entre muitos outros métodos.

O método de *max pooling* é um dos mais amplamente adotados na camada de agrupamento, sendo notável por sua simplicidade e eficácia na obtenção de resultados consistentes (YAMASHITA et al., 2018). A popularidade está na sua facilidade de implementação enquanto preserva as características proeminentes, reduzindo o tamanho da representação de forma eficiente. Este método ilustra como a simplicidade pode coexistir com a eficiência, proporcionando uma abordagem robusta para a redução de dimensionalidade nas tarefas de Aprendizagem Profunda.

A camada de agrupamento oferece algumas outras vantagens na análise de imagens em redes neurais, tais como: a atenuação de flutuações locais e a robustez contra variâncias. Ou seja, tenta-se preservar as características relevantes, descartando detalhes menos significativos, o que contribui para a generalização do modelo. Além disso, a camada de agrupamento pode reduzir distorções na escala, rotação ou posição dos objetos na imagem (GOODFELLOW; BENGIO; COURVILLE, 2016).

Na execução da camada de agrupamento é inevitável que ocorra a perda de informação espacial (DROZDZAL et al., 2016). Em aplicações de classificação isso pode ser considerado um efeito desejado pois o modelo se torna mais abrangente (YAMASHITA et al., 2018). Contudo, para redes destinadas à geração de imagens em sua saída, a omissão do índice do elemento de maior valor dentro de sua seção, e possivelmente a alteração desse valor, introduzem pequenos erros de imprecisão durante a fase de reconstrução (DROZDZAL et al., 2016). Essas imprecisões podem ter um impacto significativo na qualidade da predição final, especialmente em aplicações em que a fidelidade visual é um requisito essencial.

Tanto o processo de *stride* quanto a camada de agrupamento resultam na redução das dimensões da entrada. A distinção entre esses dois processos reside no fato de que, durante o *stride*, o índice não é perdido, mas não é possível analisar os valores individuais dos elementos na seção. Em contrapartida, na camada de agrupamento, a análise dos valores individuais dos elementos na seção é possível, mas ocorre a perda da informação

relacionada à posição desses elementos.

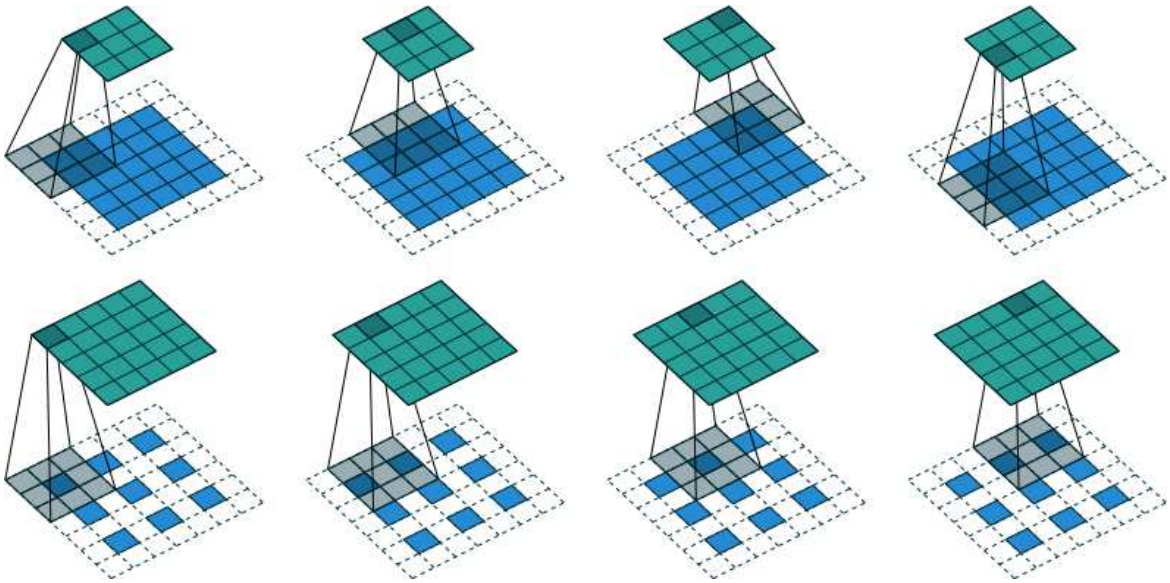
2.1.9 Convolução Transposta

Existem muitas formas de se produzir uma saída com dimensões maiores que as da entrada, tais como: vizinho mais próximo, interpolação bilinear, cama de pregos, entre outros. A convolução transposta é notável devido à sua implementação baseada em *perceptrons*, tornando-a capaz de aprendizado (NOH; HONG; HAN, 2015).

O processo de convolução transposta, também conhecido como *fractionally strided convolutions*, é uma técnica que pode ser empregada para aumentar a dimensionalidade de uma entrada (DUMOULIN; VISIN, 2016). Em outras palavras, busca-se obter uma saída com dimensões superiores às da entrada. Para compreender esse processo de forma mais fácil, a matriz de entrada é expandida pela inserção de valores zero em suas bordas e entre todos os seus elementos, seguida pelo o processo de convolução tradicional (conforme detalhado na Subseção 2.1.7.

A Figura 8 exemplifica de forma gráfica como uma operação de convolução tradicional (localizada na parte superior da figura) pode ser revertida através da convolução transposta (encontrada na parte inferior da figura). Na convolução, uma entrada (representada pelos quadrados azuis) tem tamanho (`5x5`) e é convoluída com um *kernel* (representado pelos quadrados hachurados) de tamanho (`3x3`), com *padding* de `1` e *stride* de `2`. Essa operação resulta em uma saída (representada pelos quadrados verdes) de tamanho (`3x3`).

Figura 8 – Exemplo de convolução e de sua convolução transposta equivalente.



Fonte: Dumoulin e Visin (2016).

Para reconstruir a entrada de tamanho (`5x5`) a partir da saída (`3x3`), é aplicado

o processo de convolução transposta. A saída (agora representada pelos quadrados azuis na parte inferior da Figura 8) de tamanho (`3x3`) é expandida, adicionando-se elementos de valor zero nos quadrados em branco com bordas descontínuas. Em seguida, ocorre a convolução convencional com um *kernel* (representado pelos quadrados hachurados) de mesmo tamanho (`3x3`), com *padding* de `1` e *stride* de `1`. O resultado (representado pelos quadrados verdes) tem o tamanho original de (`5x5`), como na entrada inicial.

É importante observar que os *kernels* usados na convolução e na convolução transposta, apesar de terem o mesmo tamanho, não necessariamente possuem os mesmos valores em seus elementos. Outros parâmetros também podem ser diferentes, nesse exemplo, o *padding* coincide, mas o *stride* não. Esses parâmetros devem ser considerados ao adicionar zeros para garantir que o resultado tenha as dimensões desejadas.

Apesar de este exemplo demonstrar que a convolução transposta é capaz de reverter um processo de convolução, é necessário enfatizar que, neste trabalho, a convolução transposta será utilizada para reverter o processo de redução de dimensionalidade provocada pelas camadas de agrupamento. Como mencionado anteriormente, considerando que a camada de agrupamento é análoga à decimação, a convolução transposta também pode ser comparada a uma operação de interpolação.

2.2 Redes Neurais Convolucionais

O período após a publicação do algoritmo de treinamento por retropropagação em 1974 até o ano de 1980 ficou conhecido como "Primeiro Inverno da IA" (TOOSI et al., 2021). Devido à limitação dos recursos computacionais disponíveis para lidar com a complexidade dos problemas limitou os avanços na área. O interesse em IA é reavivado apenas com a publicação da primeira arquitetura de Rede Neural Convolucional (CNNs, do inglês *Convolutional Neural Networks*) por Fukushima (1980).

Apesar dos investimentos realizados durante a década de 1980, muitas empresas não conseguiram entregar os resultados prometidos, pois o poder de processamento ainda não era suficiente. Isso desencadeou o "Segundo Inverno da IA" (TOOSI et al., 2021), que se iniciou em 1987 e se estendeu até a publicação do trabalho de Lecun et al. (1998). A partir desse momento, com a demonstração da eficácia das CNNs em tarefas de reconhecimento de dígitos manuscritos utilizando o conjunto de dados MNIST, as CNNs se tornaram uma das principais arquiteturas para tratar tarefas de reconhecimento de padrões em imagens (O'SHEA; NASH, 2015).

As CNNs têm nos *perceptrons* seu elemento constituente fundamental. Conforme já discutido em seções anteriores, cada *perceptron* possui pelo menos uma entrada sobre a qual realiza uma transformação. Geralmente, cada entrada é ponderada por um peso associado que multiplica o valor da entrada. Posteriormente, o produto de todas as entradas

é somado, e o resultado é submetido a uma função de ativação, conferindo-lhe a capacidade de lidar com problemas não lineares. Durante o processo de treinamento, os pesos são ajustados, e é por meio dessa calibração refinada que diversos padrões e estruturas de imagens podem ser aprendidos.

As CNNs são compostas por três tipos de camadas: camadas convolucionais, camadas de agrupamento e camadas totalmente conectadas. O empilhamento dessas camadas forma a arquitetura da CNN. O funcionamento inicia com a recepção da imagem de entrada pela camada convolucional, que a submete a diversos tipos de filtros (ou *kernels*) para extrair características importantes. Em seguida, a camada de agrupamento realiza a redução da dimensionalidade dos dados filtrados, preservando as informações essenciais. A camada totalmente conectada é a última a ser executada e tem a função de gerar as taxas de probabilidade a partir dos valores de ativação recebidos, utilizando-os para realizar a classificação. Durante o processo de treinamento, os pesos associados aos filtros são ajustados para otimizar o desempenho da rede, permitindo a aprendizagem de padrões e a capacidade de generalização para novos dados.

As camadas de convolução têm a função de realizar a filtragem de estruturas e padrões presentes na imagem de entrada. Isso é possível graças ao treinamento e aprendizado dos pesos associados aos filtros. Geralmente, em uma camada de convolução, existem diversos filtros organizados de forma paralela. Cada filtro tem tamanho reduzido, mas é capaz e especializado em detectar padrões específicos. A saída do filtro contém uma função de ativação, a mais utilizada é a ReLU, que é responsável por conferir a capacidade de trabalhar com dados não lineares.

Quando uma imagem é inserida na camada de convolução, o processo de convolução é executado por todos os filtros presentes na camada. Cada filtro gera um mapa de ativação correspondente, destacando as características relevantes que foram identificadas. É esse mecanismo que permite que as redes neurais convolucionais desempenhem tarefas como o reconhecimento de objetos e a classificação de imagens.

Os mapas de ativação são empilhados e transmitidos para a próxima camada de agrupamento. O principal objetivo dessa camada é reduzir o tamanho do mapa de ativação, visando diminuir o custo computacional. Além disso, esse processo contribui positivamente para a generalização do modelo. Conforme abordado na Subseção 2.1.8, diversos algoritmos são utilizados para efetuar a redução do tamanho da imagem, sendo o método de *max pooling* um dos mais populares. Essencialmente, esse método descarta informações menos relevantes, preservando os *pixels* de maior valor, destacados pela camada de convolução.

A primeira parte de uma CNN é constituída por diversas camadas de convolução intercaladas com camadas de agrupamento, empilhadas consecutivamente e aprofundando-se até alcançar a camada totalmente conectada.

A saída da última camada de agrupamento é linearizada antes de ser inserida na camada totalmente conectada. A camada totalmente conectada, também conhecida como "camada densa", é uma estrutura composta por vários conjuntos de *perceptrons*. Cada conjunto possui vários *perceptrons* dispostos em paralelo, e a saída de cada *perceptron* de um conjunto está conectada à entrada de todos os *perceptrons* do conjunto seguinte.

A estrutura e a operação matemática dos *perceptrons* da camada totalmente conectada são idênticas às da camada de convolução: as entradas são multiplicadas pelos pesos respectivos, seguido de uma função de ativação. No entanto, o propósito dessa camada é diferente, pois ela tem a função de aprender limiares de decisões complexos entre as classes.

Normalmente, o último conjunto de *perceptrons* da camada totalmente conectada utiliza a função de ativação *softmax*, em vez da ReLU. Isso é justificado, pois essa função de ativação produz valores proporcionais às exponenciais dos valores processados no último conjunto de *perceptrons*, ou seja, os valores produzidos pela função *softmax* podem ser diretamente interpretados como probabilidades de uma classe. Dessa forma, é importante observar que o número de saídas da camada totalmente conectada deve ser igual ao número de classes que se deseja separar. Além disso, é garantido pela função *softmax* que haverá sempre uma predição para uma das classes possíveis.

O modelo CNN generaliza as características extraídas pelas camadas de convolução, permitindo que a rede reconheça essas características de maneira independente. À medida que as camadas de convolução realizam a filtragem, padrões são identificados e destacados nos mapas de ativação. Com base nos padrões presentes nos mapas de ativação, a camada totalmente conectada realiza a predição. A Figura 9 foi adaptada e traduzida de Purwono et al. (2023) exibe um exemplo do diagrama de uma CNN com aplicação em classificação de veículos.

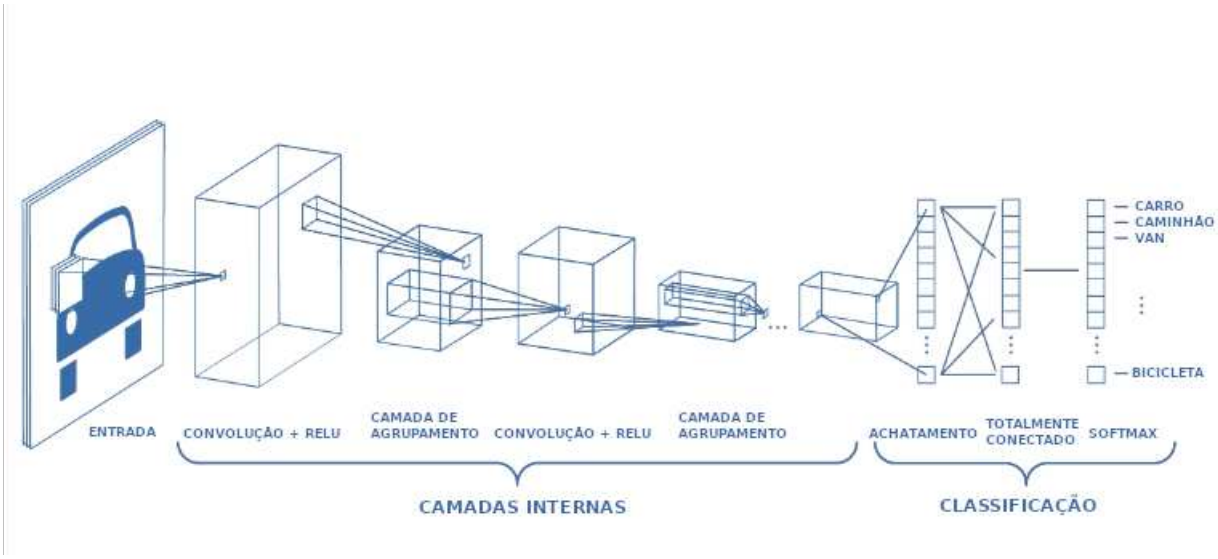
2.3 Segmentação Semântica e Modelo *U-Net*

O conceito de segmentação semântica define que, em uma imagem, cada *pixel* pertence a uma classe, e *pixels* de mesmo valor pertencem à mesma classe. Diversos modelos de redes neurais são capazes de realizar essa tarefa, e nesse contexto específico este trabalho propõe o estudo e análise de desempenho da rede *U-Net*.

O modelo *U-Net* foi inicialmente descrito em Ronneberger, P.Fischer e Brox (2015) e a arquitetura está exibida na Figura 10. Embora tenham sido concebidas originalmente para fins biomédicos, logo se percebeu que seriam adequadas para diversas outras aplicações de segmentação de objetos.

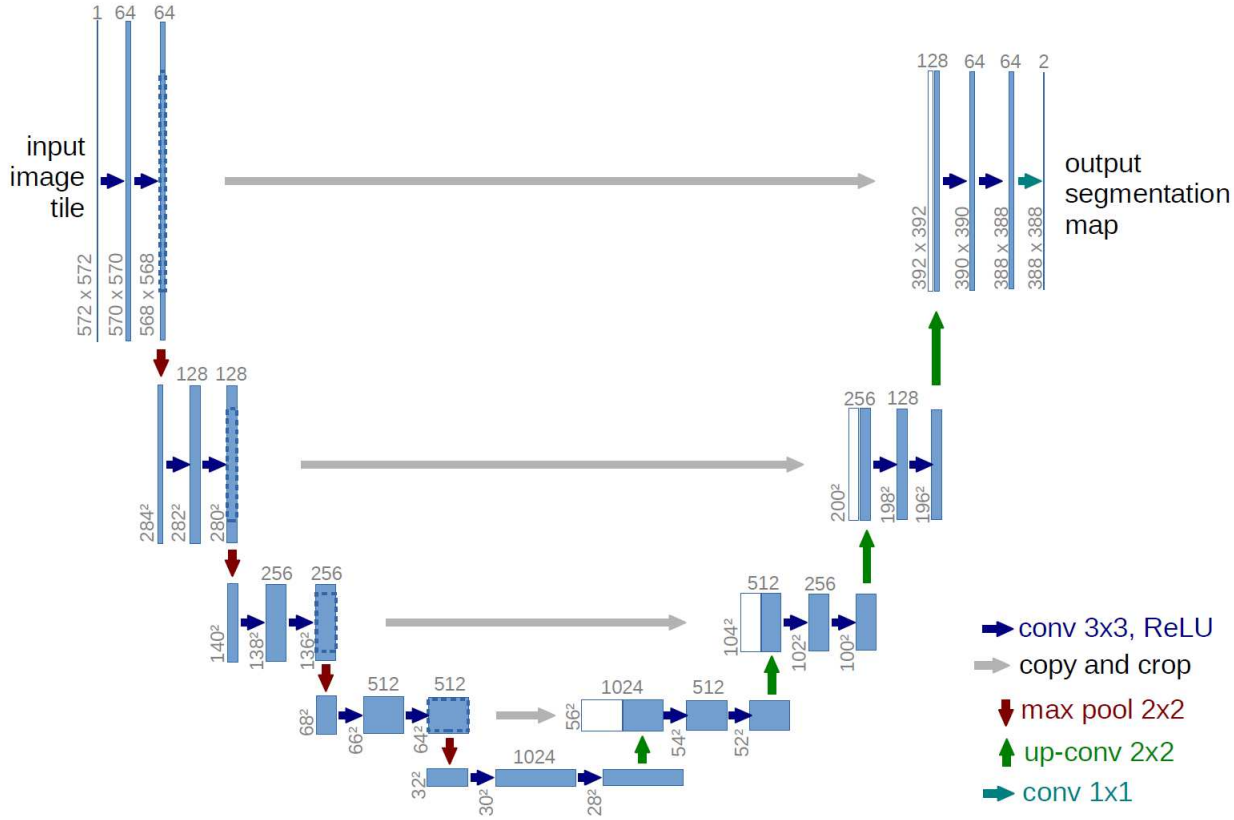
Uma *U-Net* convencional é dividida em duas partes: um codificador e um decodificador. O codificador tem a função de aprender a representação abstrata da imagem

Figura 9 – Arquitetura CNN.



Fonte: adaptado e traduzido de Purwono et al. (2023).

Figura 10 – Arquitetura *U-Net* original.



Fonte: Ronneberger, P.Fischer e Brox (2015).

de entrada para extrair suas características. É composto por vários blocos concatenados que realizam filtragem convolucional, seguida por uma função de ativação ReLU e uma camada de agrupamento que utiliza o algoritmo de *max pooling*. A saída da filtragem convolucional é chamada de mapa de características, e a camada de agrupamento reduz as dimensões desses mapas quando as encaminha para as camadas mais profundas.

Entre o codificador e o decodificador, há uma ponte que conecta ambas as partes. Na verdade, ela se assemelha muito a qualquer bloco do codificador, exceto pelo fato de não ter uma camada de agrupamento ao final, mas sim uma camada de convolução transposta.

O decodificador recebe a representação abstrata do codificador e gera uma máscara de segmentação. Ele é praticamente um espelho do codificador, composto pelo mesmo número de blocos concatenados que realizam uma convolução transposta e filtragem convolucional. Devido ao processo de convolução transposta, o tamanho de entrada desse bloco é aumentado na saída.

As *U-Nets* superam a perda de informação espacial durante a etapa de codificação usando *skip connections*, ou seja, transferindo os mapas de características de entrada do caminho de compressão para o caminho de expansão, embora isso acarrete na necessidade de muitos mais parâmetros. Devido a isso, a estrutura é apresentada em formato de U, o que também empresta seu nome ao modelo.

A saída do último decodificador passa por uma convolução com ativação sigmoidal. A função de ativação sigmoidal gera a máscara de segmentação representando a classificação por *pixels*.

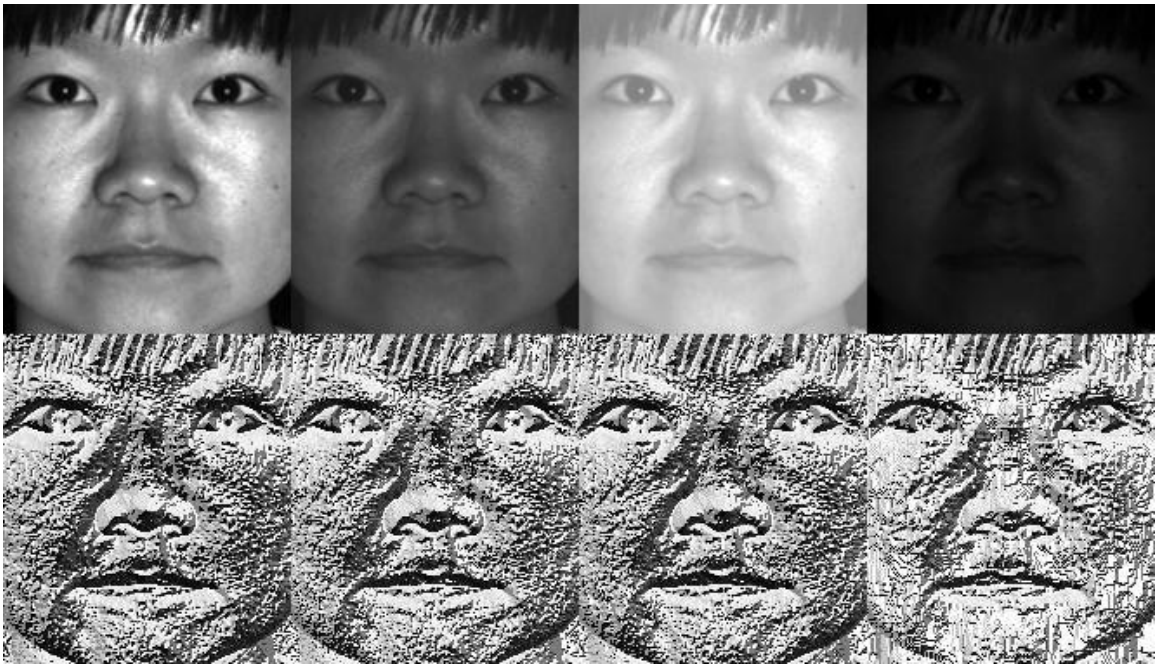
2.4 Descritor de Texturas *Local Binary Pattern*

A concepção do algoritmo *Local Binary Pattern* (LBP) foi proposta em Ojala, Pietikäinen e Harwood (1996), com o objetivo de extrair as características locais de texturas das imagens. Através de um operador matemático de baixa complexidade computacional, porém de alta sensibilidade aos detalhes, processa a imagem gerando um mapa de texturas como resultado.

O trabalho original ressalta vantagens como invariância em relação à iluminação. Por exemplo, a Figura 11, apresentada em Aizan, Ezin e Motamed (2016), ilustra como uma mesma imagem sob diferentes condições de luminosidade gera mapas de textura LBP muito semelhantes. Além disso, as descontinuidades na imagem provocadas por mudanças abruptas de contraste, podem ser usadas para detectar as bordas em diferentes objetos.

O operador LBP é representado por um *patch*, uma janela quadrada de tamanho variável N (normalmente `9` ou `3x3`), responsável por processar cada *pixel* na imagem. A

Figura 11 – Efeito da variação na iluminação em uma imagem rasterizada na saída do operador LBP



Fonte: Aizan, Ezin e Motamed (2016).

característica LBP associada ao *pixel* na posição (x_c, y_c) é expressa pela equação (2.45),

$$LBP_{(x_c,y_c)} = \sum_{n=0}^{N-2} s_{(i_n,i_c)} 2^n. \tag{2.45}$$

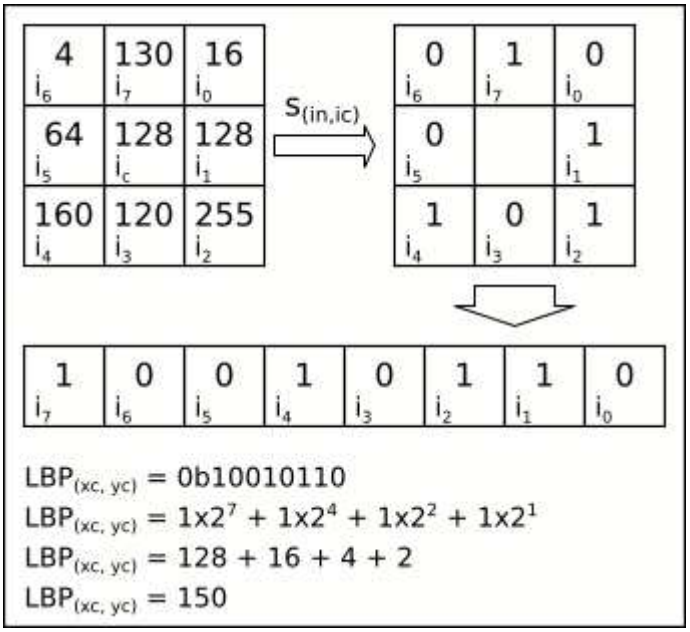
Cada *pixel* assume, por sua vez, a posição central (x_c, y_c) do descritor LBP e é cercado por $N-1$ *pixels* vizinhos, indicados por i_0, i_1, ..., i_{L-1}. O *pixel* central, conhecido como pivô, tem seu valor i_c comparado com o de cada vizinho i_n por meio da função $s_{(i_n,i_c)}$, que é definida pela equação (2.46),

$$s_{(i_n,i_c)} = \begin{cases} 1 \text{ , se } i_n \geqslant i_c \\ 0 \text{ , caso contrário} \end{cases} . \tag{2.46}$$

Quando a diferença entre um vizinho e o *pixel* central é não negativa, atribui-se o valor `1`; caso contrário, atribui-se o valor `0`. Todos esses valores de comparação são ordenados para formar um número binário, representando a característica atribuída ao *pixel* central. A Figura 12 ilustra um exemplo da codificação LBP para um operador de tamanho `3×3`.

A partir do processamento do operador LBP sobre uma imagem rasterizada, é gerado um mapa de texturas LBP. Originalmente, as aplicações com o operador LBP visavam realizar a classificação e identificação do tipo do material retratado na imagem

Figura 12 – Exemplo do processamento LBP usando uma janela de tamanho 3x3.



Fonte: De autoria própria.

a partir do histograma do mapa de texturas LBP. Ou seja, cada material deve gerar um histograma com formato característico. A Figura 13 exemplifica como é o histograma de algumas amostras de diferentes materiais obtidas de (MALLIKARJUNA et al., 2006).

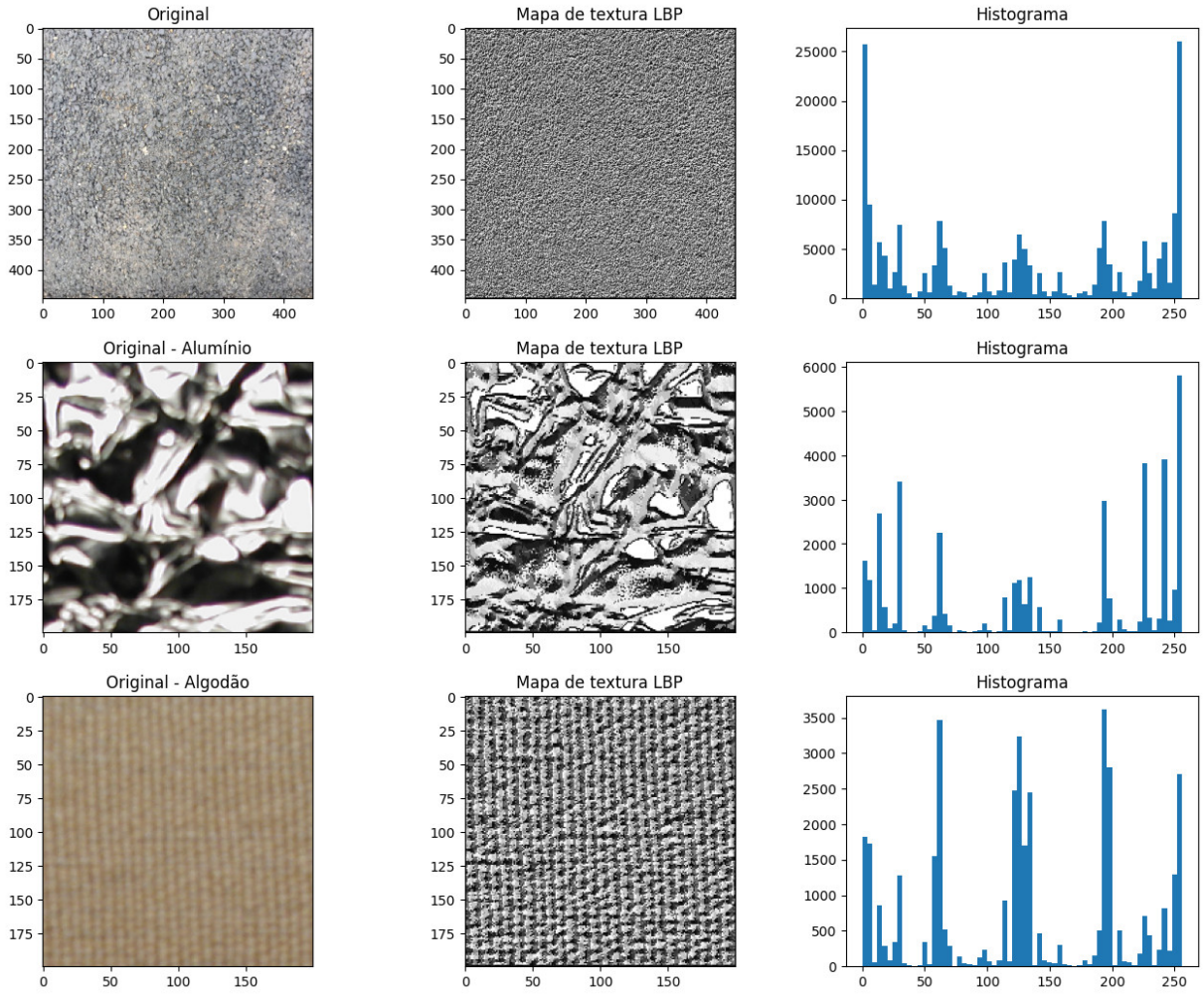
A invariância à rotação pode ser explicada pela transformação do mapa LBP em um histograma em aplicações de classificação de textura, pois esse processo elimina a informação espacial dos *pixels*. No entanto, a invariância aos tons de cinza é uma característica relevante quando se deseja minimizar a interferência causada pela luminosidade. Isso é particularmente importante para realçar os padrões de texturas da pavimentação asfáltica, permitindo que o mapa de texturas LBP produza resultados consistentes, independentemente das condições de iluminação.

2.4.1 Biblioteca *scikit-image*

A *scikit-image* é uma biblioteca de código aberto para processamento de imagens desenvolvida para *Python* (PEDREGOSA et al., 2011). A função `local_binary_pattern()` é utilizada para produzir um mapa de texturas LBP. No *website* que documenta essa função, são levantadas algumas considerações interessantes acerca de informações que podem ser extraídas e interpretadas do mapa de texturas.

A Figura 14 apresenta exemplos de resultados nos quais os *pixels* de cor preta ou branca tem intensidade menor ou maior que o *pixel* central. Quando os *pixels* circundantes são todos pretos ou brancos, a região da imagem é considerada plana, ou seja, sem características distintivas. Grupos contínuos de *pixels* pretos ou brancos são identificados

Figura 13 – Histogramas gerados por diferentes texturas de materiais.



Fonte: Montagem a partir de algumas amostras da base de dados Mallikarjuna et al. (2006).

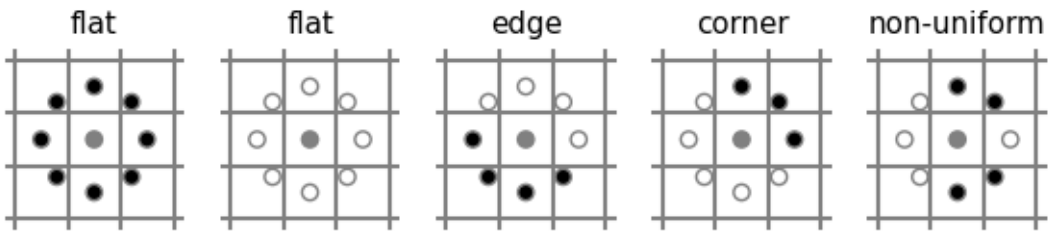
como padrões uniformes, os quais podem ser interpretados como cantos ou bordas. Se os *pixels* alternam entre preto e branco, o padrão é classificado como "não uniforme".

Na Figura 15 são apresentadas seis imagens dispostas em duas linhas e três colunas. Na coluna da esquerda, a imagem de cima destaca em vermelho os *pixels* com padrões de bordas, enquanto a imagem de baixo, também destaca em vermelho, as barras que pertencem aos *pixels* que compõem as bordas no respectivo histograma. De forma análoga, a coluna do meio destaca as superfícies planas e a da direita, os cantos.

2.5 Considerações Parciais

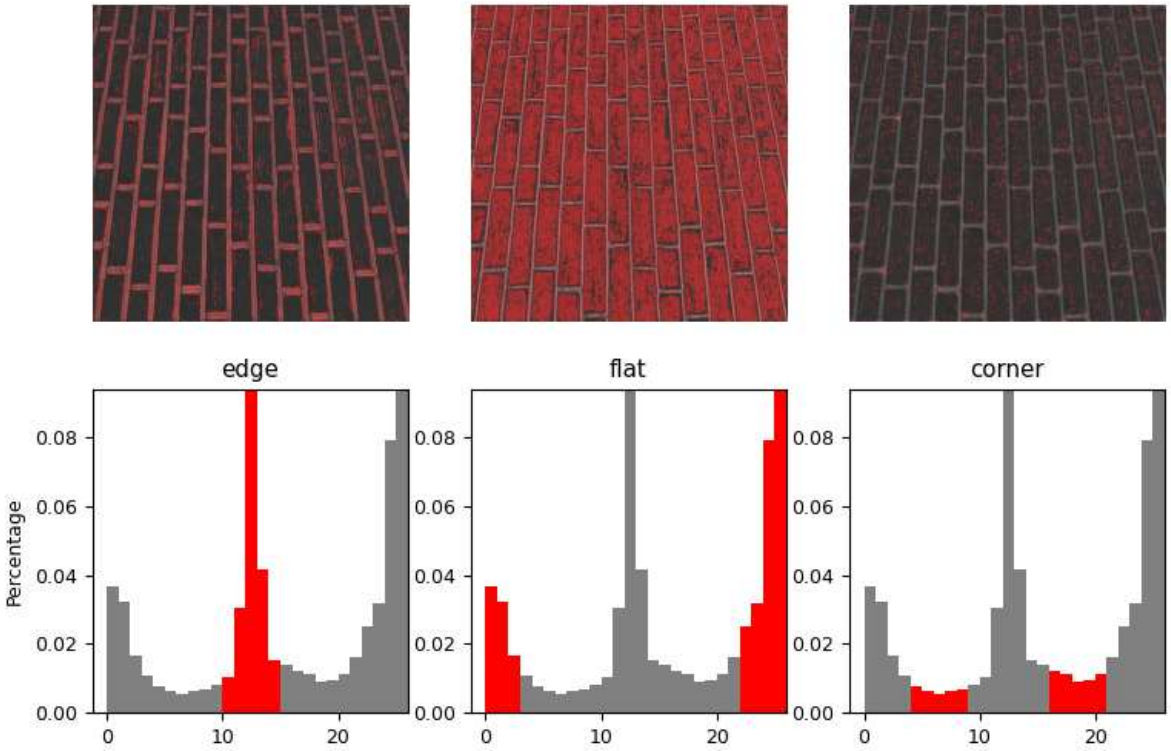
Os conceitos de Aprendizagem Profunda apresentados neste capítulo forneceram uma base para a compreensão e desenvolvimento de sistemas de IA. Ao explorar as redes neurais artificiais e seus diversos componentes, este capítulo proporciona algumas fer-

Figura 14 – Estruturas e variação da tonalidade de *pixels*.



Fonte: Pedregosa et al. (2011).

Figura 15 – Exemplo de estruturas em um histograma.



Fonte: (PEDREGOSA et al., 2011).

ramentas para tomada de decisões informadas na criação de projetos inovadores nessa área.

O *perceptron*, as operações de convolução, da cama de agrupamento e da convolução transposta constituem blocos fundamentais na construção de redes neurais. A flexibilidade na combinação e modificação desses módulos permite a criação de arquiteturas personalizadas, que podem revelar melhores resultados no reconhecimento de padrões.

O descritor de texturas LBP, originalmente desenvolvido para análise de texturas, foi aqui empregado como uma técnica para realçar as características visuais mais relevantes em imagens. Ao fornecer uma representação mais discriminativa dos dados, os mapas de texturas LBP contribuem para melhorar a performance de redes neurais em alguns casos.

O próximo capítulo apresentará alguns estudos relacionados ao tema deste trabalho. Na primeira parte, serão citados artigos que adotaram modelos distintos para enfrentar o problema de segmentação semântica de pavimentações asfálticas. Na segunda parte, serão analisados dois artigos que justificam o uso do mapa de texturas LBP no treinamento de redes neurais em aplicações diversas à segmentação semântica.

3 ESTADO DA ARTE E TRABALHOS RELACIONADOS

Este capítulo explora duas áreas de pesquisa: segmentação semântica de pavimentação asfáltica e utilização de mapas de textura LBP para treinamento de redes neurais. No primeiro tema, são citadas brevemente algumas soluções tradicionais baseadas em gráficos, e logo em seguida, são apresentadas algumas publicações que usam métodos mais recentes de Aprendizagem Profunda. Diferentes arquiteturas e configurações são descritos, permitindo observar como essas variações impactam na precisão modelos.

No segundo tema, cita-se dois artigos que utilizam o mapa de texturas LBP no treinamento de redes neurais, e declaram um aumento na precisão dos modelos em comparação com o método convencional. É relevante ressaltar que os trabalhos contidos no segundo tema não tem aplicação em segmentação semântica.

3.1 Segmentação de Ruas Baseada em Aprendizagem Profunda

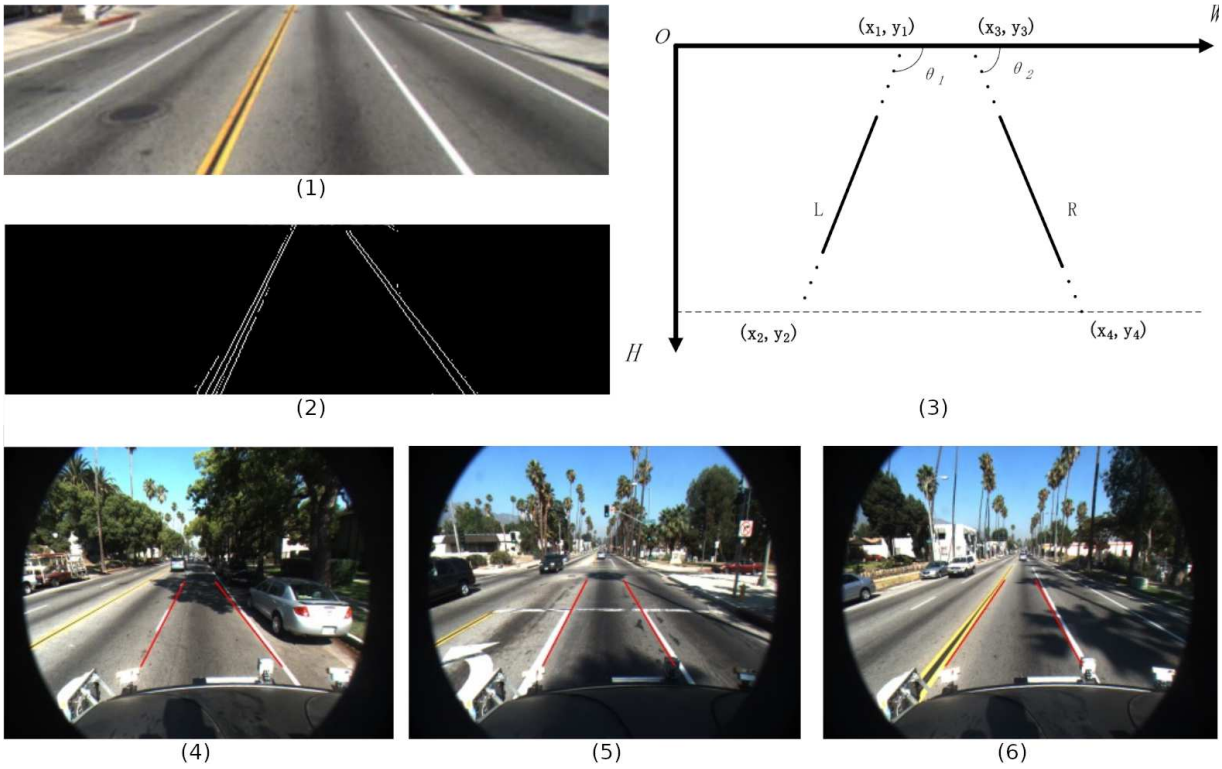
A segmentação da imagem de ruas em imagens rasterizadas não é um problema recente e, portanto, pode-se encontrar publicações que propõe soluções baseadas em métodos que não estão relacionados à área de Aprendizagem de Máquina. A exemplo, o trabalho em Zhang e Nagel (1994) propõe uma solução baseada na intensidade da textura determinada pelas mudanças dos valores dos *pixels* em uma matriz de covariância.

Outras referências como em Li-Yong et al. (2018), em Phueakjeen et al. (2011) e em Nie et al. (2019), adotam soluções gráficas e geométricas. Através da combinação um método de filtragem de borda e em seguida com a aplicação da Transformada de Hough, é possível evidenciar as bordas das faixas de trânsito, conforme mostrado na Figura 16. Uma desvantagem dessa abordagem é a aplicação em cenários cujas faixas estão ausentes ou desgastadas pelo uso e exposição às condições climáticas.

Adentrando no campo da Aprendizagem Profunda, os autores Yousri, Elattar e Darweesh (2021) propõem um padrão de comparação para ser utilizado como referência em segmentação de ruas. O trabalho reúne cinco modelos diferentes: *ResUNet*, *ResUNet++*, *U-Net*, *Modified SegNet*, e *SegNet*. O termo *ResUNet* deriva de *Residual U-Net*, sendo um modelo inspirado na arquitetura *U-Net*, mas que combina técnicas como conexões residuais, convoluções dilatadas, *Spatial Pyramid Pooling* (SSP), e inferência multi-tarefa.

As conexões residuais representam uma forma específica de *skip connections*, em que a entrada do bloco é adicionada à saída, geralmente após um processo de convolução. As convoluções dilatadas expandem o *kernel* tradicional possibilitando um campo receptivo maior sem aumentar a complexidade computacional da rede. A técnica SSP aplica a camada de agrupamento com diferentes tamanhos, gerando saídas com diferentes esca-

Figura 16 – (1) Imagem original. (2) Detecção de borda. (3) Diagrama do modelo para detecção das linhas das faixas. (4)-(6) Resultados.



Fonte: montagem de Li-Yong et al. (2018).

las de regiões de interesse. A inferência multi-tarefa pode envolver, além da segmentação semântica, a predição de outras informações relacionadas, como a detecção de objetos.

A *ResUNet++* foi a que apresentou os melhores resultados. A arquitetura incorpora blocos residuais mais elaborados em relação à *ResUNet*, nos quais as conexões residuais são estabelecidas entre todas as camadas subsequentes, aumentando a conectividade e a troca de informações.

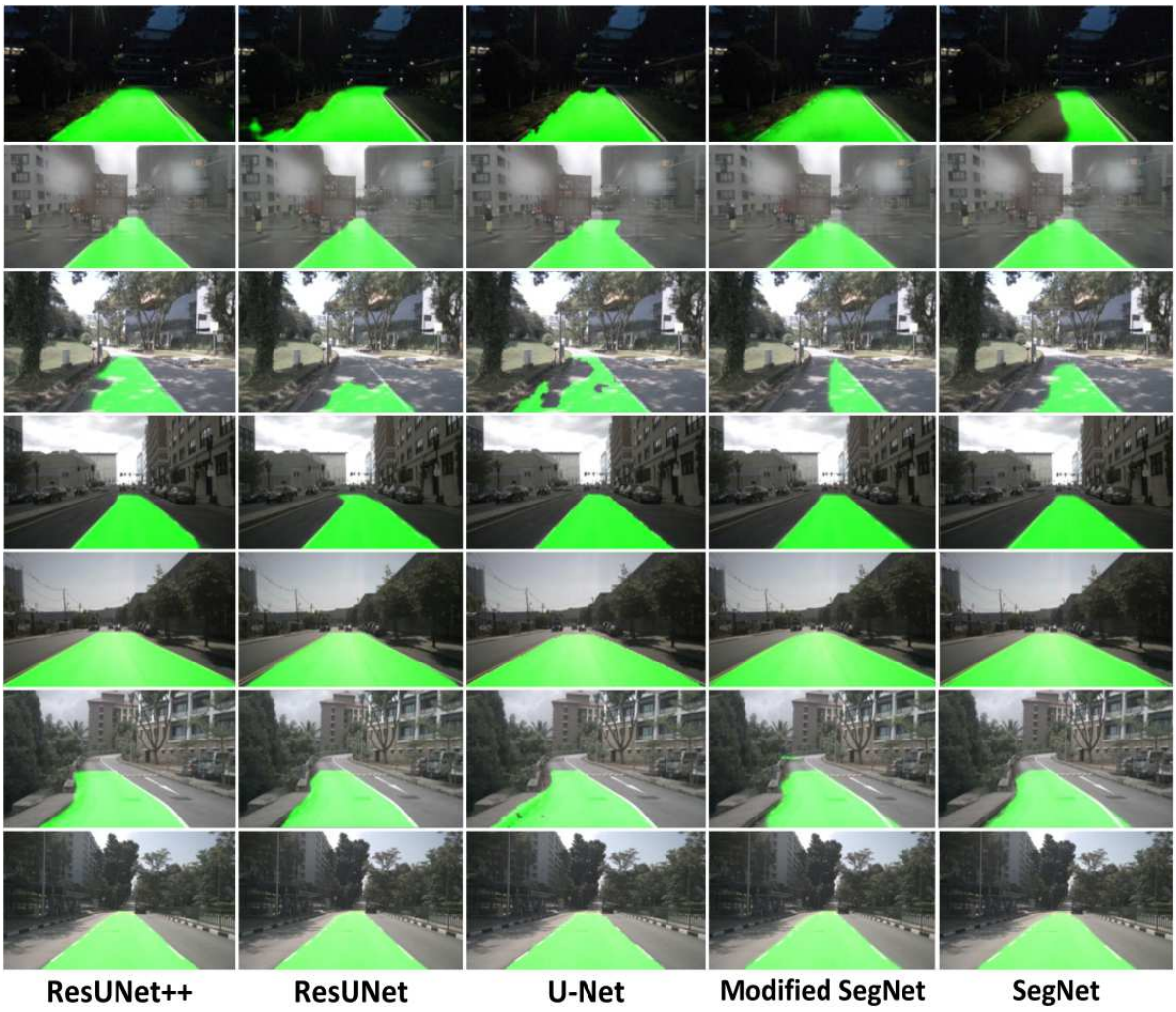
O modelo *SegNet* utiliza mapas de índices gerados durante o processo de codificação para guiar o processo de decodificação. Isso reduz a carga de processamento em comparação com o modelo *U-Net*, já que os mapas de índices armazenam as posições dos valores máximos, sem a necessidade de realizar o processo de convolução transposta.

Todas essas redes são treinadas para realizar a segmentação semântica de pavimentação asfáltica em cenários diferentes: noite, chuva, com sombras, nublado, dia com faixas amarelas, faixas curvas e faixas descontínuas, conforme mostra a Figura 17.

O trabalho em Jebamikyous e Kashef (2021) apresenta quatro variantes do modelo *U-Net* e os resultados obtidos são comparados com os de outros três modelos FCN-16, FCN-8 e *SegNet*.

A *Fully Convolutional Network* (FCN) (LONG; SHELHAMER; DARRELL, 2015)

Figura 17 – Resultados da segmentação semântica. Cada linha representa uma condição de teste específica, organizadas de cima para baixo: noite, chuva, com sombras, nublado, dia com faixas amarelas, faixas curvas e faixas descontínuas.



Fonte: Yousri, Elattar e Darweesh (2021).

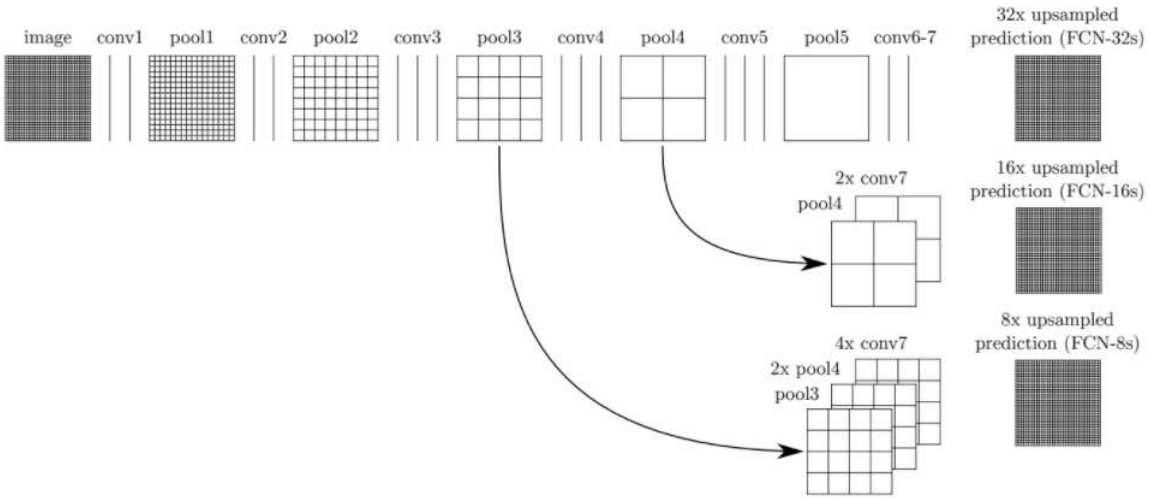
é um modelo cuja estrutura é muito semelhante à de uma CNN tradicional. De fato, a arquitetura da FCN foi inspirada no modelo VGG16 (SIMONYAN; ZISSERMAN, 2014), uma CNN reconhecida no meio acadêmico por sua simplicidade estrutural e capacidade de classificação de imagens em `1000` categorias distintas. A FCN difere na substituição da camada totalmente conectada por camadas convolucionais que possuem um número apropriado (e geralmente grande) de filtros para que se possa preservar a informação espacial, e gerar uma saída com as dimensões equivalentes às da entrada inicial.

A FCN-32 executa o processo de convolução e agrupamento por cinco vezes consecutivas, reduzindo as dimensões espaciais da saída em `32` vezes em relação à entrada original. Após a última camada de agrupamento, ocorre uma convolução com filtros de tamanho `1x1` em um número geralmente grande de camadas, seguida por um único processo de convolução transposta. A convolução transposta restaura as dimensões originais,

multiplicando a saída da última camada de agrupamento por 32, gerando a máscara de segmentação semântica.

As FCN-16 e FCN-8 modificam a FCN-32 adicionando *skip connections* e convoluções transpostas para ajuste das dimensões em diferentes estágios, conforme o esquema da Figura 18.

Figura 18 – Arquitetura da FCN (FCN-32, FCN-16, FCN-8).



Fonte: Jebamikyous e Kashef (2021) e Long, Shelhamer e Darrell (2015).

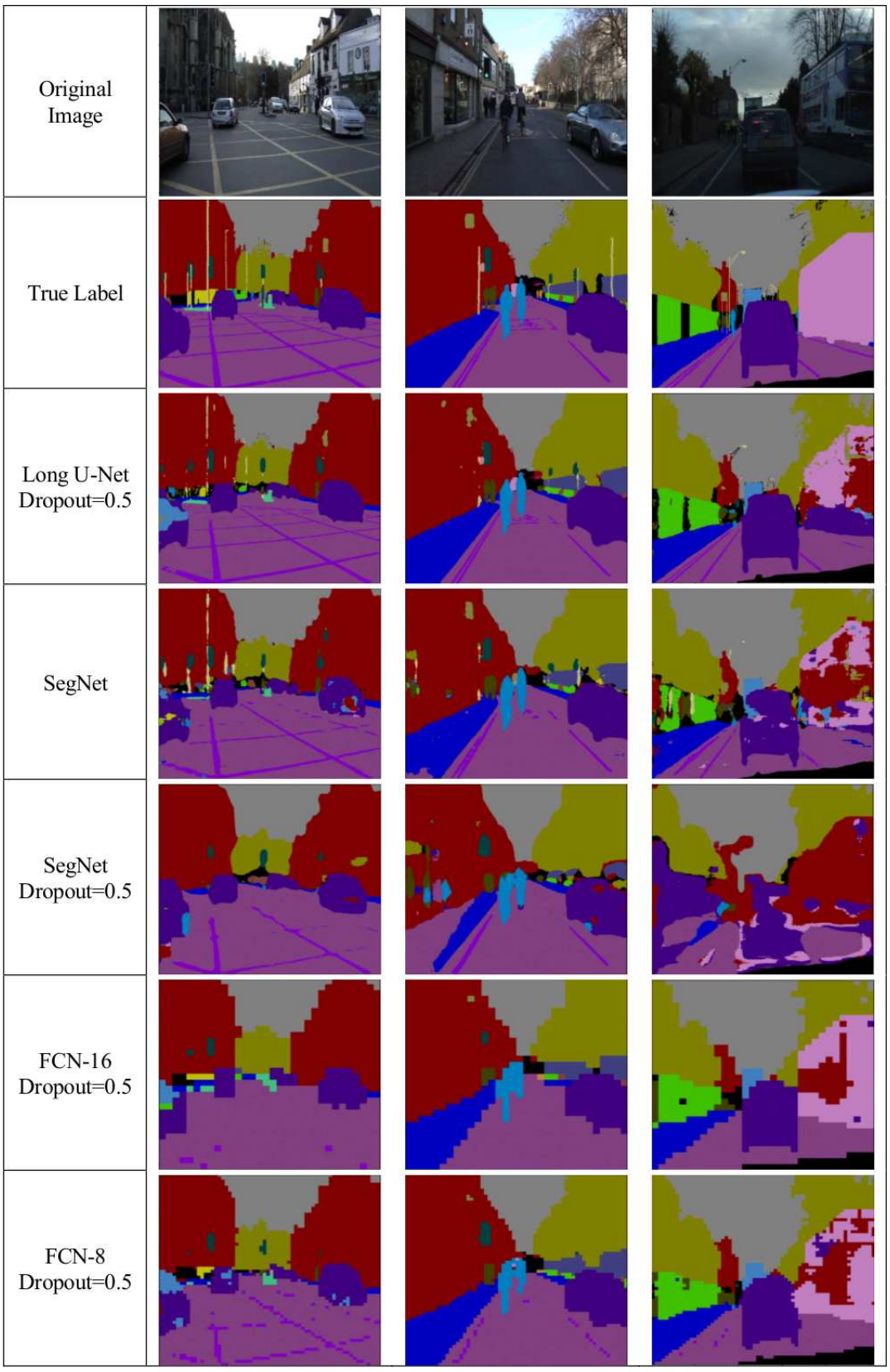
As *U-Nets* foram modificadas reduzindo ou aumentando o número de camadas convolucionais, modificando a função de ativação para *LeakyReLu*, ou adicionando uma técnica de regularização *Dropout*.

O treinamento foi realizado com imagens da base de dados *CamVid* que traz 32 classes diferentes. O melhor resultado foi dado pela *U-Net* com mais camadas e *Dropout* de 0,7. A Figura 19 exibe os resultados da segmentação semântica publicada pelo estudo de Jebamikyous e Kashef (2021).

Os autores em Sarmah, Gogoi e Kalita (2022) conduziram um estudo comparativo envolvendo cinco variantes da arquitetura *U-Net* com modificações pontuais. O propósito era investigar possíveis vantagens da técnica de Transferência de Aprendizagem. Para isso, o primeiro modelo, considerado como controle, manteve a estrutura da *U-Net* original. Os quatro modelos restantes tiveram a parte do codificador substituída: dois deles utilizando o modelo VGG16 (SIMONYAN; ZISSERMAN, 2014) e os outros dois o modelo VGG19 (SIMONYAN; ZISSERMAN, 2014). Em cada par, um modelo empregou pesos pré-treinados, enquanto o outro não.

A base de dados *CamVid* foi utilizada para treinar os modelos. A versão modificada da *U-Net* com o codificador VGG16, contendo pesos pré-treinados, apresentou uma taxa de acurácia um pouco maior em comparação com o modelo padrão da *U-Net*. A Figura 20

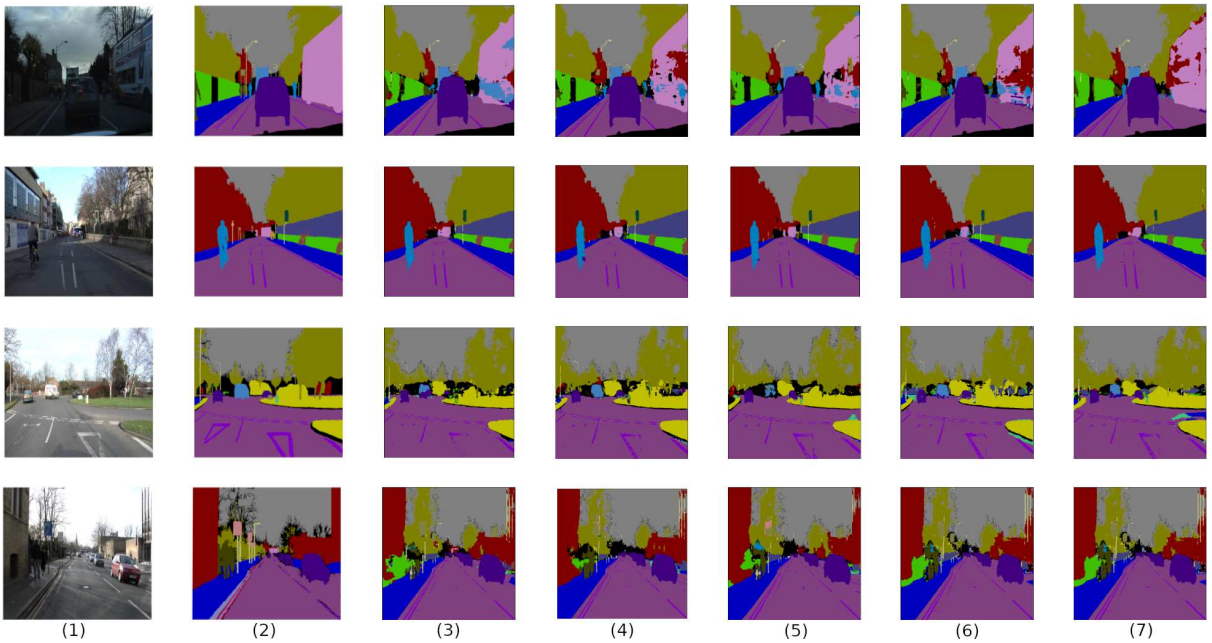
Figura 19 – Resultados da segmentação semântica.



Fonte: Jebamikyous e Kashef (2021).

exibe alguns resultados de segmentação semântica publicados em Sarmah, Gogoi e Kalita (2022).

Figura 20 – (1) Imagem original. (2) Verdade de referência. (3) Modelo *U-Net* sem modificações. (4) Codificador VGG16 pré-treinado com decodificador *U-Net*. (5) Codificador VGG16 com decodificador *U-Net*. (6) Codificador VGG19 pré-treinado com decodificador *U-Net*. (7) Codificador VGG19 com decodificador *U-Net*.



Fonte: Sarmah, Gogoi e Kalita (2022).

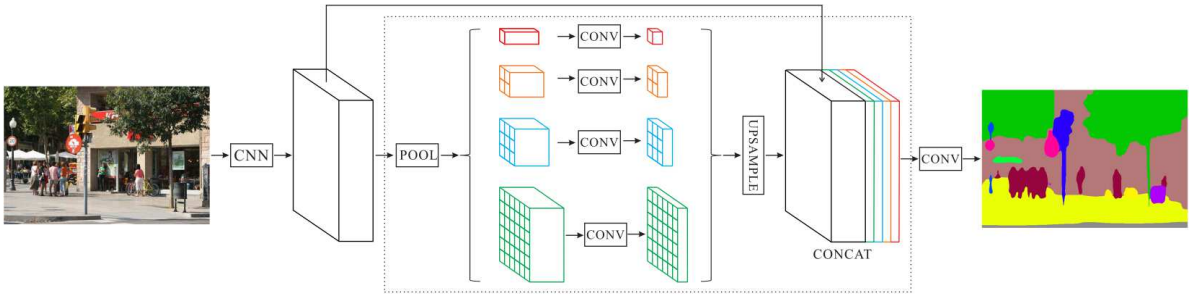
Em Pham (2021) é feia uma análise comparativa entre três modelos: *PSPNet*, FCN e *SegNet*. A proposta do modelo *PSPNet* foi apresentada por Zhao et al. (2017). Assim como a maioria dos modelos de segmentação semântica, o *PSPNet* é composto por duas partes principais: um codificador e um decodificador. O codificador é essencialmente composto por camadas convolucionais, normalização por lote, ativação ReLU e camada de agrupamento. O decodificador, por sua vez, possui todos os elementos do codificador, exceto pela camada de agrupamento que é substituída pela convolução transposta.

O componente central do *PSPNet* é o módulo da pirâmide de agrupamento, onde a saída do codificador é submetida a agrupamentos em diferentes escalas para evitar a perda de características distintivas. As saídas do módulo da pirâmide de agrupamento têm tamanhos diferentes e são ajustadas por convoluções transpostas para ajuste das dimensões.

No decodificador, é necessário aumentar as dimensões em oito vezes, uma etapa realizada de maneira semelhante à abordagem da *U-Net*. Nesse processo, as camadas dos blocos de codificação são concatenadas nos blocos de decodificação, uma prática eficaz para recuperar a informação espacial. A Figura 21 retrata a estrutura da *PSPNet*, e a

Figura 22 exibe alguns regulstados da segmentação semântica realizada em Pham (2021).

Figura 21 – Estrutura da *PSPNet*.



Fonte: Pham (2021) e Zhao et al. (2017).

O objetivo em Pham (2021) é o de encontrar o melhor modelo para ser utilizado em sistemas embarcados. Dessa forma, além da acurácia, o tempo de processamento também é relevante. No treinamento, a base de dados *Cityscapes* foi utilizada e 19 classes diferentes foram consideradas, resultando no modelo *PSPNet* apresentar a melhor taxa de acerto, porém o tempo de processamento tomado pelo modelo FCN é cerca de oito vezes menor.

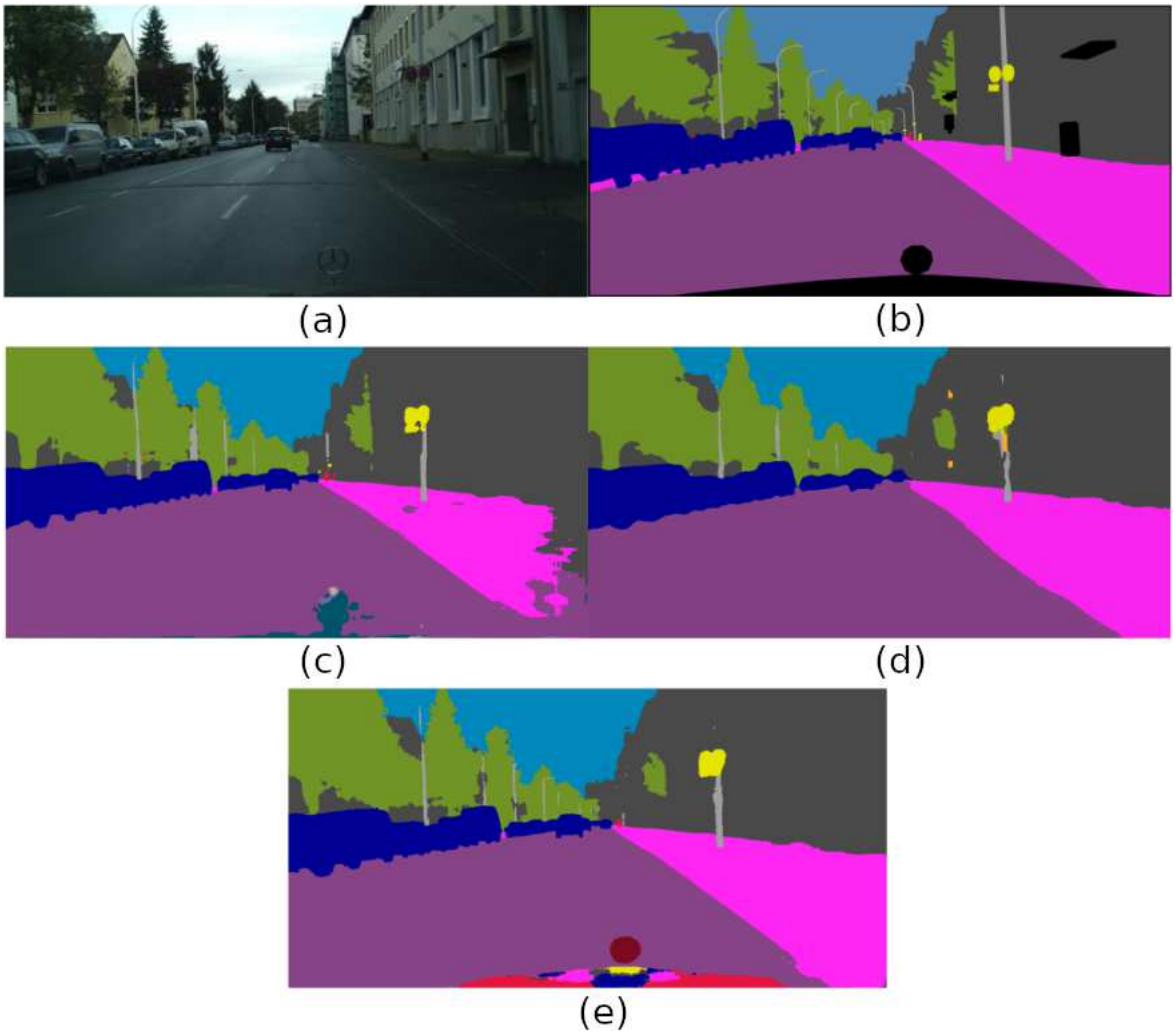
Por fim, em Lazuardi et al. (2019) cinco modelos *U-Net* são implementados com a finalidade de serem integrados em um sistema com um núcleo *Raspberry Pi 3B+*. Devido ao poder de processamento limitado do sistema, o número de parâmetros é reduzido através da diminuição do número de filtros, para que a carga computacional seja a menor possível, e a saída é limitada para identificar até cinco classes diferentes de objetos. Os cinco modelos de rede neurais começam com 7,7 milhões de parâmetros que vão sendo gradualmente reduzidos para 1,9 milhões, 700 mil, 300 mil até 100 mil parâmetros.

O artigo reporta que a menor rede de 100 mil parâmetros tem a performance bastante degradada na segmentação de pessoas, por isso o modelo com 300 mil parâmetros é preferido.

A etapa de treinamento utilizou a base de dados *Cityscapes* e precisou ser realizada em um ambiente com mais recursos, diferente do ambiente de implantação. Apesar das limitações, é reportado que o sistema entrega uma boa performance com taxa de acurácia maior que 90%.

A Tabela 1 sintetiza as especificações mais importantes dos trabalhos relacionados.

Figura 22 – (a) Imagem original. (b) Verdade de referência. (c) Resultado *PSPNet*. (d) Resultado FCN. (e) Resultado *SegNet*.

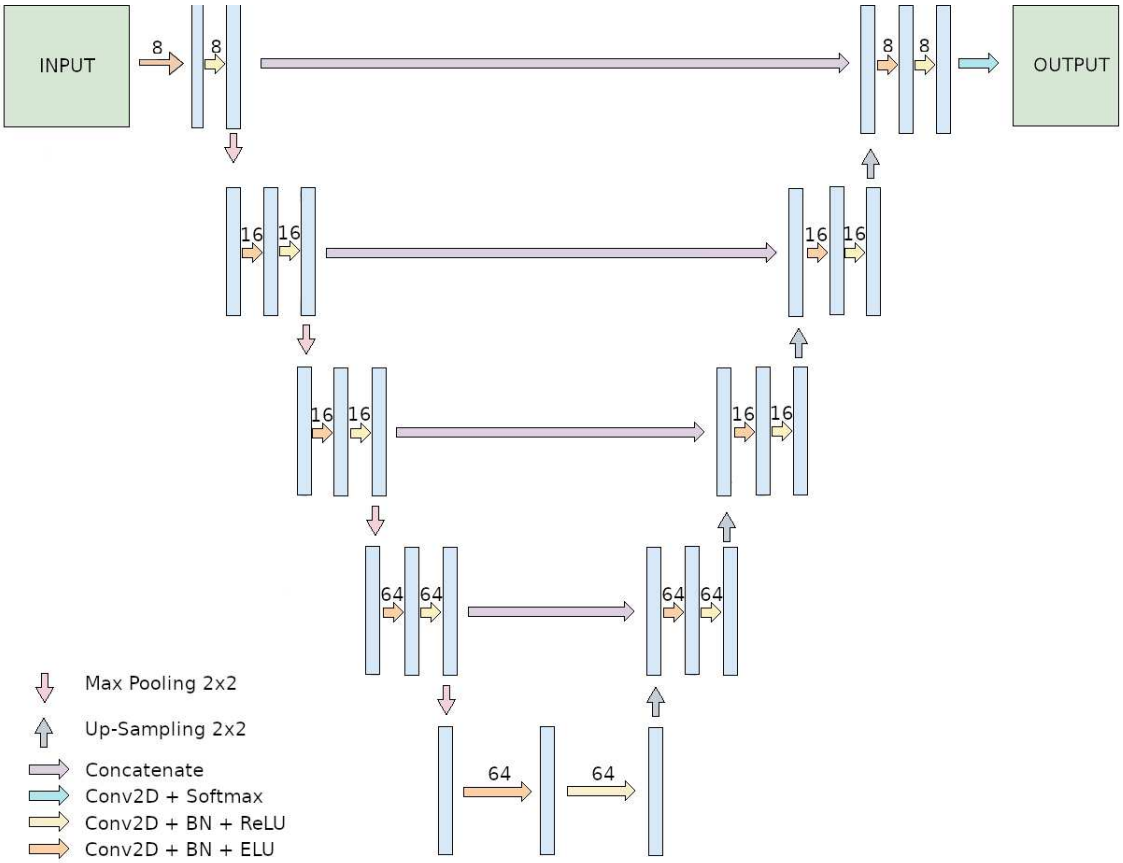


Fonte: Pham (2021).

Tabela 1 – Relação das especificações dos trabalhos relacionados

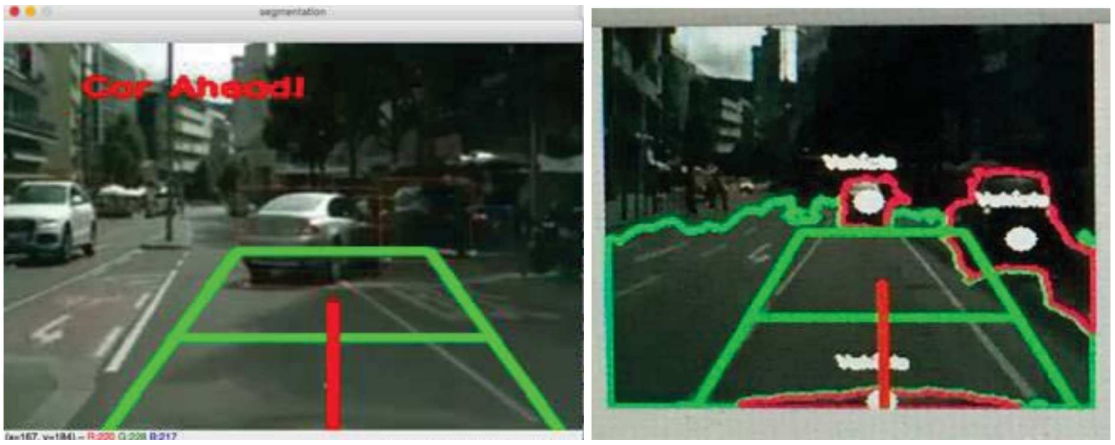
Referência	**Modelos**	**Base de Dados**	**Classes**
Yousri, Elattar e Darweesh (2021)	*ResUNet++*, *ResUnet*, *U-Net*, *Modified SegNet* e *SegNet*	*nuScenes*	2
Jebamikyous e Kashef (2021)	FCN-16, FCN-8 e *SegNet*	*CamVid*	32
Sarmah, Gogoi e Kalita (2022)	VGG16 e VGG19	*CamVid*	32
Pham (2021)	*PSPNet*, FCN e *SegNet*	*Cityscapes*	19
Lazuardi et al. (2019)	*U-Net*	*Cityscapes*	5

Figura 23 – Arquitetura *U-Net* para *Raspberry Pi 3B+*.



Fonte: Lazuardi et al. (2019).

Figura 24 – *U-Net* para *Raspberry Pi* 3B+.



Fonte: Lazuardi et al. (2019).

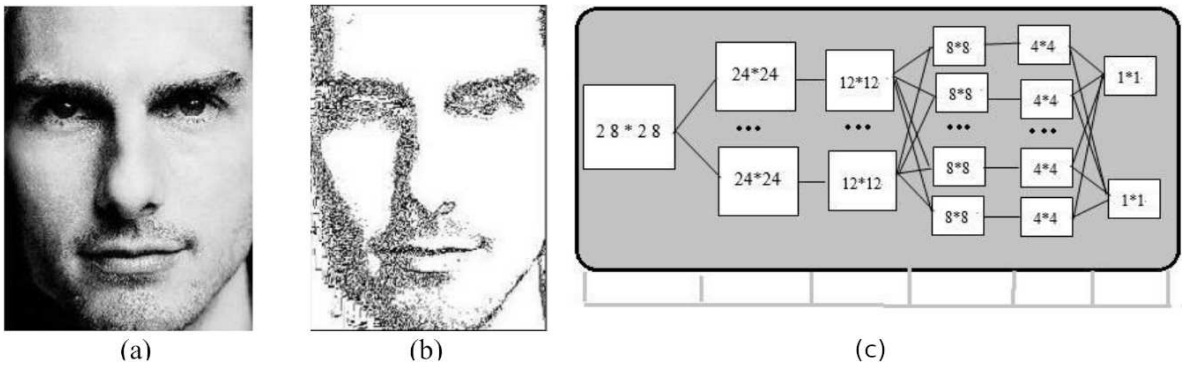
3.2 Descritor de Texturas LBP e Aprendizagem Profunda

O operador LBP foi desenvolvido com a finalidade de salientar a textura de um objeto retratado em uma imagem rasterizada para facilitar a análise e realizar sua classificação no campo de visão computacional (OJALA; PIETIKäINEN; HARWOOD, 1996).

Uma imagem tratada com o operador LBP gera um mapa de texturas que apresenta algumas características vantajosas, tais como pouca influência das variações de luminosidade na saída, imagem em nível de cinza, e destaque de estruturas como bordas e nódoas. Todos esses itens citados podem tornar alguns padrões estruturais contidos na imagem mais fáceis de serem identificados e aprendidos por uma rede neural artificial.

No estudo conduzido em Zhang et al. (2017), demonstrou-se que um modelo de CNN é suficiente para realizar o reconhecimento facial de uma imagem. As aplicações práticas dessa abordagem destacam-se especialmente no campo da segurança, onde a confirmação biométrica da identidade do usuário é crucial para o acesso a sistemas, como tratamento médico, monitoramento, interação humano-computador e serviços financeiros. A inovação do trabalho consiste na proposta de utilizar o mapa de texturas LBP para o reconhecimento facial, em contraste com a abordagem convencional baseada na imagem integral, como mostra o exemplo da Figura 25 (a) e (b).

Figura 25 – (a) Imagem original. (b) Mapa de texturas LBP. (C) Arquitetura CNN.



Fonte: montagem de Zhang et al. (2017).

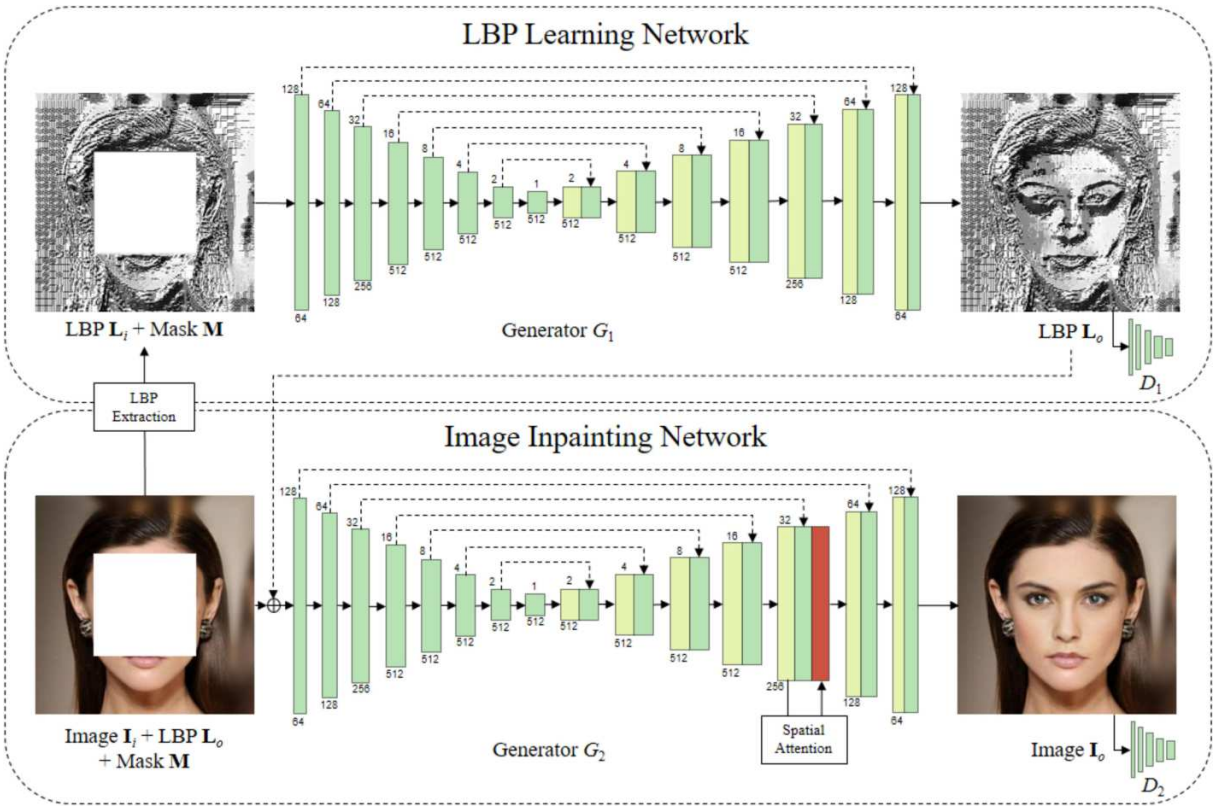
A arquitetura da CNN proposta neste estudo está retratada na Figura 25 (c), e é notavelmente simples, as imagens de entrada possuem resolução de `28x28` *pixels*. A rede é composta por duas camadas convolucionais, cada uma seguida por camadas de agrupamento correspondente, além de uma camada totalmente conectada. A camada de saída, por sua vez, apresenta apenas duas categorias, predizendo se a entrada corresponde a de uma face humana ou não.

Para efeitos de comparação, o mesmo modelo é treinado usando imagens rasterizadas. Os resultados do estudo revelam que as métricas de avaliação indicaram uma taxa de acerto significativamente maior quando treinadas com o mapa de texturas LBP. O estudo

declara que isso se deve ao fato de que o treinamento com imagens rasterizadas ocorre no nível mais fundamental do *pixel*, enquanto no mapa de texturas LBP, o aprendizado é fundamentado nas bordas da imagem processada. A conclusão é a de que o aprendizado da CNN é mais eficaz ao utilizar as informações de textura, melhorando a taxa de acerto do reconhecimento facial.

O trabalho em Wu, Zhou e Li (2022) apresenta um modelo generativo de duas *U-Nets* distintas concatenadas, representado na Figura 26, para reconstruir imagens que contêm áreas com *pixels* ausentes. O primeiro modelo *U-Net* é projetado para predizer as informações estruturais nas regiões faltantes, utilizando o mapa de texturas LBP da imagem original. A escolha do mapa de texturas LBP é justificada por conter uma quantidade significativa de informações estruturais. O segundo modelo *U-Net* visa preencher os *pixels* ausentes com cores adequadas, utilizando a imagem original junto com o mapa de texturas LBP reconstruído no primeiro estágio, como um guia para esse processo. A Figura 27 ilustra alguns exemplos de resultados obtidos pelo método proposto.

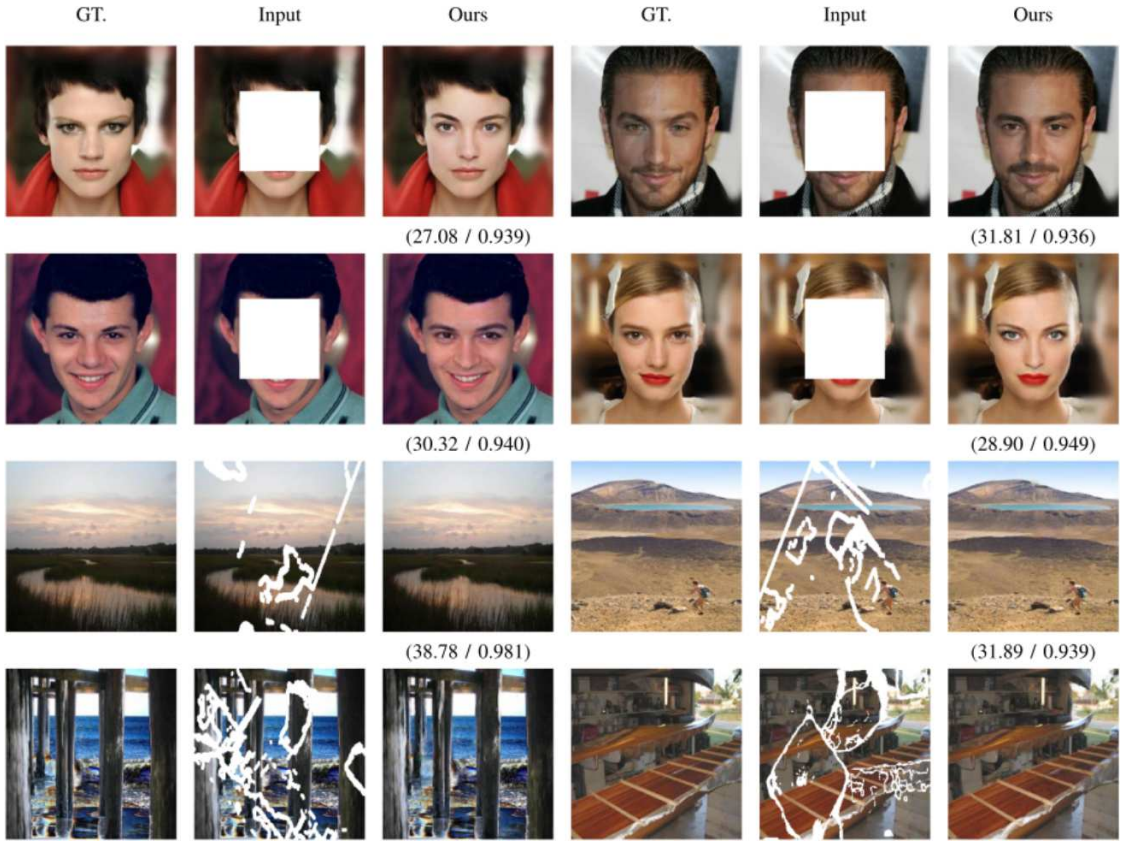
Figura 26 – Panorama da rede generativa proposta.



Fonte: Wu, Zhou e Li (2022).

Assim como na publicação anterior em Zhang et al. (2017), foi feito um teste para validar o impacto do mapa de texturas LBP. A segunda rede foi treinada para preencher a região dos *pixels* ausentes sem a referência do mapa de texturas LBP, e a Figura 28 ilustra que o resultado apresenta distorções conferindo um aspecto não natural. O estudo aponta

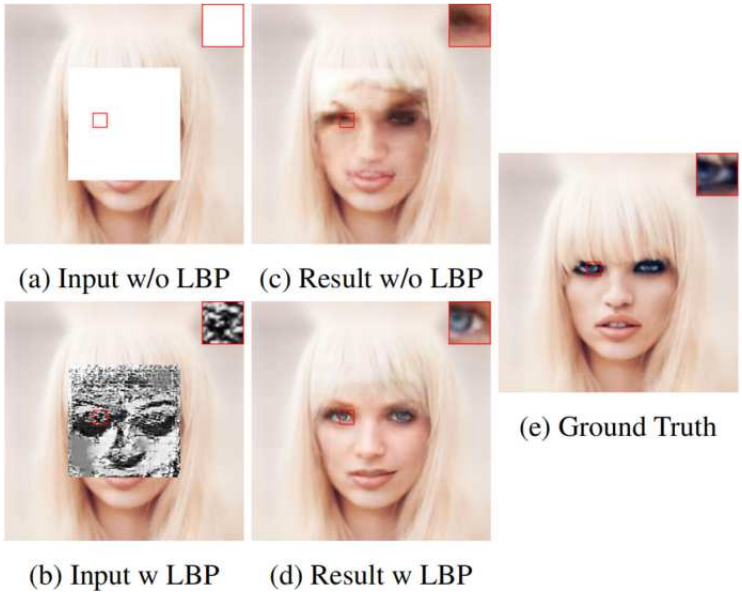
Figura 27 – Resultados da rede generativa. Cada conjunto possui três imagens, a esquerda corresponde à verdade de referência, a do meio, a imagem de entrada, e a da direita, o resultado gerado pelo método proposto.



Fonte: Wu, Zhou e Li (2022).

que a predição feita no primeiro estágio fornece um informações estruturais relevantes para melhorar o desempenho do segundo estágio.

Figura 28 – Impacto do mapa de texturas LBP. (a)-(b) são as entradas com e sem o mapa de texturas LBP. (c)-(d) são os resultados com e sem o mapa de texturas LBP. (e) é a verdade de referência.



Fonte: Wu, Zhou e Li (2022).

3.3 Considerações Parciais

Este capítulo pode ser dividido em duas partes. Na primeira parte, são citadas algumas técnicas de segmentação gráfica aplicadas à pavimentação asfáltica. É fundamental ressaltar que a eficácia dessas técnicas está condicionada a imagens com boa visibilidade das faixas de trânsito e com a via livre de outros objetos (como outros automóveis, motocicletas, e pessoas, por exemplo), pois isso tornaria o processo automatizado de segmentação demasiadamente complexo para ser descrito por um algoritmo. Nesse contexto, o uso de técnicas de IA apresenta-se como uma solução promissora. Esses modelos são capazes de aprender os padrões estruturais presentes nas imagens, permitindo a realização de predições com boa precisão mesmo com diversos tipos diferentes de objetos presentes na mesma cena.

Naturalmente, também dentro do campo de IA, existem inúmeras arquiteturas que podem ser utilizadas para a realização da segmentação semântica. Diversas pesquisas analisam as respostas de diferentes modelos, cada um com suas próprias características distintas. Essas variações podem incluir a base de dados específica adotada, a quantidade de categorias classificadas, a otimização do número de camadas para redução do tempo de processamento, entre outros fatores influenciados pela aplicação específica a que se destinam. Tantos detalhes discrepantes podem explicar as diferenças na precisão dos modelos nos trabalhos relacionados.

A segunda parte deste capítulo destaca dois artigos que foram uma fonte inspiradora para o desenvolvimento deste trabalho, ilustrando o potencial do mapa de texturas LBP em aprimorar o desempenho de redes neurais. O primeiro estudo utiliza mapas de texturas para treinar uma CNN na classificação de imagens, enquanto o segundo, utiliza um mapa de texturas LBP para reconstruir uma imagem com regiões ausentes.

O próximo capítulo descreve a metodologia deste trabalho. Serão detalhados os *softwares*, bibliotecas e plataformas computacionais utilizados, a escolha, implementação e parâmetros do descritor de texturas LBP, implementação da arquitetura *U-Net*, os ensaios de avaliação de desempenho do modelo, bem como as métricas empregadas para quantificar sua precisão.

4 MATERIAIS E MÉTODOS

Este capítulo descreve os recursos, métodos e procedimentos adotados no desenvolvimento deste trabalho, dividindo-se em quatro partes. Na primeira parte, são apresentados os recursos de *software* e *hardware*. Os recursos de *software* são todos gratuitos e incluem bibliotecas de código aberto e bases de dados para treinamento e teste do modelo. As bibliotecas utilizadas variam desde aquelas com funcionalidades simples para manipulação de arquivos até outras mais complexas e específicas para tratamento de imagens e implementação de redes neurais. Já os recursos de *hardware*, embora possam ser acessados em uma versão gratuita, oferecem maior quantidade de recursos na versão paga.

Na segunda parte, é apresentado o método que utiliza os mapas de texturas LBP para o treinamento da *U-Net*. Nesse ponto, a função de extração do mapa LBP a partir de imagens rasterizadas é detalhada. A implementação da *U-Net* é baseada nas bibliotecas de código aberto citadas no primeiro parágrafo. Os códigos mais importantes, incluindo o de todos os blocos convolucionais, camadas de agrupamento e blocos de convolução transposta que a compõem, também são detalhados.

Na terceira parte, são propostos três ensaios diferentes. Esses ensaios utilizam a arquitetura *U-Net* implementada no parágrafo anterior, e as bases de dados apresentadas no primeiro parágrafo. O objetivo é o de comprovar que a incorporação dos mapas de texturas LBP no treinamento aprimora o desempenho do modelo em condições não previstas no conjunto de treinamento original.

Na última parte, são discutidas algumas das métricas mais populares para avaliação de performance de redes neurais artificiais e segmentação semântica. Essas métricas medem a taxa de acerto das predições e podem ser utilizadas como valores de referência para comparações com outros trabalhos.

4.1 Bibliotecas e Ferramentas de Código Aberto

A listagem a seguir compreende as bibliotecas utilizadas no desenvolvimento deste projeto:

- *Open Source Computer Vision Library 4.10.0* (OpenCV): é uma biblioteca voltada para aplicações em visão computacional, análise e manipulação de imagens. É desenvolvida em C++, e oferece suporte para integração com diversas linguagens, incluindo *Python*, *Java* e MATLAB. Apesar de ser muito rica em recursos foram utilizadas apenas funcionalidades básicas para abrir imagens e redimensioná-las;

- *Scikit-Image 1.3.2*: é uma biblioteca que disponibiliza funções para processamento de imagens, tais como filtros, manipulação de cores, transformações geométricas, entre outras finalidades. Também pode ser utilizada com a linguagem *Python*;
- *Tensorflow 2.17.0*: é uma biblioteca desenvolvida em sua maior parte em C e C++ pela equipe *Google Brain* da empresa *Alphabet*. Foi projetada com o objetivo de facilitar a pesquisa nas áreas de Aprendizagem de Máquina e Inteligência Artificial;
- *Keras 3.4.1*: é uma biblioteca que se beneficia das características de alto nível de abstração, simplicidade da sintaxe e propósito geral da linguagem *Python* para fornecer uma interface de acesso aos recursos da biblioteca *TensorFlow.*;

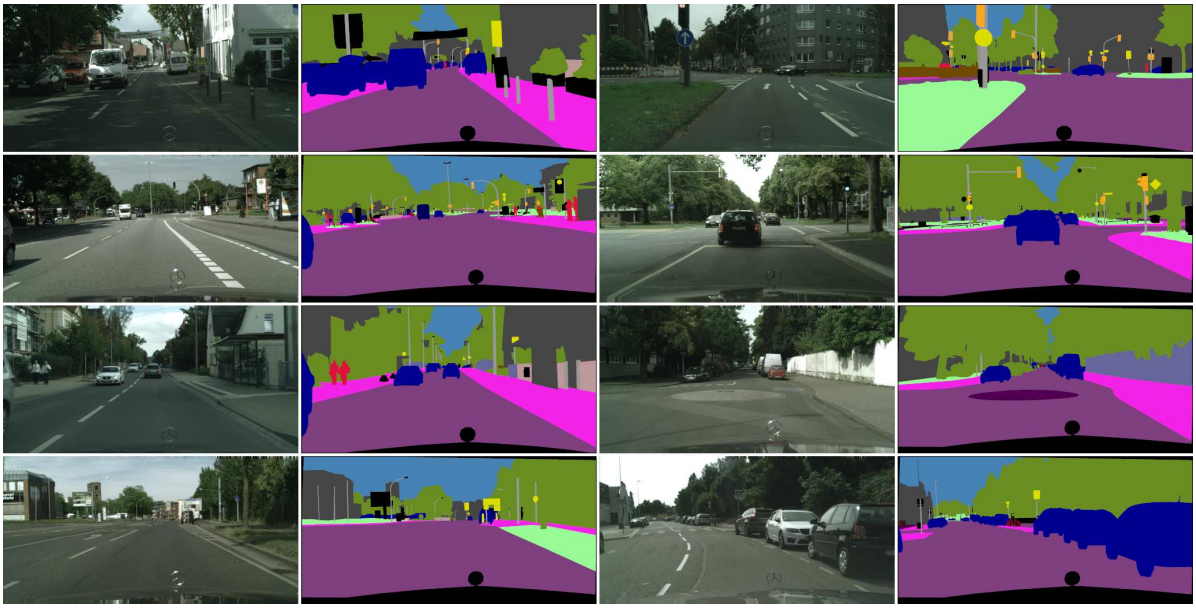
4.2 Base de Dados

Foram utilizadas duas bases de dados: *Cityscapes* (CORDTS et al., 2016) e *Adverse Conditions Dataset with Correspondences* (ACDC) (SAKARIDIS; DAI; GOOL, 2021). Conforme descrito em Cordts et al. (2016), a base de dados *Cityscapes* foi montada e disponibilizada gratuitamente com o intuito de facilitar a pesquisa científica na análise do desempenho de algoritmos de visão computacional em tarefas essenciais de interpretação semântica de ambientes urbanos. Contém imagens de alta resolução, com dimensões de `2048x1024` *pixels*, sendo 2974 imagens para treinamento e `500` imagens de teste, predominantemente capturadas nas ruas de diversas cidades europeias durante o período diurno em condições de tempo sem chuva.

Observando que a base de dados foi criada com o intuito de auxiliar nos estudos na área de segmentação semântica, as máscaras de classificação dos objetos também são disponibilizadas juntamente com cada imagem. Cada máscara contém até trinta classes de objetos diferentes, no entanto, como a proposta deste trabalho é realizar apenas a segmentação de pavimentações asfálticas, apenas a classe associada com ruas e estradas foi considerada e todas as demais foram descartadas. A Figura 29 exibe alguns exemplos de imagens e suas respectivas máscaras retiradas da base de dados *Cityscapes*.

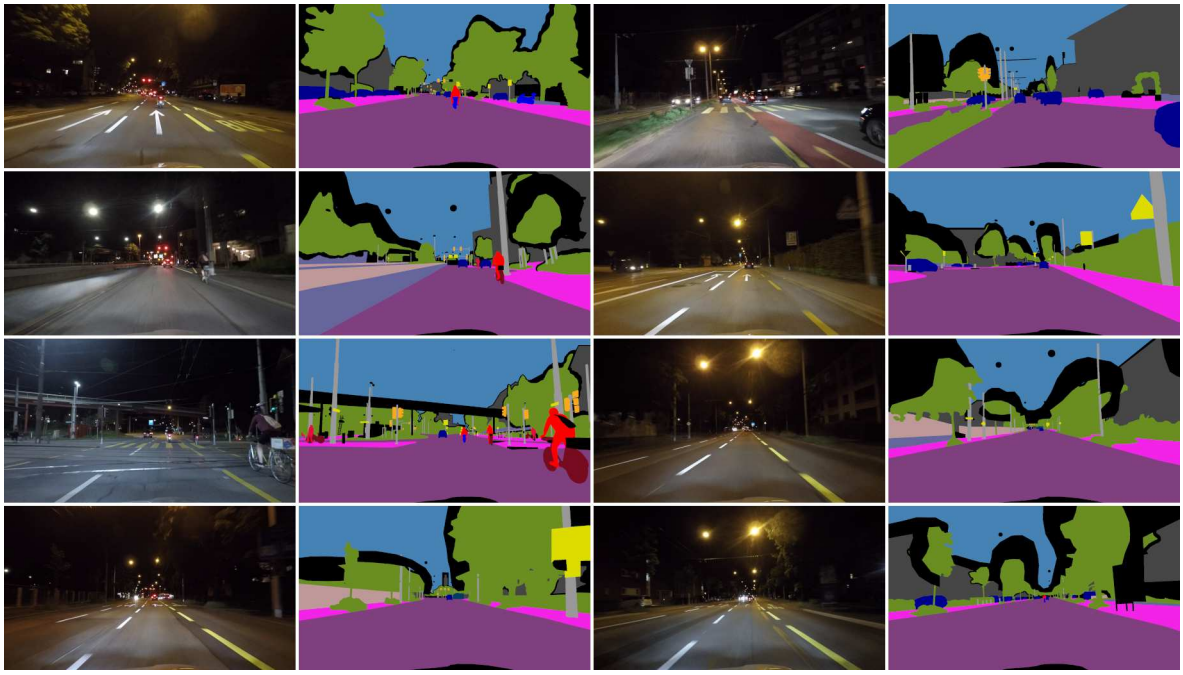
Devido à abrangência da base *Cityscapes*, outros conjuntos de dados adotaram o mesmo padrão de resolução para imagens e classes, o que possibilita o uso de múltiplas bases para treinamento e teste de um único modelo. De fato, a segunda base de dados, ACDC foi montada adotando praticamente as mesmas dimensões para as imagens, de `1920x1080` *pixels*, e classes para as máscaras associadas. Possui `4006` imagens capturadas em diferentes condições climáticas adversas. Entretanto, apenas `400` imagens retratando cenários noturnos foram utilizadas para treinamento, enquanto `106` imagens foram destinadas ao teste. A Figura 30 exibe alguns exemplos de imagens e suas respectivas máscaras retiradas da base de dados *Cityscapes*.

Figura 29 – Exemplos de imagens da base de dados *Cityscapes* com suas respectivas máscaras.



Fonte: Cordts et al. (2016).

Figura 30 – Exemplos de imagens noturnas da base de dados ACDC com suas respectivas máscaras.



Fonte: Sakaridis, Dai e Gool (2021).

O uso das bases de dados da maneira sugerida estabelece um padrão de comparação. Ou seja, outros trabalhos que utilizarem as mesmas bases de dados podem empregar o mesmo conjunto de imagens para treinamento e teste, permitindo comparações mais justas entre os resultados.

4.3 Ambiente de Desenvolvimento

O desenvolvimento foi realizado na plataforma *Colab* (ou *Colaboratory*) disponibilizada pela *Google Research*, uma divisão da empresa *Alphabet*. O *Colab* oferece um serviço de *notebooks* para diversos propósitos incluindo educação, Aprendizagem de Máquina e análise de dados. A principal vantagem é a disponibilização do uso de GPUs para treinamento e testes de redes neurais artificiais, sem a necessidade de configuração. Os recursos podem ser acessados e utilizados de forma gratuita, embora com algumas limitações. Além disso, é possível adquirir créditos computacionais, ou optar por planos para se ter prioridade no acesso desses recursos.

O processador disponibilizado pelo *Colab* é o modelo *Intel*(R) *Xeon*(R) `CPU @ 2.20 GHz`, a memória RAM é de `83,5` GB para a GPU `A100 40 GB` ou de `51,0` GB para as demais opções, e a linguagem de programação *Python* está na versão `3.10`. O sistema possui algumas opções de placas GPUs, que estão listadas na Tabela 2 com suas principais características.

Tabela 2 – Relação das especificações das GPUs disponibilizadas na plataforma *Colab*

Modelo	**Memória**	**FP64**	**FP32**
A100 40 GB	40 GB HBM2@1.6TBps	9.7 TFLOPS	19.5 TFLOPS
V100	16 GB HBM2@900GBps	9.7 TFLOPS	19.5 TFLOPS
T4	16 GB GDDR6@320GBps	0.25 TFLOPS	8.1 TFLOPS

O tempo de treinamento é mais demorado quanto maiores forem a quantidade de imagens alocadas para treino, o tamanho e o número de camadas de cores. Todas as `2974` imagens da base de dados *Cityscapes* é apenas possível de ser treinada no modelo `A100 40 GB` por ser a opção que disponibiliza mais memória. No entanto, para efeitos simples de comparação, foi realizado um teste para aferir o tempo de treinamento aproximado de uma época nos três modelos de GPUs utilizando um conjunto reduzido de `1000` imagens. Os resultados estão registrados na Tabela 3.

O tempo de treinamento utilizando apenas a CPU é demasiadamente demorado, extrapolando 40 minutos por época.

4.4 O Método Baseado no Mapa LBP para *U-Net*

A utilização do algoritmo LBP na forma de histograma descreve as características de texturas da imagem, mas descarta a informação da posição dos *pixels*. No entanto,

Tabela 3 – Comparação entre os tempos de treinamento aproximados das GPUs para um conjunto de 1000 imagens

Modelo	**Tempo (s)**
A100 40GB	12
V100	25
T4	94

para aplicações em Aprendizagem Profunda pode existir um interesse maior na extração de informações a partir do mapa de texturas LBP, pois o mapa contém os registros de ordenação relativa em um bloco de *pixels*, ou seja, uma quantidade de padrões melhor evidenciados tais como bordas, nódoas e outras estruturas que podem melhorar e facilitar a aprendizagem em relação à imagem original, além de atenuar os efeitos de iluminação.

Conforme discutido na Subseção 3.2, o uso de mapas LBP para treinamento de redes neurais artificiais pode apresentar uma melhora nas métricas de avaliação das redes neurais artificiais. Isso pode ser explicado por duas razões principais: remoção dos efeitos de iluminação, e evidenciação de estruturas na imagem. Isso facilita para a rede aprender diretamente pelos padrões evidenciados aliviando o processo de aprendizado de um método para extraí-las.

A função `imread()` é utilizada para carregar uma imagem, o parâmetro `0` indica que a imagem deve ser carregada em preto e branco.

O funcionamento do operador LBP foi apresentado na Seção 2.4, por se tratar de um método estabelecido, existem inúmeras fontes abertas que já disponibilizarm a implementação em código. Este trabalho aproveita a função `local_binary_pattern()`, da biblioteca *scikit-image* (PEDREGOSA et al., 2011) para produzir um mapa LBP, a partir de três parametros de entrada: a imagem em preto e branco, a quantidade de vizinhos `P`, e a distância entre o *pixel* central e os vizinhos `R`. O Código 1 apresenta o código utilizado para carregar uma imagem rasterizada e gerar seu mapa de texturas LBP, enquanto a Figura 31 compara uma imagem rasterizada com o seu respectivo mapa de texturas LBP.

Código 1 – Código para aplicação do algoritmo LBP em imagens

```
1 import cv2
2 from skimage.feature import local_binary_pattern
3
4 imagem = cv2.imread('/content/drive/MyDrive/Colab Notebooks/imagem.png',
      0)
5 mapa_lbp = local_binary_pattern(imagem, P=8, R=1)
```

Figura 31 – Imagem original à esquerda e mapa LBP à direita

Fonte: Montagem a partir de uma imagem da base de dados Cordts et al. (2016)

4.4.1 Segmentação de Pavimentações Asfálticas Utilizando *U-Net*

A segmentação de pavimentações asfálticas é a tarefa de classificar cada *pixel* em uma imagem rasterizada para determinar se pertence a uma pavimentação asfáltica ou não. Isso pode ser realizado por meio de uma rede neural, treinando-a de forma convencional com imagens rasterizadas contendo três camadas de cores, conforme vários trabalhos publicados em Yousri, Elattar e Darweesh (2021), em Jebamikyous e Kashef (2021), em Sarmah, Gogoi e Kalita (2022), em Pham (2021) e em Lazuardi et al. (2019). Ao seguir esse método, na fase de aprendizado, a extração das características deve ser realizada a partir do nível mais baixo, ou seja, o nível do *pixel*. A substituição da imagem rasterizada pelo seu mapa de texturas LBP na entrada do modelo deve facilitar o processo de aprendizagem, uma vez que algumas características são evidenciadas, elevando assim o ponto de partida para a extração das características. Espera-se que o modelo apresente um desempenho melhor, especialmente em casos de variações de luminosidade que não estavam presentes na base de dados de treinamento.

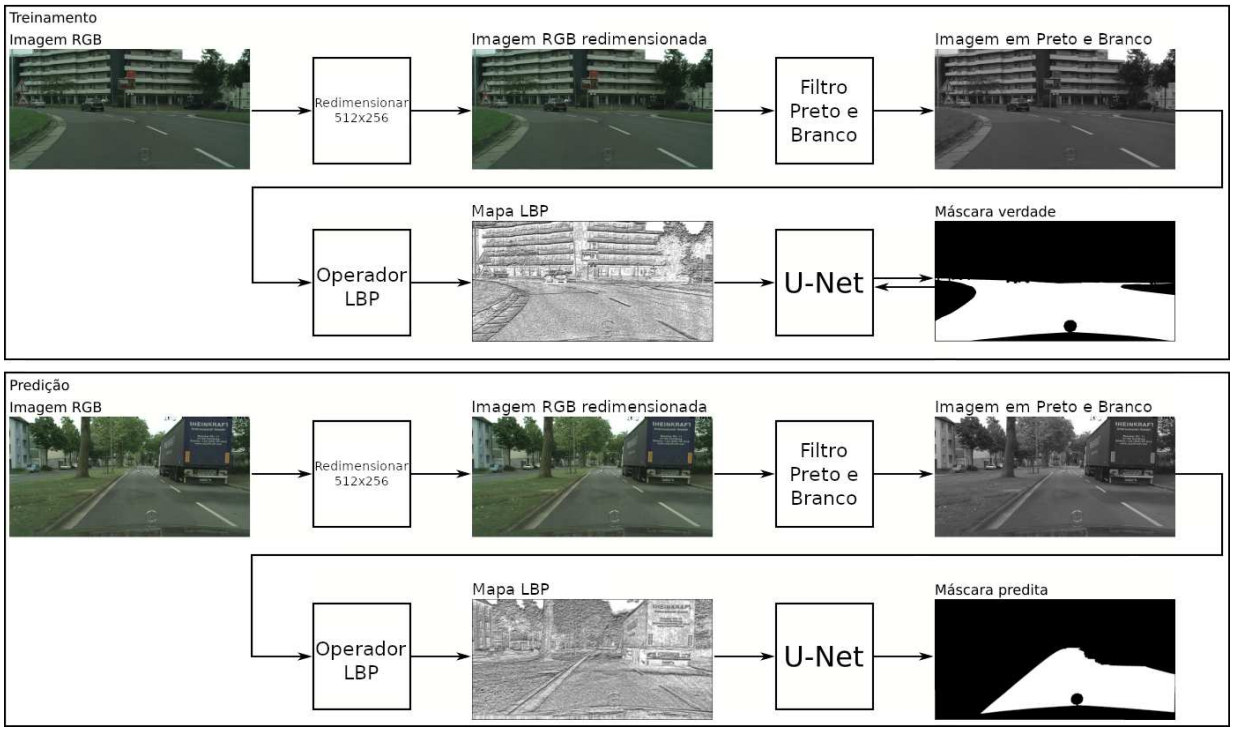
4.4.2 Aprendizagem com Mapas de Texturas LBP para *U-Net*

Este trabalho propõe a segmentação de ruas em imagens rasterizadas utilizando os mapas LBP das imagens nas etapas de treinamento e testes em um modelo *U-Net*, como exibido na Figura 32.

As bases de dados disponibilizam imagens rasterizadas que precisam ser redimensionadas. Isso é feito através da função `cv2.imread()`. O parâmetro `interpolation` foi especificado com o tipo `INTER_AREA`, pois essa técnica considera os *pixels* vizinhos e calcula a média ponderada para determinar os valores dos *pixels* na nova imagem.

No entanto, as máscaras também precisam ser ter suas dimensões reduzidas, e o tipo `INTER_NEAREST` deve ser utilizado para evitar a alteração no valor dos elementos da máscara. Esse método é o mais simples em termos de cálculos, pois atribui a cada *pixel* na imagem redimensionada o valor do *pixel* mais próximo na imagem original. O Código 2 mostra como as funções com os parâmetros adequados para redimensionamento

Figura 32 – Descrição do projeto. A parte de cima descreve a etapa de treinamento da rede, enquanto a parte de baixo, a etapa de predição



Fonte: De autoria própria.

da imagem rasterizada e sua respectiva máscara.

Código 2 – Código para redimensionamento de imagens

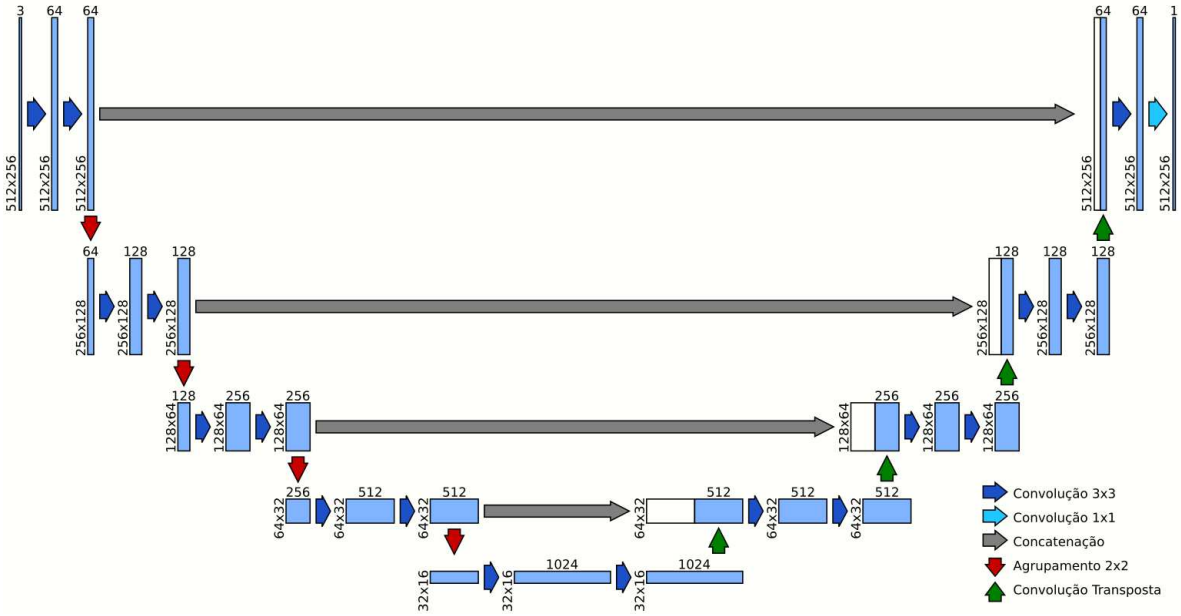
```
imagem = cv2.imread('/content/drive/MyDrive/Colab Notebooks/imagem.png')
imagem_reduzida = cv2.resize(imagem, (512, 256), interpolation = cv2.
    INTER_AREA)

mascara = cv2.imread('/content/drive/MyDrive/Colab Notebooks/mascara.png
    ')
mascara_reduzida = cv2.resize(mascara, (512, 256), interpolation = cv2.
    INTER_NEAREST)
```

A imagem reduzida pode ser salva utilizando o comando `cv2.imwrite()` - isso é benéfico pois evita que o processo de redução de resolução tenha que ser feito sempre que o código seja executado.

As imagens reduzidas são encaminhadas para a entrada da rede *U-Net*. A arquitetura *U-Net* utilizada neste trabalho é apresentada na Figura 33, e é muito semelhante à descrita pelo trabalho original em Ronneberger, P.Fischer e Brox (2015) (na Figura 10).

A *U-Net* possui dois estágios distintos: o codificador e o decodificador. O codificador possui quatro camadas. Na entrada, a imagem é submetida a camada `Input()` que retorna um tensor, uma estrutura matemática que representa os dados de entrada para

Figura 33 – Arquitetura *U-Net* utilizada neste trabalho.



Fonte: De autoria própria.

o modelo e que será utilizado pelas camadas subsequentes. Ao criar a camada `Input()`, especifica-se o formato da imagem de entrada. Em seguida, é realizado o primeiro processos de convolução pela função `Conv2D()`, em que são especificados `64` filtros de tamanho (`3x3`), *stride* padrão de tamanho (`1x1`) e *padding* de tamanho `2`, a fim de manter a saída gerada com as mesmas dimensões da entrada. O resultado é armazenado no tensor $c1$.

Em seguida, a função `BatchNormalization()` é aplicada. Este procedimento tem como objetivo normalizar os elementos da camada durante o treinamento, acelerando o processo de aprendizado e aprimorando o desempenho e a estabilidade da rede. A normalização em lote contribui para manter uma distribuição mais estável ao longo das iterações, facilitando a convergência durante o treinamento.

A função `Activation()` apenas define a função de ativação com o tipo ReLU. O processo de convolução é repetido de forma idêntica, e no final, a função `MaxPool2D()` executa a camada de agrupamento com tamanho (`2x2`). É crucial observar que a saída da função `MaxPool2D()` é armazenada em um tensor $p1$ diferente. Isso é feito pois o tensor $c1$ terá utilidade ainda no próximo estágio de decodificação. O Código 3 mostra a implementação do código da camada de entrada.

Código 3 – Código da parte de entrada do codificador da U-Net

```
entradas = Input((256,512,1))

c1 = Conv2D(64, 3, padding="same")(entradas)
c1 = BatchNormalization()(c1)
c1 = Activation("relu")(c1)
```

```
6
7  c1 = Conv2D(64, 3, padding="same")(c1)
8  c1 = BatchNormalization()(c1)
9  c1 = Activation("relu")(c1)
10
11 p1 = MaxPool2D((2, 2))(c1)
```

As três camadas seguintes do codificador são bastante semelhantes. As diferenças são a ausência da camada de `Input()`, e a quantidade de filtros da função `Conv2D` que dobra após a execução da função `MaxPool2D()`. O Código 4 mostra a implementação do código das camadas internas do codificador.

Código 4 – Código da segunda parte do codificador da U-Net

```
1  c2 = Conv2D(128, 3, padding="same")(p1)
2  c2 = BatchNormalization()(c2)
3  c2 = Activation("relu")(c2)
4
5  c2 = Conv2D(128, 3, padding="same")(c2)
6  c2 = BatchNormalization()(c2)
7  c2 = Activation("relu")(c2)
8
9  p2 = MaxPool2D((2, 2))(c2)
```

A terceira e quarta camadas podem ser implementadas substituindo-se os termos `p1` por `p2` e `p3` (na primeira linha), `c2` por `c3` e `c4` (em todas as linhas), `p2` por `p3` e `p4` (na última linha), e a quantidade de filtros de `128` para `256` e `512` (na função `Conv2D`), respectivamente.

A ponte é uma estrutura que conecta o codificador ao decodificador. Novamente, a implementação é bastante semelhante ao das partes `2`, `3`, e `4`, sendo a distinção feita pela ausência da camada de agrupamento e pelo número de filtros na função `Conv2D` ser de `1024`. O Código 5 mostra a implementação do código da ponte.

Código 5 – Código da ponte da U-Net

```
1  b1 = Conv2D(1024, 3, padding="same")(p4)
2  b1 = BatchNormalization()(b1)
3  b1 = Activation("relu")(b1)
4
5  b1 = Conv2D(1024, 3, padding="same")(b1)
6  b1 = BatchNormalization()(b1)
7  b1 = Activation("relu")(b1)
```

O decodificador também está sub-dividido em quatro camadas. Na fase inicial, o tensor de saída da ponte é introduzido na função `Conv2DTranspose()`, em que ocorre o processo de convolução transposta. São especificados `512` filtros de tamanho (`2x2`), *stride*

de tamanho (2x2) e *padding* de tamanho 2. Isso faz com que as dimensões do tensor de saída d1 coincidam com as do tensor s4 da última camada do codificador.

Em seguida, a função Concatenate() realiza a concatenação dos dois tensores $d1$ e $s4$. Nesse ponto, ocorre a recuperação da informação espacial que foi perdida pela camada de agrupamento na quarta parte do estágio de codificação. O restante da implementação segue de maneira idêntica à da ponte. O Código 6 mostra a implementação do código das camadas do decodificador.

Código 6 – Código do decodificador da U-Net

```
d1 = Conv2DTranspose(512, (2, 2), strides=2, padding="same")(b1)
d1 = Concatenate()([d1, c4])

d1 = Conv2D(512, 3, padding="same")(d1)
d1 = BatchNormalization()(d1)
d1 = Activation("relu")(d1)

d1 = Conv2D(512, 3, padding="same")(d1)
d1 = BatchNormalization()(d1)
d1 = Activation("relu")(d1)
```

Todas as partes do decodificador compartilham a mesma estrutura, variando apenas nos parâmetros dos filtros e nos índices dos tensores. Analogamente, a implementação das partes 2, 3 e 4 podem ser feitas apenas substituindo-se o termo b1 por d1, d2 e d3, o termo d1 por d2, d3 e d4, e o termo c4 por c3, c2 e c1, respectivamente.

Finalmente, a camada de saída pode ser implementada recebendo o tensor $d4$ gerado na quarta parte do decodificador, como mostra o Código 7.

Código 7 – Código da camada de saída da U-Net

```
saidas = Conv2D(1, 1, padding="same", activation='sigmoid')(d4)
```

Na camada de saída, é realizado um único processo de convolução com os seguintes parâmetros: um filtro de tamanho (1x1), pois a predição ocorre exclusivamente para uma classe (se o *pixel* pertence ou não a uma rua), *padding* de tamanho 1, para a saída manter as mesmas dimensões da entrada, e ativação sigmoide. A escolha da ativação sigmoide é fundamentada na sua característica de produzir valores em um intervalo entre 0 e 1, o que pode ser interpretado como uma probabilidade de pertencer à classe. Isto é, se o valor for maior que 0,5, interpreta-se que o *pixel* pertence à classe; caso contrário, não pertence.

O código completo de implementação do modelo *U-Net* está disposto no Apêndice A. É relevante comentar que o modelo *U-Net* apresentado nesta seção pode ser convertido para predizer múltiplas classes, apenas modificando os parâmetros de quantidade de filtros (de 1 para o número de classes desejadas) e a função de ativação (de sigmoide para *softmax*).

O modelo deve ser instanciado e compilado, ou seja, ter os seus hiperparâmetros (parâmetros internos), inicializados. Os hiperparâmetros de interesse foram abordados no decorrer da Seção 2.1.6, e são fixos para este trabalho: taxa de aprendizado de `0,2e-3`, otimizador "Adam", e função de custo "Binary Crossentropy". O Código 8 exibe a implementação em código para a criação do modelo e compilação.

Código 8 – Instanciação da U-Net e compilação com hiperparâmetros

```
1 modelo = Model(entradas, saidas, name="U-Net")
2 modelo.compile(optimizer=Adam(learning_rate = 0.2e-3), loss='
    binary_crossentropy', metrics=['accuracy'])
```

A compilação do modelo aloca o espaço necessário em memória para todos os seus elementos, conforme listadas na Tabela 4

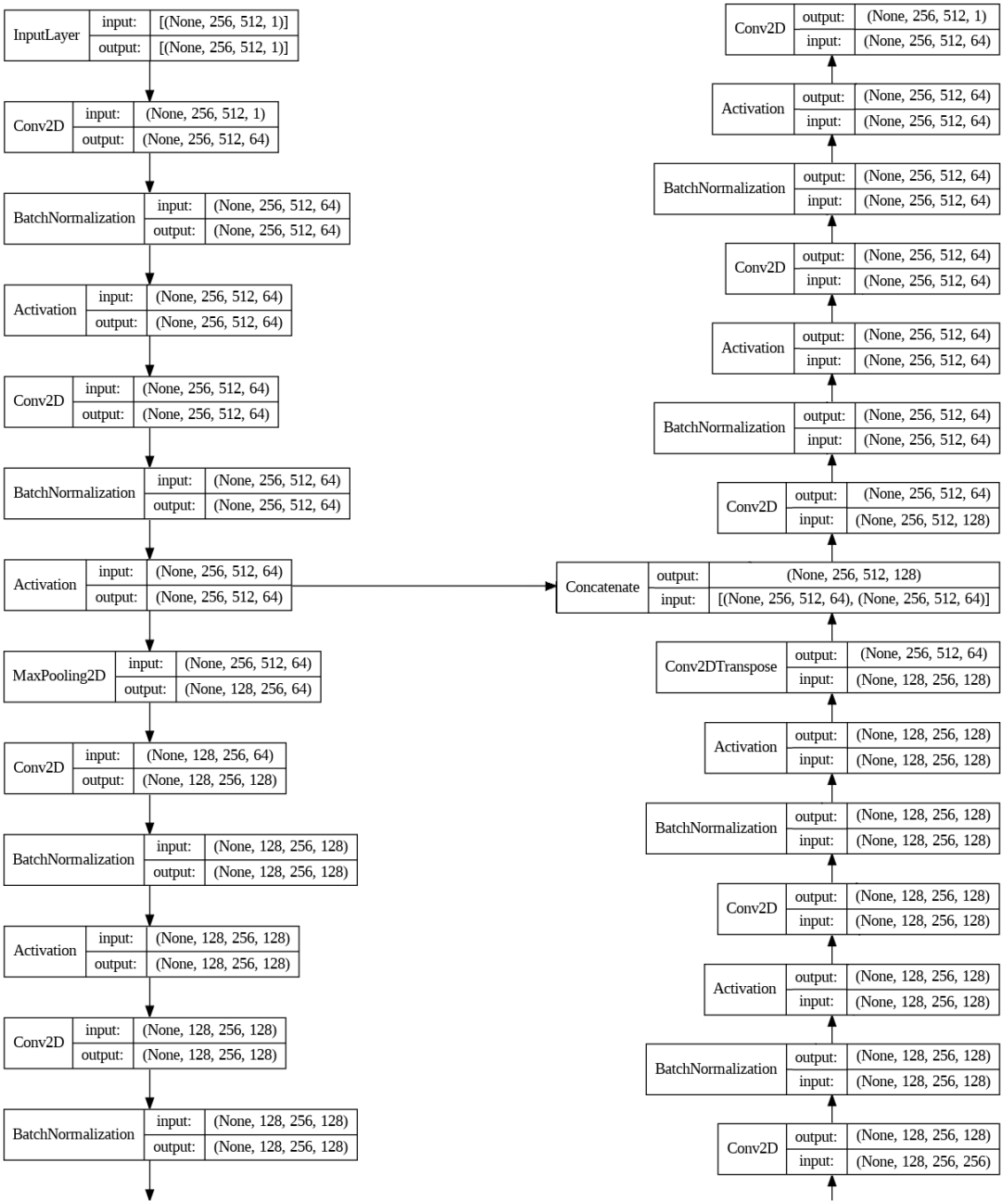
Tabela 4 – Características da *U-Net*

Total params:	31.055.297	118,47 MB
Trainable params:	31.043.521	118,42 MB
Non-trainable params:	11.776	46,00 KB

Em que `Total params` é a somatória de `Trainable params` mais `Non-trainable params`. É relevante deixar claro que o campo `Trainable params` se refere ao total de pesos treináveis e não ao total de *perceptrons*. Por exemplo, na convolução, um *perceptron* irá atuar como o *kernel* do processo, e se o *kernel* tiver tamanho `9` (ou `3x3`), então o total de parâmetros treináveis desse *perceptron* será de `9` pesos referentes às entradas mais `1` peso referente ao *bias*, ou seja, `10` parâmetros. Pesos treináveis são aqueles que podem ser ajustados durante a etapa de treinamento. O campo `Non-trainable params`, por sua vez, se refere ao total de pesos cujos valoresweb não podem ser alterados. Cada parâmetro é do tipo *float* e possui `4` *bytes*, então a terceira coluna da tabela 4 será o resultado direto da multiplicação da segunda coluna por `4`. O sumário completo do modelo foi incluído em Anexos-B.

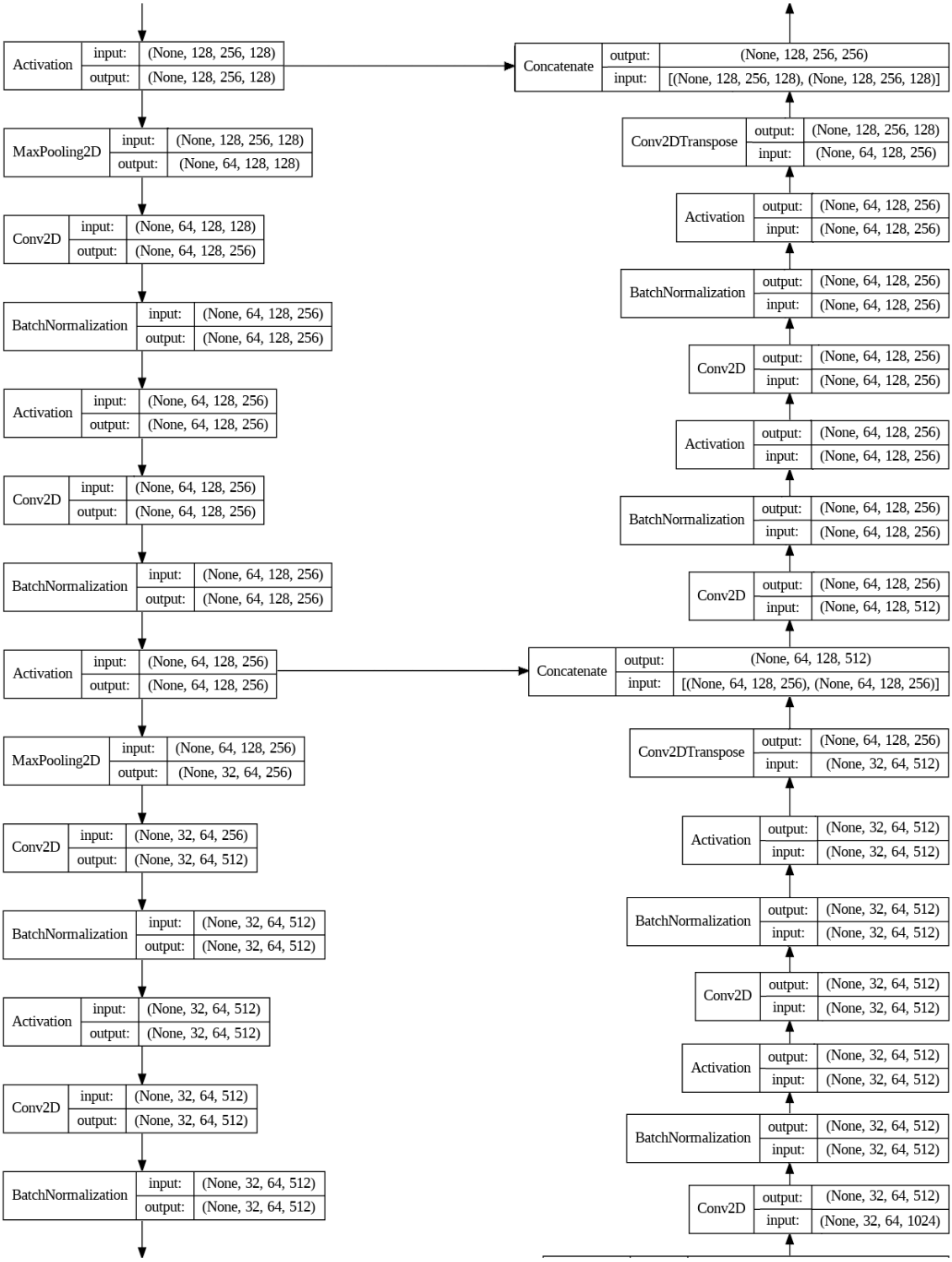
A partir do modelo compilado, é possível ainda utilizar a biblioteca `plot_model` para produzir um diagrama, permitindo a visualização na forma gráfica. O diagrama foi dividido nas Figuras 34, 35 e 36 para ficar com tamanho adequado.

Figura 34 – Estrutura *U-Net* - parte 1.



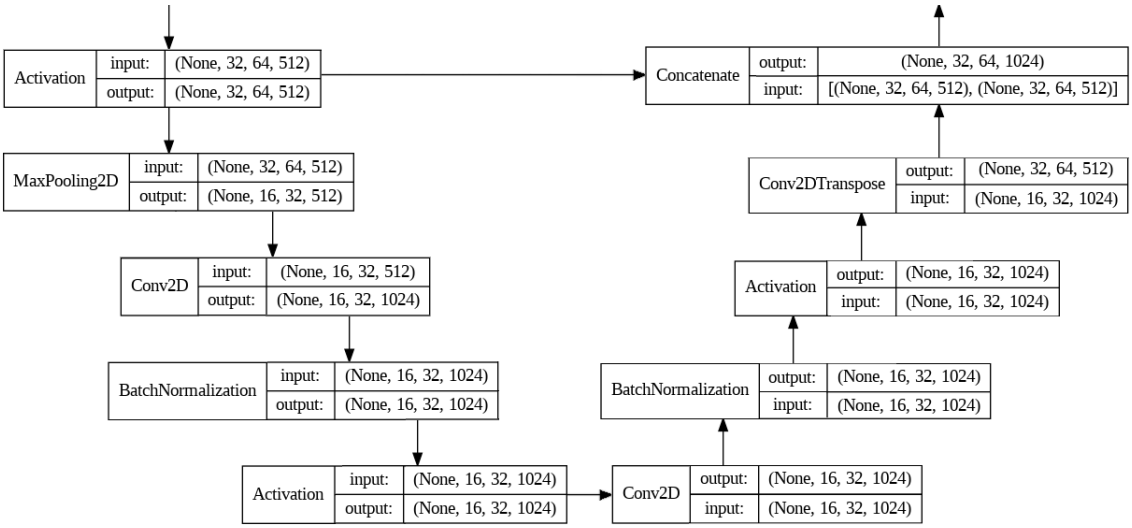
Fonte: De autoria própria.

Figura 35 – Estrutura *U-Net* - parte 2.



Fonte: De autoria própria.

Figura 36 – Estrutura *U-Net* - parte 3.



Fonte: De autoria própria.

4.5 Ensaios

São realizados três ensaios com o objetivo de validar a efetividade da *U-Net* na segmentação semântica de pavimentações asfálticas em imagens rasterizadas e investigar as vantagens do uso de mapas de texturas LBP no treinamento do modelo.

A fim de garantir a homogeneidade e a comparabilidade dos resultados, todos os ensaios empregam a mesma arquitetura *U-Net* definida na Seção 4.4.2, e as duas bases de dados descritas na Seção 4.2: *Cityscapes*, que contém apenas imagens diurnas, e ACDC, que contém apenas imagens noturnas.

A avaliação possui duas etapas: treinamento e teste. É importante relembrar que cada base de dados possui uma divisão pré-definida e fixa de imagens para cada etapa. Isto é, para a etapa de treinamento, a base de dados *Cityscapes* possui `2974` imagens, e a ACDC possui `400` imagens; e para a etapa de testes, a *Cityscapes* possui `500` imagens, e a ACDC possui `106` imagens.

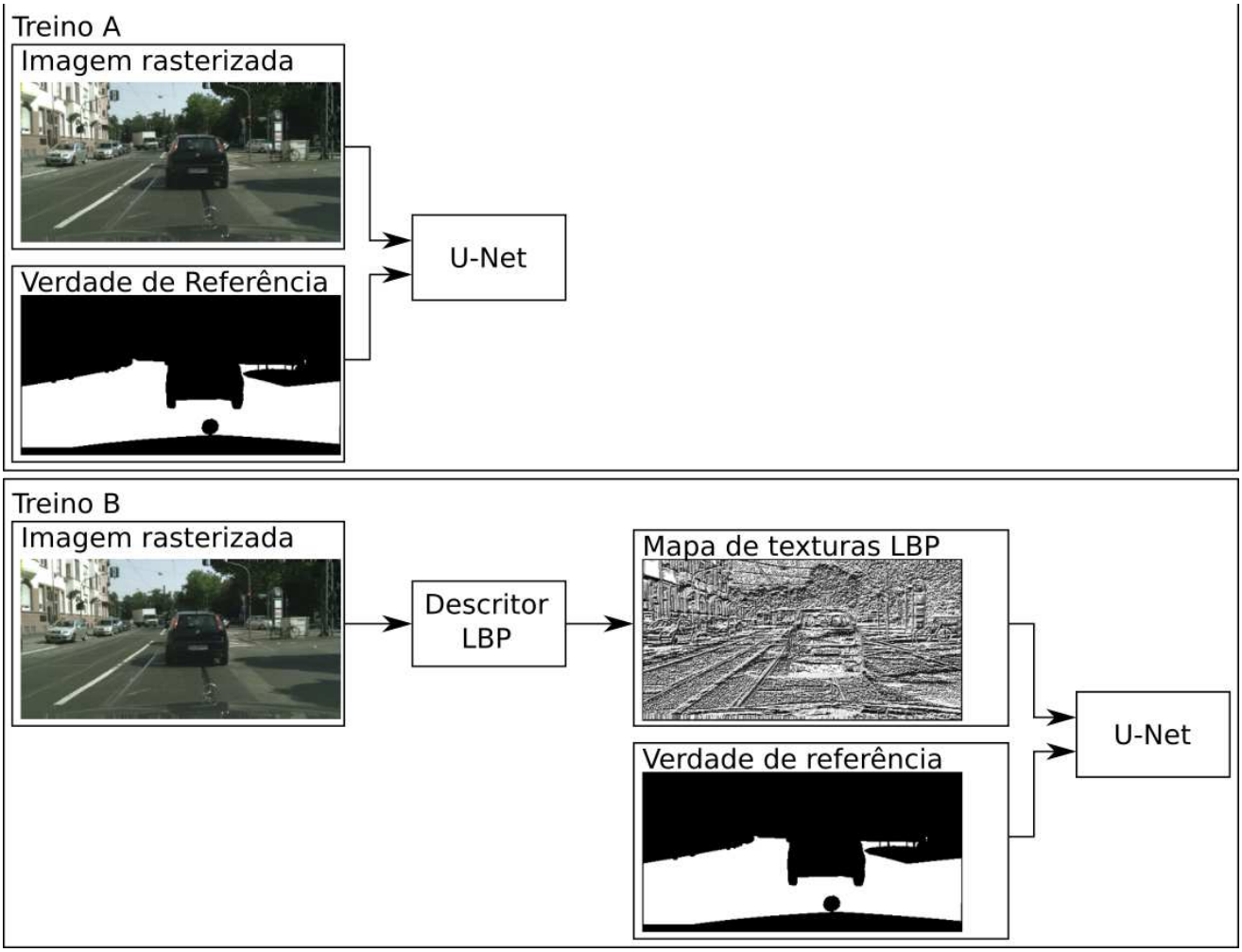
Cada treinamento possui objetivos diferentes e as subseções seguintes correspondentes a cada ensaio devem especificar o uso ou não das `2974` imagens de treinamento da base de dados *Cityscapes* e das `400` imagens da ACDC. Qualquer que seja a base de dados utilizada, ou mesmo na utilização de ambas, `80`% das imagens são destinadas para o treinamento propriamente dito, enquanto os `20`% restantes são destinados à validação desse treinamento. É preciso destacar que o processo de validação ainda é parte do treinamento, sendo anterior e distinto da etapa de testes.

A etapa de testes é comum a todos os ensaios, e compreende três testes. O primeiro teste é realizado usando apenas as `500` imagens de teste diurnas da base de dados *Cityscapes*. Já no segundo teste, são usadas apenas as `106` imagens de teste noturnas da base de dados ACDC. O terceiro teste combina as `500` imagens de teste diurnas da base de dados *Cityscapes* com as `106` imagens da ACDC. É relevante mencionar que o resultado do terceiro teste é uma média ponderada dos resultados obtidos nos dois testes anteriores. Os resultados da etapa de testes são usados para processar as métricas de avaliação de desempenho mais importantes, e que servem de referência para comparações.

4.5.1 Ensaio 1: Teste de Invariância de Iluminação dos Mapas de Texturas LBP

O primeiro ensaio tem como objetivo testar a característica de invariância à iluminação dos mapas de texturas LBP. Dessa forma, dois treinamentos distintos, denominados Treino A e Treino B, são realizados utilizando exclusivamente as imagens diurnas, de acordo com o diagrama da Figura 37. O treino A emprega as imagens rasterizadas originais, enquanto o Treino B, os mapas de texturas correspondentes às imagens do Treino A.

Figura 37 – Treino A acima e Treino B abaixo



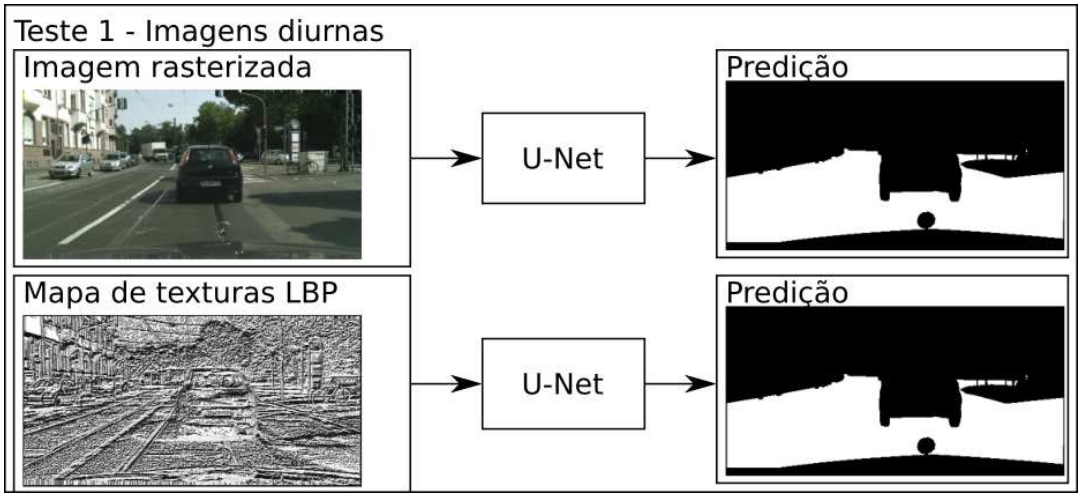
Fonte: De autoria própria.

Após a etapa de treinamento, são realizados os três testes descritos no início desta Seção 4.5. O primeiro teste é realizado com imagens diurnas de acordo com o diagrama da Figura 38. Este teste comprova a funcionalidade de segmentação semântica de pavimentações asfálticas em imagens rasterizadas e em mapas de texturas LBP.

O segundo teste é realizado com imagens noturnas, de acordo com o diagrama da Figura 39. Este teste é crítico pois a melhor performance do Treino B em relação ao Treino A reforça a declaração de invariância dos mapas de texturas LBP em relação às variações de tons de cinza (OJALA; PIETIKäINEN; HARWOOD, 1996), e reforça também a declaração de que a capacidade dos mapas de texturas evidenciam os padrões estruturais e de textura (ZHANG et al., 2017) (WU; ZHOU; LI, 2022).

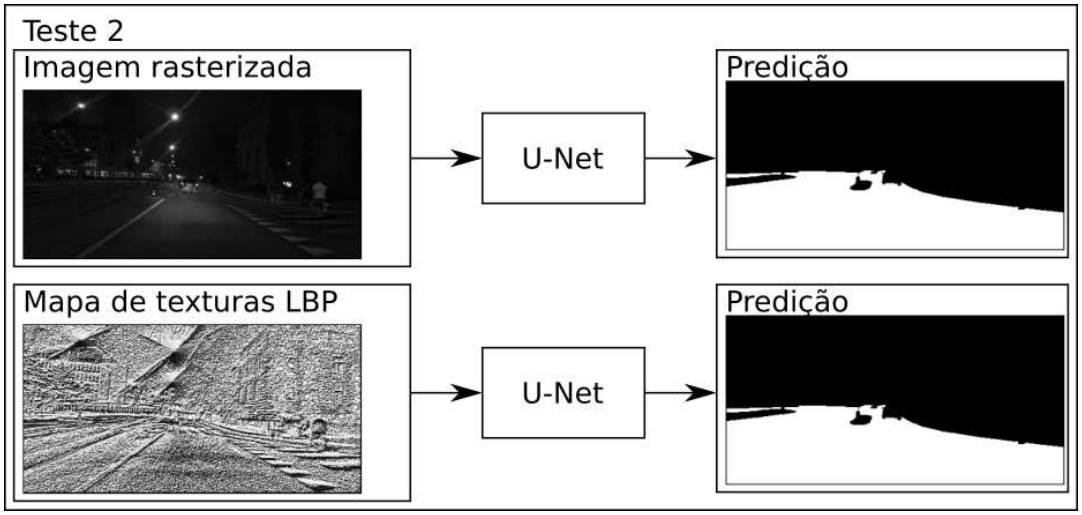
Conforme mencionado no início desta Seção 4.5, o terceiro teste combina as imagens diurnas e noturnas dos dois testes anteriores. Em outra interpretação, o resultado do terceiro teste é uma média ponderada dos resultados obtidos nos dois testes anteriores.

Figura 38 – Teste 1 com imagens diurnas.



Fonte: De autoria própria.

Figura 39 – Teste 2 com imagens noturnas.



Fonte: De autoria própria.

4.5.2 Ensaio 2: Comparação de Performance do Operador LBP

O segundo ensaio serve como controle experimental, com o objetivo de certificar que a inclusão de imagens noturnas na etapa de treinamento efetivamente melhora o desempenho do modelo em testes com imagens noturnas.

São realizados dois treinamentos distintos: Treino C e Treino D. Ambos utilizam os dois conjuntos de imagens diurnas e noturnas. O Treino C emprega as imagens rasterizadas originais, enquanto o Treino D, os mapas de texturas correspondentes às imagens do Treino C.

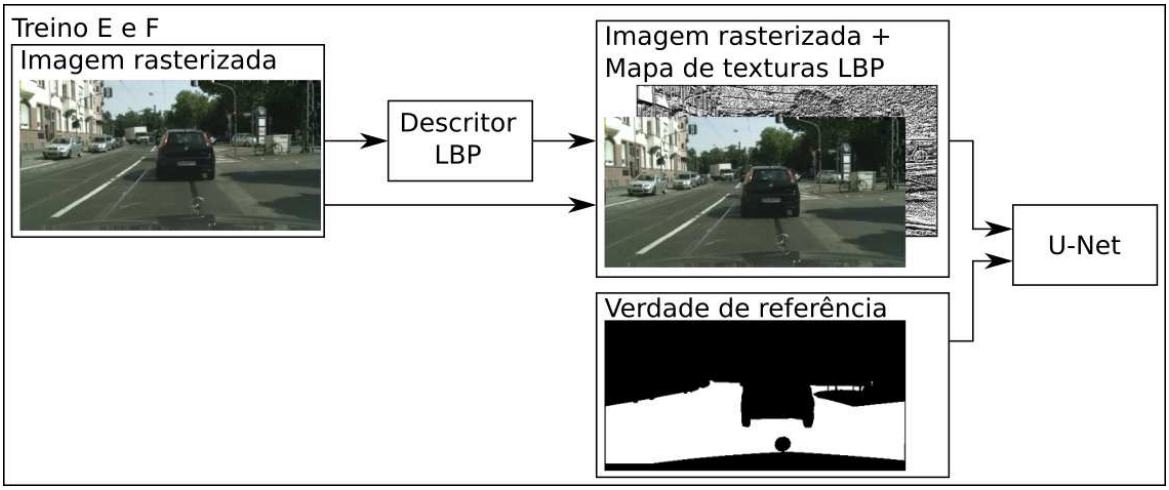
Os três testes mencionados no início desta Seção 4.5 são realizados. A performance deve ser muito próxima em todos os casos.

4.5.3 Ensaio 3: Treinamento com Mapa de Texturas LBP Concatenado

O terceiro ensaio tem uma abordagem diferente dos anteriores. O mapa de texturas LBP é concatenado a uma quarta camada na imagem rasterizada. O objetivo é que o modelo aprenda ao mesmo tempo tanto os padrões contidos na imagem rasterizada quanto no respectivo mapa de texturas LBP.

São realizados dois treinamentos distintos: Treino E e Treino F. No Treino E, são utilizadas apenas as imagens diurnas, enquanto no Treino F, são utilizadas imagens diurnas e noturnas. Ambos os treinos utilizam a mesma estrutura composta da imagem rasterizada com o mapa de texturas LBP respectivo concatenado, conforme mostra o diagrama da Figura 40

Figura 40 – Treino com imagem rasterizada e mapa de texturas LBP concatenado.



Fonte: De autoria própria.

Os três testes mencionados no início desta Seção 4.5 são realizados. Os resultados do primeiro teste devem ser muito semelhantes. No segundo teste, o Treino E deve sofrer com a degradação de performance pelas imagens noturnas não terem sido incluídas na etapa de treinamento. No entanto, o desempenho do Treino E neste segundo teste deve ser maior se comparado com o mesmo teste do Treino A. O terceiro teste é uma média ponderada dos resultados obtidos nos dois testes anteriores.

4.6 Métricas de Avaliação das Redes

A avaliação de desempenho é essencial para avaliar e otimizar qualquer modelo de Aprendizagem de Máquina e compará-lo com outros modelos. Em segmentação de imagens, Rainio, Teuho e Klén (2024) sugere a utilização de três métricas: Acurácia, Coeficiente de *Dice* e *Intersection over Union* (IoU).

A Acurácia, representada na equação (4.1), é a métrica de avaliação comum utilizada nos trabalhos relacionados estudados (YOUSRI; ELATTAR; DARWEESH, 2021) (JEBAMIKYOUS; KASHEF, 2021) (SARMAH; GOGOI; KALITA, 2022) (PHAM, 2021) (LAZUARDI et al., 2019),

$$\text{Acurácia} = \frac{VP + VN}{VP + FP + VN + FN} \ , \tag{4.1}$$

em que VP representa as predições verdadeiramente positivas, VN, as verdadeiramente negativas, FP, as falso positivas, e FN, as falso negativas. Essa métrica mensura todas as classes corretamente identificadas e é útil quando todas as classes têm igual importância.

O coeficiente de *Dice* (D) também é comumente conhecido como pontuação F1, e está representada pela equação (4.2),

$$D = \frac{2\,|X \cap Y|}{|X| + |Y|} \ , \tag{4.2}$$

em que X representa a área da máscara predita, e Y a área da máscara verdade. O coeficiente de *Dice* também pode ser expresso em termos das predições, como mostra a equação (4.3),

$$D = \frac{2VP}{2VP + FP + FN} \ , \tag{4.3}$$

em que VP representa as predições verdadeiramente positivas, FP, as falso positivas, e FN, as falso negativas.

A métrica IoU, também conhecida como Índice de *Jaccard*, está representada na equação (4.4),

$$\text{IoU} = \frac{|\text{X} \bigcap \text{U}|}{|\text{X} \bigcup \text{U}|} \ , \tag{4.4}$$

em que X representa a área da máscara predita, e Y a área da máscara verdade. O IoU também pode ser expresso em termos das predições, como na equação (4.5)

$$\text{IoU} = \frac{VP}{VP + FP + FN} \ , \tag{4.5}$$

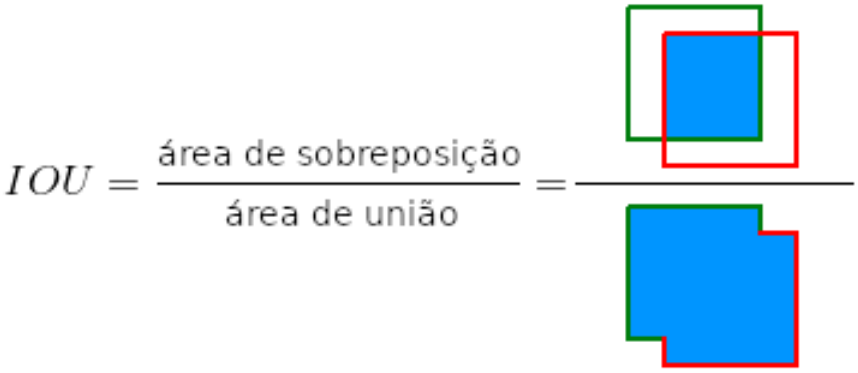
em que VP representa as predições verdadeiramente positivas, FP, as falso positivas, e FN, as falso negativas.

As duas métricas Coeficiente de *Dice* e IoU são diretamente correlacionadas, portanto não há necessidade de utilizar ambas as medidas pois o resultado seria diretamente proporcional. A diferença entre as duas é descrita em Müller, Soto-Rey e Kramer (2022),

apontando que o Coeficiente de *Dice* é uma média bem balanceada entre sensibilidade e precisão, enquanto a IoU penaliza com maior intensidade os erros de segmentação. Assim, por ter uma natureza mais rigorosa, apenas a IoU é calculada neste trabalho.

De forma mais intuitiva, a IoU calcula a sobreposição percentual entre a máscara verdade e a saída da predição, conforme demonstrado na Figura 41. Em outras palavras, o número de *pixels* comuns entre a máscara de predição e a máscara verdade é dividido pelo número total de *pixels* presentes em ambas as máscaras.

Figura 41 – Exemplo de IoU.



Fonte: traduzido de Padilla, Netto e Silva (2020).

Em segmentação semântica, em Rainio, Teuho e Klén (2024) é citado que a média IoU, representada na equação (4.6), é uma medida mais apropriada para uma métrica de avaliação em múltiplas imagens,

$$mIoU = \frac{1}{N}\sum_{1}^{N} IoU = \frac{1}{N}\sum_{1}^{N} \frac{|\mathrm{X} \bigcap \mathrm{U}|}{|\mathrm{X} \bigcup \mathrm{U}|} = \frac{1}{N}\sum_{1}^{N} \frac{VP}{VP + FP + FN} \ , \tag{4.6}$$

em que N representa o número total de imagens.

4.7 Considerações Parciais

Neste capítulo, foram detalhados os recursos de *hardware* e *software* empregados no desenvolvimento deste trabalho. Apesar de os recursos de *hardware* não serem gratuitos, os benefícios obtidos são justificados com a aceleração do treinamento do modelo e com a quantidade de memória disponibilizada. Além disso, optou-se por utilizar bibliotecas de código aberto que oferecem um conjunto completo de ferramentas para o pré-processamento de imagens, a implementação e avaliação dos modelos de redes neurais. Dessa forma, este trabalho pode contribuir na área científica com o método de treinamento da *U-Net*, especificamente, na utilização do mapa de texturas LBP para treinamento do modelo.

São apresentados três ensaios que visam avaliar o impacto da incorporação de mapas de texturas LBP no desempenho da tarefa de segmentação semântica de pavimentações asfálticas. Em cada ensaio são descritos dois métodos de treinamentos diferentes, cujos resultados serão comparados no capítulo seguinte. O objetivo é verificar se a utilização de mapas LBP confere ao modelo uma capacidade maior de generalização para aumentar a precisão em relação ao treinamento tradicional, onde se utiliza imagens rasterizadas.

O próximo capítulo irá apresentar os ensaios realizados com diferentes tipos de treinamento para comprovar e validar a efetividade do treinamento da rede *U-Net* com os mapas de texturas LBP. Os resultados obtidos são registrados e analisados, culminando em uma comparação com os resultados de outros estudos na área.

5 RESULTADOS E ANÁLISES

Neste capítulo são apresentados os resultados dos ensaios descritos na Seção 4.5. A arquitetura *U-Net*, apresentada na Subseção 4.4.2, foi escolhida para realizar a segmentação semântica de pavimentações asfálticas. A estrutura é praticamente a mesma para todos os ensaios, e é bastante semelhante ao do modelo publicado em Ronneberger, P.Fischer e Brox (2015), mas com algumas pequenas diferenças nas dimensões dos tensores.

No caso deste trabalho, a entrada do codificador tem tamanho de `512x256` com três camadas se for usada uma imagem rasterizada, ou uma única camada se for usado um mapa de texturas LBP, ou quatro camadas se o mapa de texturas LBP for concatenado à imagem rasterizada. O recurso de *padding* é adicionado para manter o mesmo tamanho após o processo de convolução, e após cada camada de agrupamento, o tamanho passado para a próxima camada é dividido por `2`. Conforme pode ser inferido, todos os processos realizados pela camada de agrupamento no codificador que reduzem as dimensões dos tensores precisam ser revertidos no decodificador através da convolução transposta.

Em todos os ensaios também foram utilizadas as duas bases de dados citadas na Seção 4.2: *Cityscapes* (CORDTS et al., 2016) e ACDC (SAKARIDIS; DAI; GOOL, 2021). A base de dados *Cityscapes* contém `2974` imagens para treinamento e `500` imagens para testes, todas tiradas durante o dia em diversas cidades da Europa; enquanto a base de dados ACDC possui `400` imagens para treinamento, e `106` imagens para testes, todas tiradas durante a noite, também em diversas cidades da Europa.

O ensaio 1 visa analisar o impacto da utilização dos mapas de textura LBP no treinamento da *U-Net* para segmentação de imagens em diferentes condições de iluminação. Para isso, a *U-Net* é treinada utilizando duas abordagens distintas: com imagens rasterizadas e com mapas de textura LBP extraídos das imagens.

As performances dos treinamentos são avaliadas em três testes: com imagens diurnas, com imagens noturnas e com imagens diurnas e noturnas. Os resultados são comparados, e observa-se que a utilização dos mapas de textura LBP, devido à sua característica de invariância em relação à iluminação, é capaz de aprimorar a performance no caso do segundo teste com imagens noturnas. Isto é, o treinamento com mapas de texturas LBP atua melhor em cenários noturnos não incluídos no treinamento.

O ensaio 2 é realizado como controle experimental. A inclusão de imagens noturnas na etapa de treinamento faz com que a segmentação tenha a performance melhorada nesses casos. Assim como no ensaio 1, a *U-Net* é treinada de duas formas: com imagens rasterizadas e com mapas de textura LBP; e também é submetida aos mesmos três testes,

onde todos os resultados obtidos são muito semelhantes.

O ensaio 3 concatena o mapa de texturas LBP em uma quarta camada na imagem rasterizada. O objetivo é permitir que, durante o treinamento, os padrões estruturais e texturas do mapa de texturas LBP sejam expostas junto com os da imagem rasterizada. Novamente são adotados dois tipos de treinamento: um com imagens diurnas e outro com imagens diurnas e noturnas. Os mesmos testes aplicados nos ensaios anteriores são aplicados a este ensaio, e verifica-se que a acurácia do treinamento com imagens diurnas e noturnas é muito próxima das obtidas no ensaio 2. No entanto, o treinamento com imagens diurnas apresenta desempenho melhor que o treinamento com imagens rasterizadas diurnas do ensaio 1, reforçando a tese de que os mapas de texturas LBP aprimoram a capacidade de generalização do modelo.

5.1 Ensaio 1: Teste de Invariância de Iluminação do Operador LBP

A *U-Net* é treinada de duas formas distintas: Treino A e Treino B. Um detalhe importante deste ensaio é que em ambos os treinos, são usadas apenas imagens diurnas no processo de treinamento. O Treino A é realizado com as imagens rasterizadas de três camadas de cores no padrão RGB, enquanto o Treino B é realizado com o mapa de texturas LBP das imagens - é importante observar que a entrada do Treino B teve de ser ajustada para receber o mapa de texturas LBP que contém apenas uma camada. A mudança no número de camadas nas entradas não altera, entretanto, o número de parâmetros totais dos modelos descritos na Tabela 4.

Os parâmetros de treinamento estão dispostos na Tabela 5. As curvas de Custo e Acurácia geradas ao longo de toda etapa de treinamento estão representadas na Figura 42, e os valores finais estão registrados na Tabela 6.

Tabela 5 – Parâmetros de treinamento para os Treinos A e B.

Treino	Tamanho do lote (*batch*)	Épocas	Otimizador	Taxa de Aprendizagem	Função de Custo	Epsilon	Momentum
A	16	30	ADAM	$0,2e^{-3}$	Entropia Cruzada Binária	0,02	0,90
B	16	30	ADAM	$0,2e^{-3}$	Entropia Cruzada Binária	0,02	0,90

Os hiperparâmetros `epsilon` e `momentum` de todas as funções `BatchNormalization` foram configurados para minimizar quebras na continuidade das curvas de Custo e de Acurácia. O valor de `epsilon = 0,02` previne instabilidades na normalização por divisão por zero, enquanto o valor de `momentum = 0,90` suaviza as atualizações dos parâmetros de

Tabela 6 – Valores finais de Custo e Acurácia para os Treinos A e B.

Treino	**Custo (Treino) (%)**	**Acurácia (Treino) (%)**	**Custo (Validação) (%)**	**Acurácia (Validação) (%)**
A	2,21	99,17	7,58	97,86
B	1,36	99,45	10,32	97,54

normalização, impedindo mudanças abruptas nas curvas de treinamento. O código de configuração das funções `BatchNormalization` está explicitado no Código 9.

Código 9 – Configuração da função BatchNormalization para os Ensaios 1 e 2

```
1 BatchNormalization(epsilon=0.02, momentum=0.90)
```

Os gráficos da Figura 42 foram dimensionados propositalmente para que possam ser mais facilmente comparados com os gráficos dos ensaios seguintes. A Figura 43 faz um *zoom* no eixo vertical das curvas da Figura 42 para que os detalhes possam ser melhor observados.

5.1.1 Teste 1: Teste com Imagens Diurnas

Os Treinos A e B são submetidos ao primeiro teste com imagens diurnas. A Acurácia e a média IoU estão registradas na Tabela 7. Ambos os treinos apresentaram resultados altos nas duas métricas.

Tabela 7 – Média IoU e Acurácia dos Treinos A e B.

Treino	**Imagens**	**Treino**	**Teste**	**Média IoU (%)**	**Acurácia (%)**
A	RGB	Dia	Dia	89,5919	97,0260
B	LBP	Dia	Dia	89,3334	96,9433

A Figura 44 exibe alguns exemplos de predições realizadas pelos Treinos A e B.

5.1.2 Teste 2: Teste com Imagens Noturnas

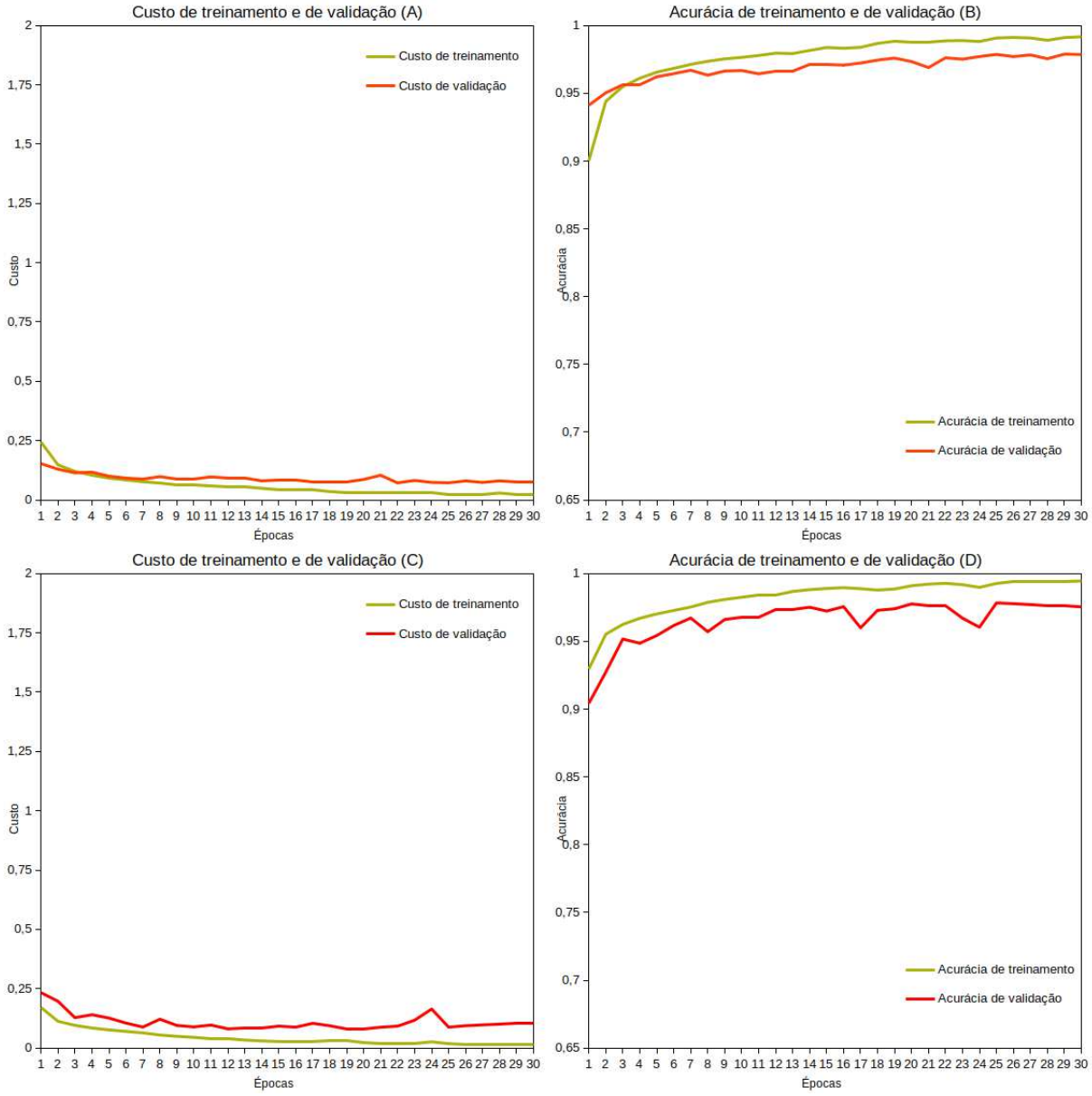
Apesar de os Treinos A e B terem sido realizados utilizando apenas imagens diurnas, ambos foram submetidos a testes com as imagens da base de dados ACDC, que contém apenas imagens noturnas. A Acurácia e a média IoU estão registradas na Tabela 8.

Tabela 8 – Média IoU e Acurácia dos Treinos A e B testados com a base de dados ACDC.

Treino	**Imagens**	**Treino**	**Teste**	**Média IoU (%)**	**Acurácia (%)**
A	RGB	Dia	Noite	61,4172	83,2596
B	LBP	Dia	Noite	81,0013	91,6363

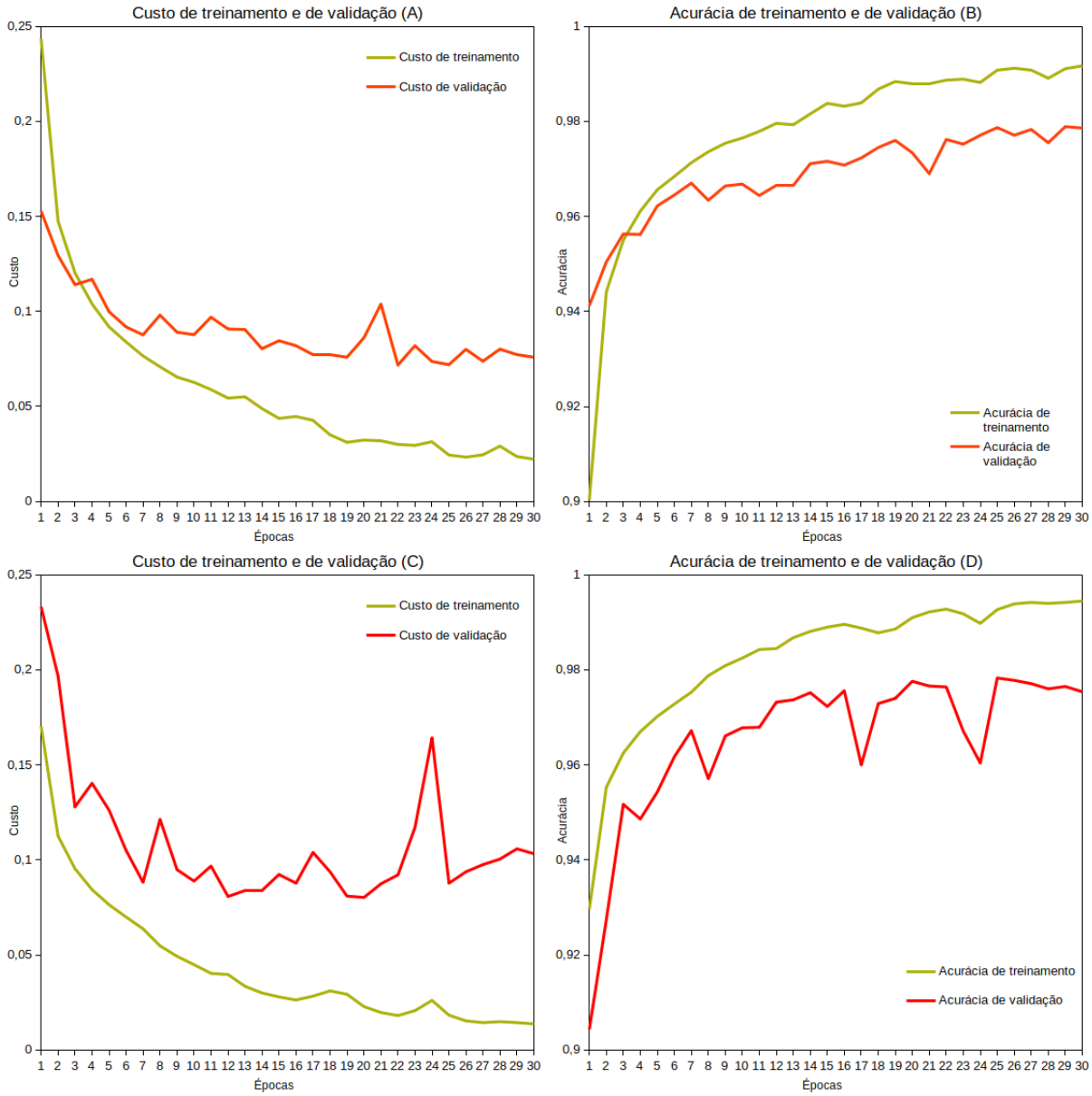
Os resultados do segundo teste mostram uma melhor performance do Treino B em relação ao Treino A. Isso pode ser justificado pelas características dos mapas de textura LBP, como invariância dos mapas de textura LBP em relação às variações de tons de

Figura 42 – (A) e (B) Custo e Acurácia do Treino A. (C) e (D) Custo e Acurácia do Treino B.



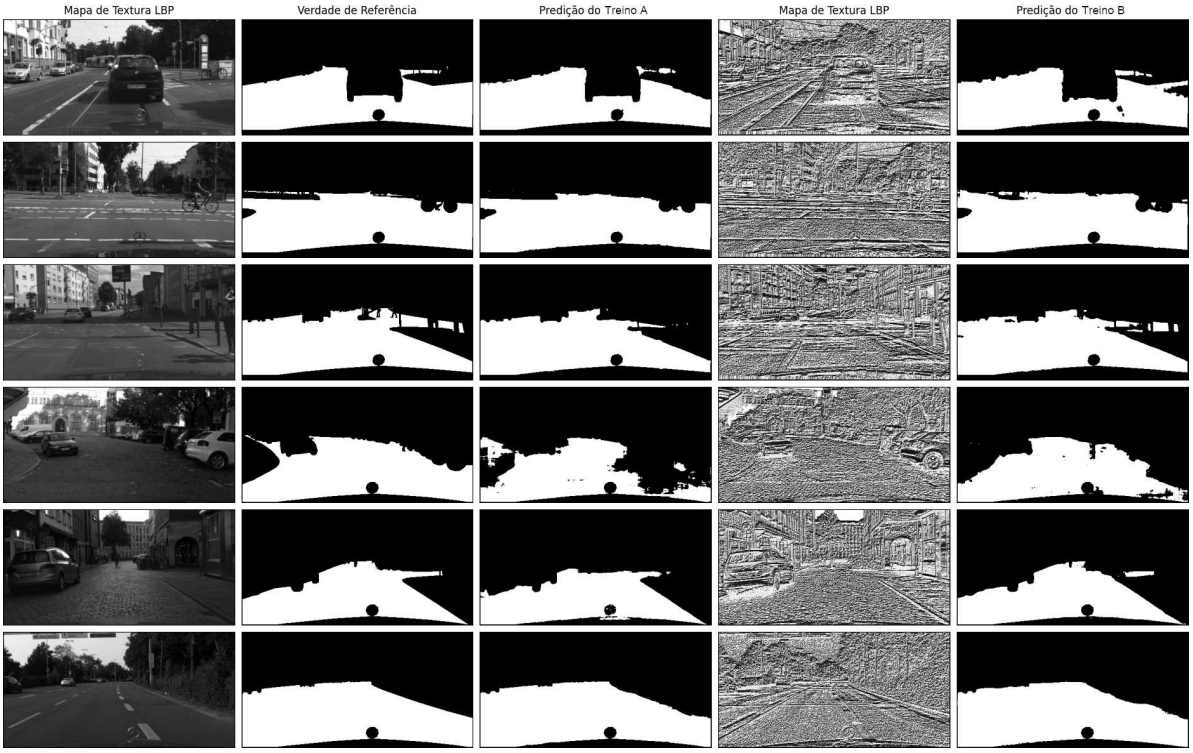
Fonte: De autoria própria.

Figura 43 – (A) e (B) Gráfico em *zoom* para Custo e Acurácia do Treino A. (C) e (D) Gráfico amplificado para Custo e Acurácia do Treino B.



Fonte: De autoria própria.

Figura 44 – Exemplos de predições dos Treinos A e B para imagens diurnas.



Fonte: De autoria própria.

cinza (OJALA; PIETIKäINEN; HARWOOD, 1996), e como a capacidade de ressaltarem os padrões estruturais e de textura (ZHANG et al., 2017) (WU; ZHOU; LI, 2022). Ao remover os efeitos de iluminação da imagem e destacar as bordas, cantos e superfícies dos objetos, o modelo deve aprender os padrões estruturais e texturas das pavimentações asfálticas com mais facilidade. Por conseguinte, a identificação de tais padrões em imagens noturnas também deve ser facilitada, mesmo que o Treino B não tenha sido treinado com imagens noturnas.

A Figura 45 exibe alguns exemplos de diferenças entre as predições dos Treinos A e B de forma gráfica.

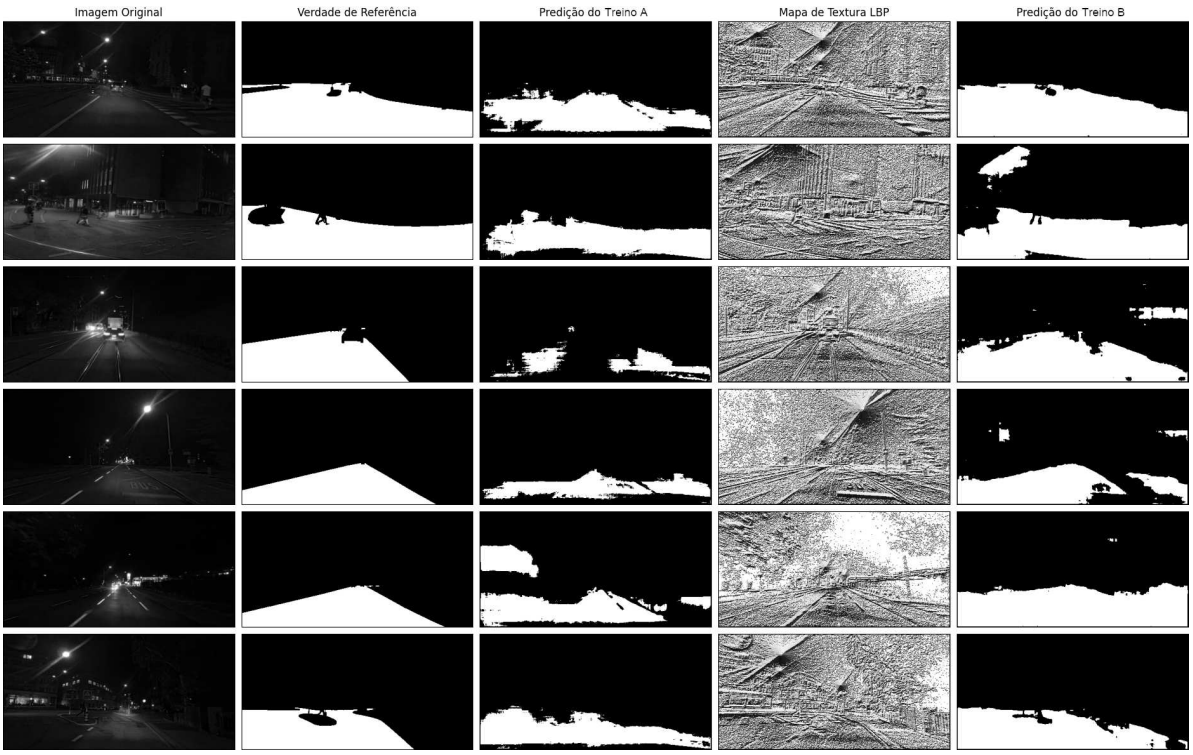
5.1.3 Teste 3: Teste com Imagens Diurnas e Noturnas

Os Treinos A e B também foram testados combinando as bases de dados de testes. A Acurácia e a média IoU estão registradas na Tabela 9.

Tabela 9 – Média IoU e Acurácia dos Treinos A e B testados com as bases de dados *Cityscapes* e ACDC.

Treino	**Imagens**	**Treino**	**Teste**	**Média IoU (%)**	**Acurácia (%)**
A	RGB	Dia	Dia e Noite	79,6938	94,6180
B	LBP	Dia	Dia e Noite	84,1281	96,0150

Figura 45 – Exemplos de predições dos Treinos A e B para imagens noturnas.



Fonte: De autoria própria.

5.2 Ensaio 2: Comparação de Performance do Operador LBP

A *U-Net* é treinada de duas formas distintas: Treino C e Treino D. Um detalhe que difere este ensaio do anterior é que são usadas imagens diurnas e noturnas no processo de treinamento. O treino C é realizado com as imagens rasterizadas de três camadas de cores no padrão RGB, enquanto o Treino D, com o mapa de texturas LBP das imagens.

Os parâmetros de treinamento estão dispostos na Tabela 10. Os hiperparâmetros `epsilon` e `momentum` das funções `BatchNormalization` também foram configurados com os mesmos valores do ensaio anterior: `epsilon = 0,02`, e `momentum = 0,90`. O código é o mesmo do Código 9.

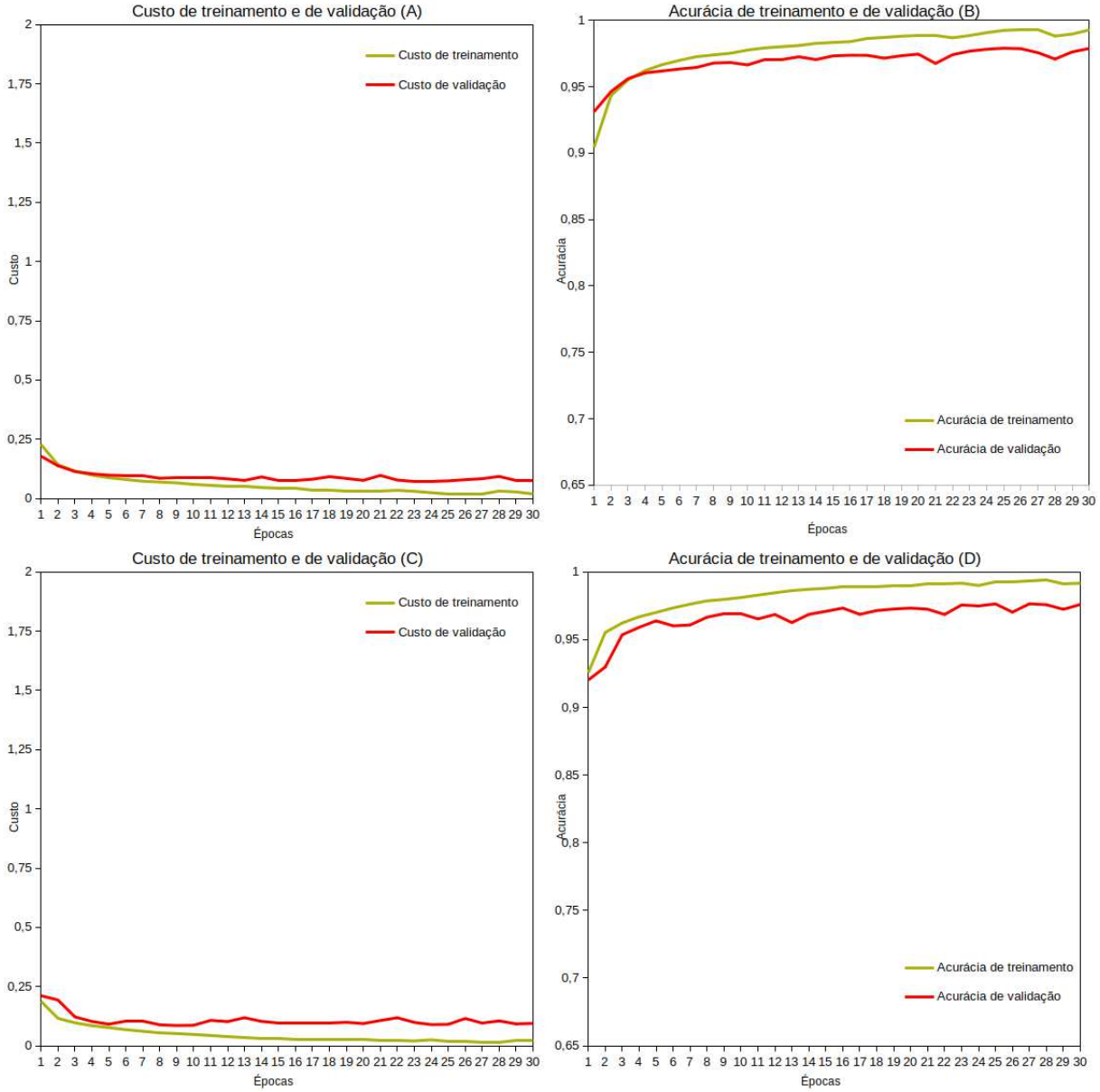
Tabela 10 – Parâmetros de treinamento para os Treinos C e D.

Treino	Tamanho do lote (*batch*)	Épocas	Otimizador	Taxa de Aprendizagem	Função de Custo	Epsilon	Momentum
C	16	30	ADAM	$0,2e^{-3}$	Entropia Cruzada Binária	0,02	0,90
D	16	30	ADAM	$0,2e^{-3}$	Entropia Cruzada Binária	0,02	0,90

As curvas de Custo e Acurácia geradas na etapa de treinamento estão dispostas

na Figura 46. Os gráficos da Figura 46 foram colocados na mesma escala que os do ensaio anterior para facilitar a comparação. No entanto, a Figura 47 faz um *zoom* do eixo vertical das curvas da Figura 46 para que os detalhes possam ser melhor observados.

Figura 46 – A) e (B) Custo e Acurácia do Treino C. (C) e (D) Custo e Acurácia do Treino D.



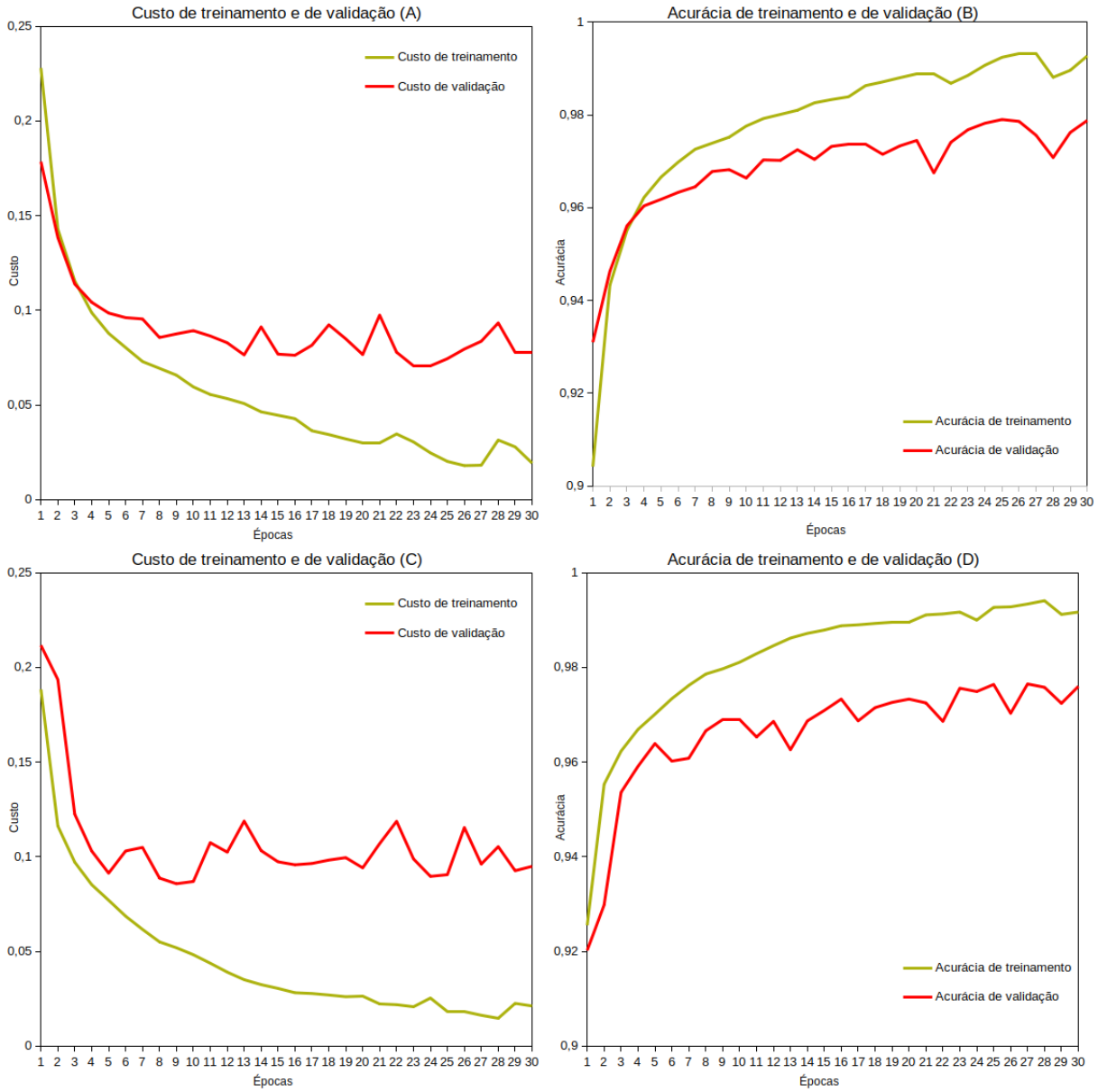
Fonte: De autoria própria.

Os valores finais de Custo e Acurácia estão registrados na Tabela 11.

Tabela 11 – Valores finais de Custo e Acurácia para os Treinos C e D.

Treino	**Custo (Treino) (%)**	**Acurácia (Treino) (%)**	**Custo (Validação) (%)**	**Acurácia (Validação) (%)**
C	1,92	99,27	7,76	97,88
D	2,11	99,17	9,49	97,60

Figura 47 – A) e (B) Gráfico em *zoom* para Custo e Acurácia do Treino C. (C) e (D) Gráfico amplificado para Custo e Acurácia do Treino D.



Fonte: De autoria própria.

Neste ensaio, os Treinos C e D foram submetidos a três testes: predição com imagens de validação diurnas, predição com imagens de validação noturnas, e predição com imagens de validação diurnas e noturnas.

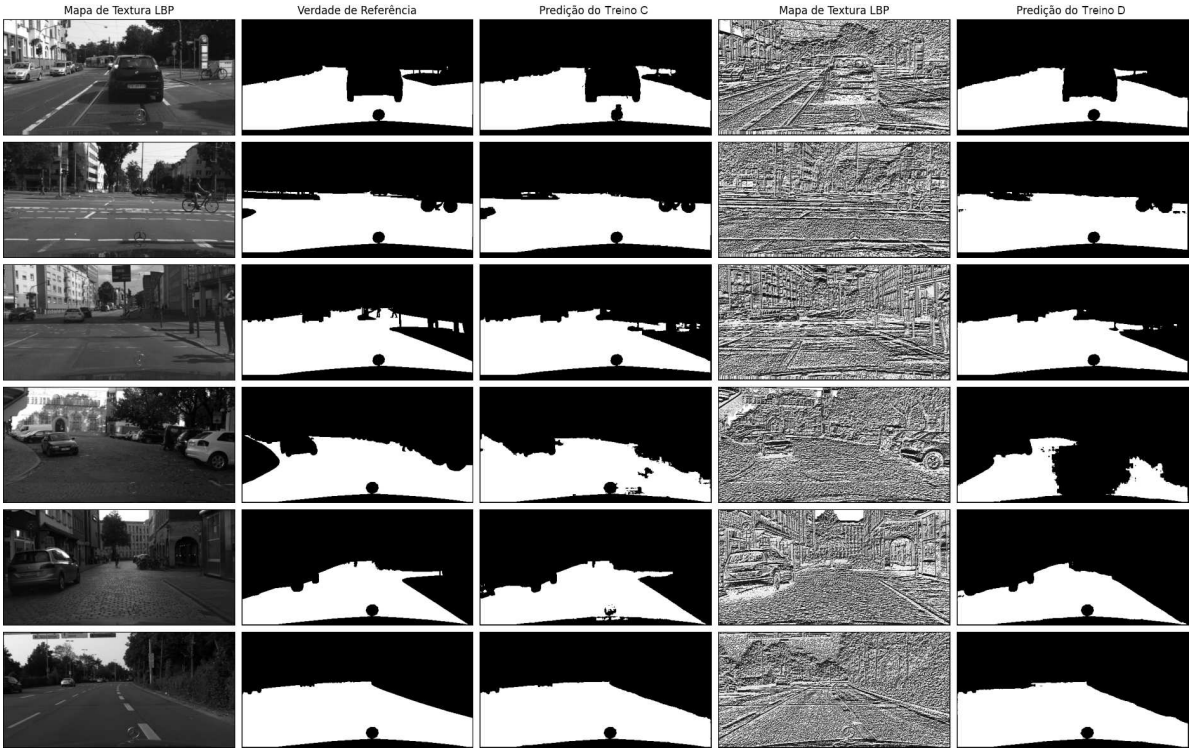
Este ensaio tem caráter de controle experimental. O objetivo é validar o bom funcionamento do modelo quando treinado com ambos os conjuntos de imagens diurnos e noturnos. De fato, os resultados obtidos apresentaram valores muito semelhantes em todos os casos, conforme a Tabela 12 que registra a Acurácia e a média IoU.

Tabela 12 – Média IoU e Acurácia dos Treinos C e D.

Treino	**Imagens**	**Treino**	**Teste**	**Média IoU (%)**	**Acurácia (%)**
C	RGB	Dia e Noite	Dia	89,8027	97,0930
C	RGB	Dia e Noite	Noite	93,5216	97,5151
C	RGB	Dia e Noite	Dia e Noite	88,1733	97,1669
D	LBP	Dia e Noite	Dia	87,7668	97,0817
D	LBP	Dia e Noite	Noite	88,3482	95,3071
D	LBP	Dia e Noite	Dia e Noite	86,7408	96,7713

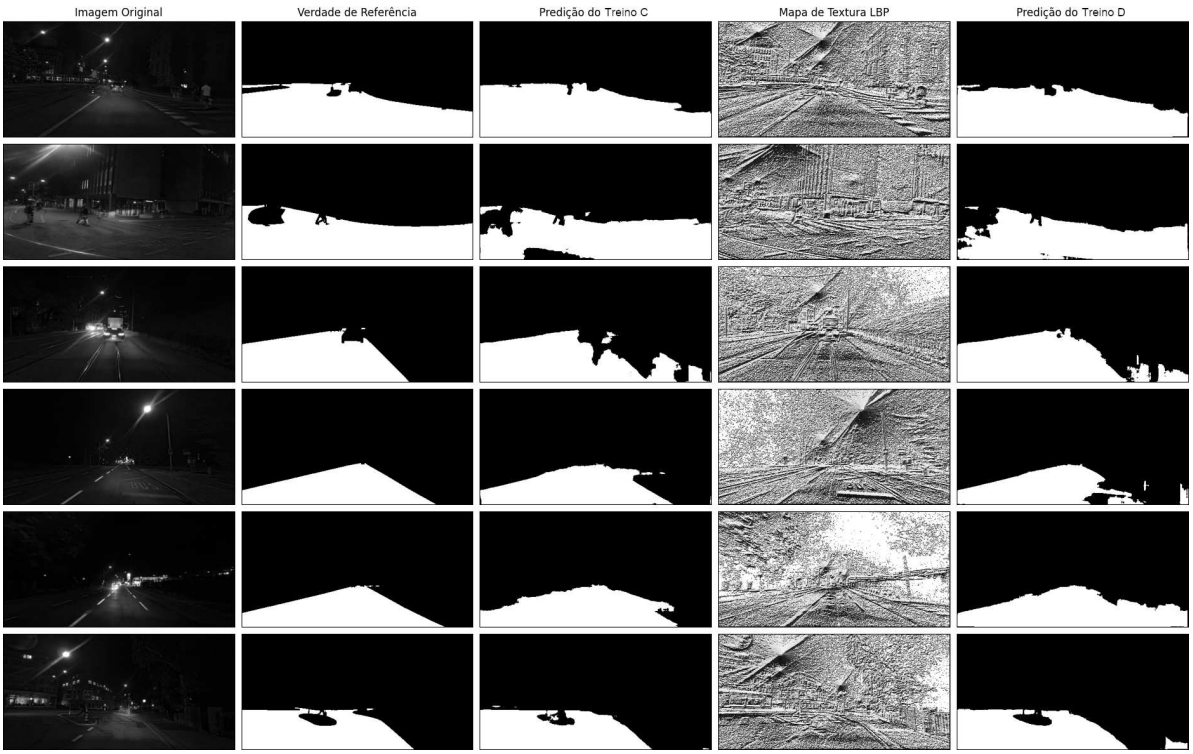
As Figuras 48 e 49 comparam as predições dos Treinos C e D nas condições de dia e noite, respectivamente.

Figura 48 – Exemplos de predições dos Treinos C e D para imagens diurnas.



Fonte: De autoria própria.

Figura 49 – Exemplos de predições dos Treinos C e D para imagens noturnas.



Fonte: De autoria própria.

5.3 Ensaio 3: Treinamento com Mapa de Texturas Concatenado

A *U-Net* é treinada de duas formas distintas: Treino E e Treino F. Este ensaio tem uma abordagem diferente dos anteriores, onde o mapa de texturas LBP é concatenado em uma quarta camada na imagem - essa estrutura é utilizada em ambos os treinos. O Treino E é realizado exclusivamente com imagens diurnas, enquanto o Treino F é realizado com imagens diurnas e noturnas.

Os parâmetros de treinamento estão dispostos na Tabela 13. As curvas de Custo e Acurácia geradas na etapa de treinamento estão dispostas na Figura 50, e os valores finais de Custo e Acurácia estão registrados na Tabela 14.

Tabela 13 – Parâmetros de treinamento para os Treinos E e F.

Treino	Tamanho do lote (*batch*)	Épocas	Otimizador	Taxa de Aprendizagem	Função de Custo	Epsilon	Momentum
E	16	22	ADAM	$0,2e^{-3}$	Entropia Cruzada Binária	0,005	0,995
F	16	40	ADAM	$0,2e^{-3}$	Entropia Cruzada Binária	0,01	0,99

Tabela 14 – Valores finais de Custo e Acurácia para os Treinos E e F.

Treino	**Custo (Treino) (%)**	**Acurácia (Treino) (%)**	**Custo (Validação) (%)**	**Acurácia (Validação) (%)**
E	`1,90`	`99,27`	`9,69`	`97,22`
F	`1,09`	`99,57`	`10,31`	`97,77`

Os hiperparâmetros `epsilon` e `momentum` das funções `BatchNormalization` também foram configurados com os seguintes valores: no Treino E foram utilizados `epsilon = 0,005`, e `momentum = 0,995`, enquanto no Treino F foram utilizados `epsilon = 0,01`, e `momentum = 0,99`. O código para configuração das funções `BatchNormalization` estão expostos nos Códigos 10 e 11, para os Treinos E e F, respectivamente.

Código 10 – Configuração da função BatchNormalization para o Treino E

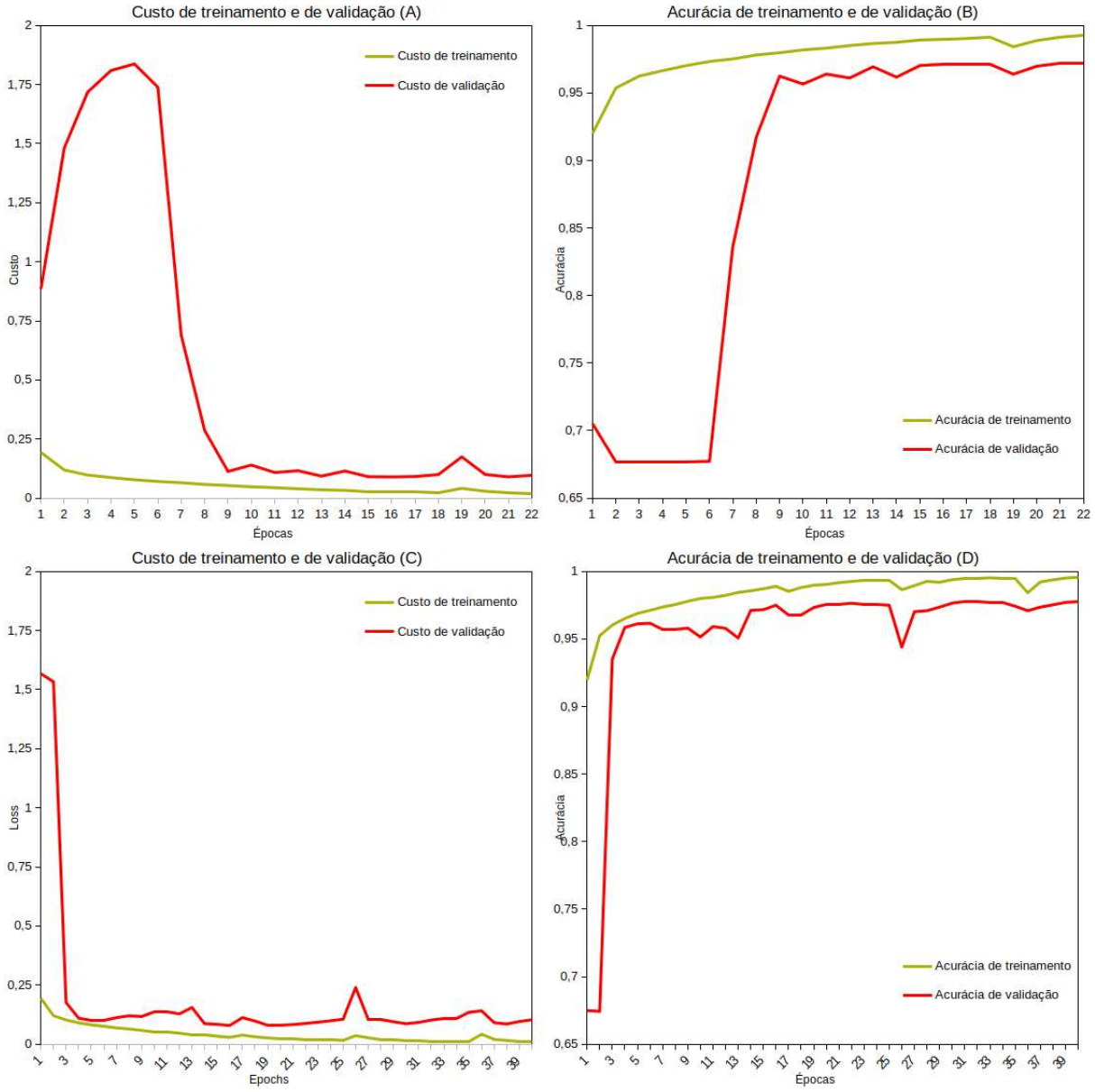
```
1 BatchNormalization(epsilon=0.005, momentum=0.995)
```

Código 11 – Configuração da função BatchNormalization para o Treino F

```
1 BatchNormalization(epsilon=0.01, momentum=0.99)
```

Pode-se observar que, na Figura 50, o formato das curvas de Custo e de Acurácia na etapa de validação estão diferentes dos formatos observados nos ensaios anteriores. Isso pode ter ocorrido pelo fato de a quarta camada ter agregado maior complexidade às amostras de treinamento e o aprendizado se tornar mais difícil. Como o Treino E é realizado apenas com imagens diurnas, a quantidade total de amostras de treinamento é

Figura 50 – (A) e (B) Custo e Acurácia do Treino E. (C) e (D) Custo e Acurácia do Treino F.

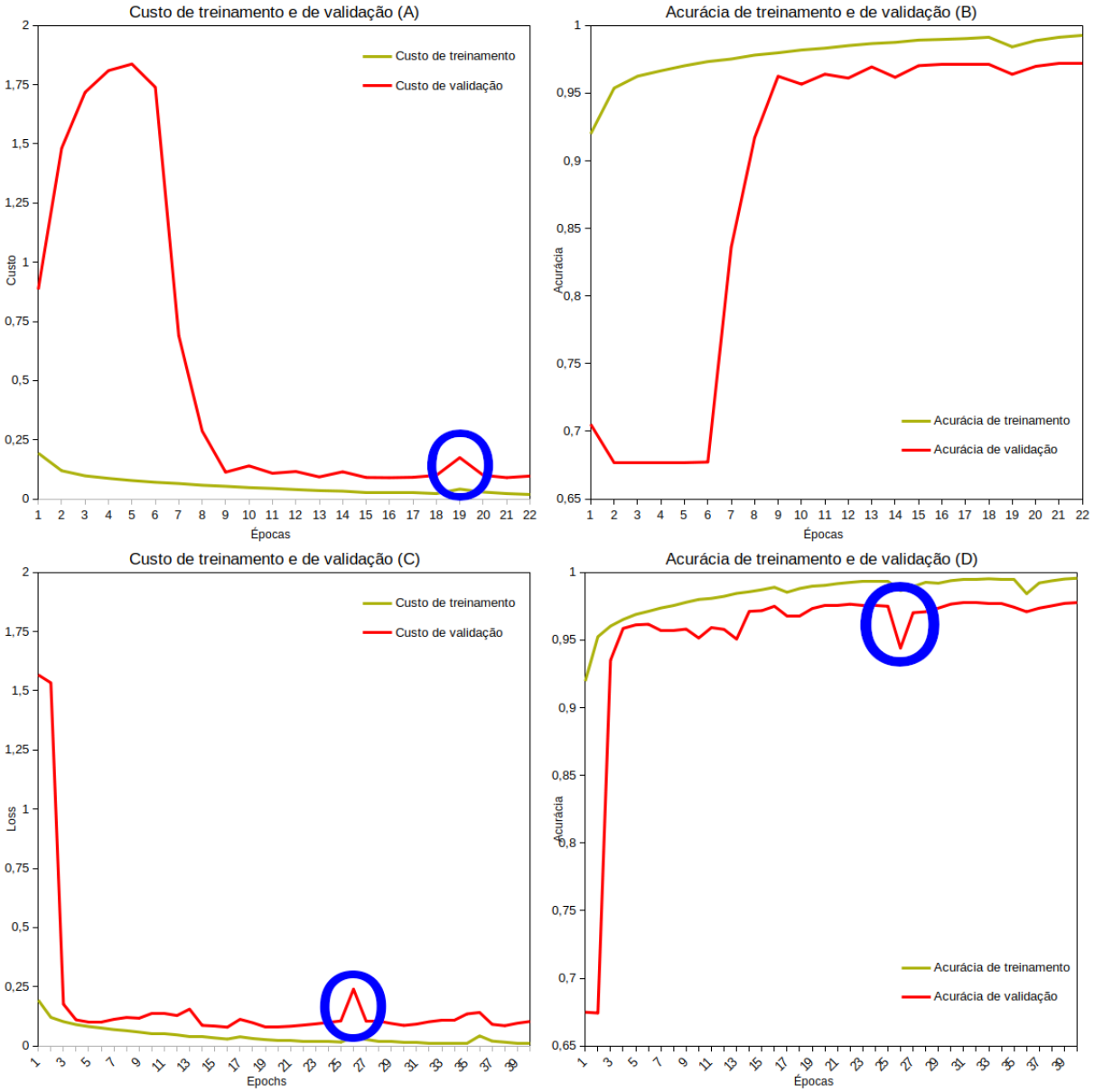


Fonte: De autoria própria.

reduzida em 400 imagens em relação ao Treino F. Por essa razão, observa-se que as curvas de Custo e de Acurácia na etapa de validação do Treino F convergem mais rapidamente nas primeiras épocas em direção aos valores das curvas de treinamento respectivas.

Pode-se notar também que nas curvas da Figura 50 existem algumas pequenas quebras acentuadas nas curvas de validação, conforme destacado em círculos azuis na Figura 51.

Figura 51 – Quebras na continuidade destadas em círculos azuis.



Fonte: De autoria própria.

Essas quebras podem ter inúmeras causas, por exemplo: baixo número de amostras de validação, distribuição das amostras de validação, ruídos nas amostras de validação, e otimização de hiperparâmetros. Embora seja possível alterar ainda mais todos os pontos

citados para deixar essas curvas com formatos mais suaves e convergência mais rápida, os melhores resultados com as menores quebras de continuidade obtidos neste trabalho foram com as configurações apresentadas.

Neste ensaio, os Treinos E e F foram submetidos a três testes: predição com imagens de validação diurnas, predição com imagens de validação noturnas, e predição com imagens de validação diurnas e noturnas.

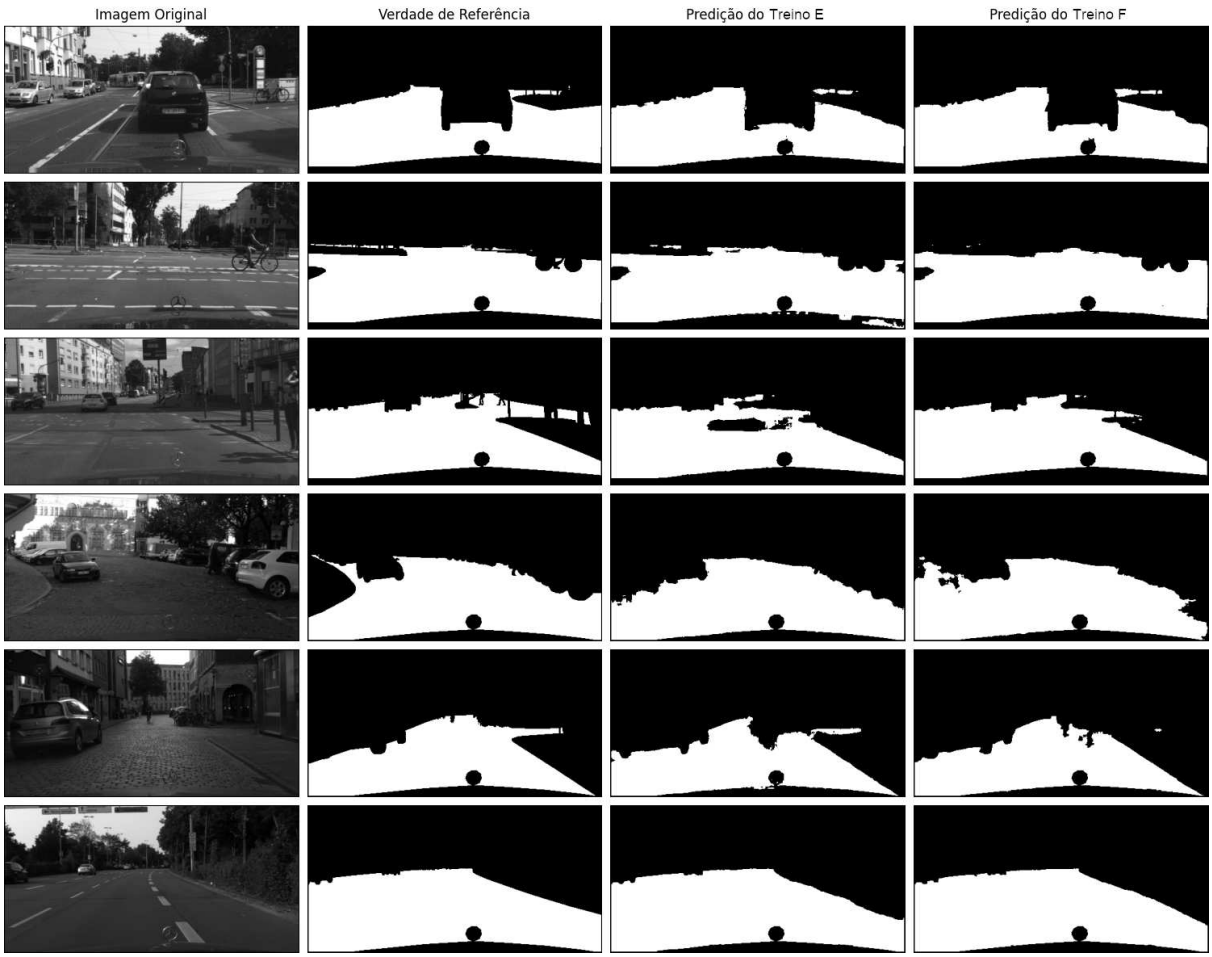
Ao concatenar o mapa de texturas LBP a uma quarta camada na imagem, as informações estruturais e texturas da imagem devem ser reforçadas. O modelo deve aprender da imagem rasterizada e do mapa de texturas ao mesmo tempo, o que deve deixar mais robusta a identificação de padrões durante os testes. A Acurácia e a média IoU estão registradas na Tabela 15.

Tabela 15 – Média IoU e Acurácia dos Treinos E e F.

Treino	**Imagens**	**Treino**	**Teste**	**Média IoU (%)**	**Acurácia (%)**
E	RGB e LBP	Dia	Dia	88,9080	96,8062
E	RGB e LBP	Dia	Noite	71,9142	87,6108
E	RGB e LBP	Dia	Dia e Noite	81,4759	95,1977
F	RGB e LBP	Dia e Noite	Dia	90,2042	97,2200
F	RGB e LBP	Dia e Noite	Noite	91,7143	96,7658
F	RGB e LBP	Dia e Noite	Dia e Noite	88,0763	97,1405

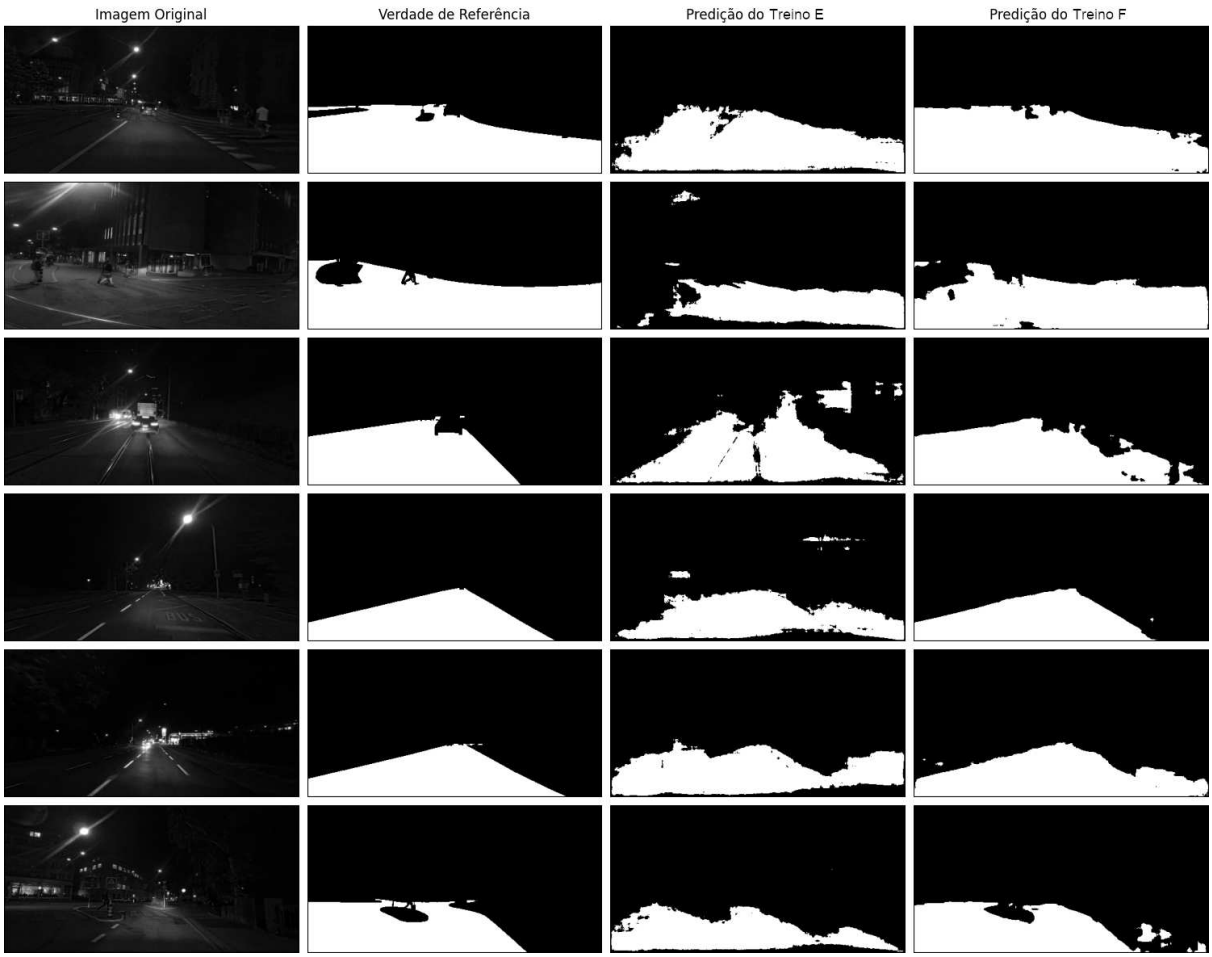
As Figuras 52 e 53 exibem as predições dos Treinos E e F nas condições de dia e noite, respectivamente.

Figura 52 – Exemplos de predições dos Treinos E e F para imagens diurnas.



Fonte: De autoria própria.

Figura 53 – Exemplos de predições dos Treinos E e F para imagens noturnas.



Fonte: De autoria própria.

5.4 Comparações com Trabalhos Relacionados

As métricas mais populares para análise de performance de segmentação semântica são dadas pela Acurácia e pela média IoU, no entanto, nem todos os trabalhos apresentam essas duas métricas. A Tabela 16 apresenta um comparativo entre os resultados obtidos neste trabalho e os resultados reportadas em trabalhos relacionados.

Tabela 16 – Comparação de performance.

Referência	**Modelo**	**Média IoU (%)**	**Acurácia) (%)**
Yousri, Elattar e Darweesh (2021) (Dia)	ResUNet++	96,7	99,1
	ResUNet	96,2	99,0
	U-Net	95,9	99,0
	Modified SegNet	96,6	98,6
	SegNet	94,2	98,2
Yousri, Elattar e Darweesh (2021) (Noite)	ResUNet++	85,5	98,0
	ResUNet	79,8	97,1
	U-Net	80,3	97,9
	Modified SegNet	73,3	93,4
	SegNet	72,7	92,0
Jebamikyous e Kashef (2021)	U-Net ReLU	-	89,74
	Small U-Net ReLU	-	87,96
	Small U-Net Leaky ReLU	-	87,96
	Long U-Net (Dropout=0,7)	-	89,85
	Long U-Net (Dropout=0,5)	-	89,13
	SegNet	-	85,37
	SegNet (Dropout=50)	-	83,97
	FCN-16	-	84,92
	FCN-16 (Dropout=50)	-	85,85
	FCN-8	-	86,03
	FCN-8 (Dropout=50)	-	87,00
Sarmah, Gogoi e Kalita (2022)	U-Net	-	90,60
	U-Net VGG16 (Pre-trained)	-	91,53
	U-Net VGG16	-	89,45
	U-Net VGG19 (Pre-trained)	-	90,04
	U-Net VGG19	-	89,59
Pham (2021)	PSPNet	64,75	93,75
	FCN	45,17	87,61
	SegNet	53,79	92,89
Lazuardi et al. (2019)	U-Net Model 1	-	94,99
	U-Net Model 2	-	93,11
	U-Net Model 3	-	91,94
	U-Net Model 4	-	92,14
	U-Net Model 5	-	90,71
U-Net (Dia)	Treino A	89,5919	97,0260
	Treino B	89,3334	96,9433
	Treino C	89,8027	97,0930
	Treino D	87,7668	97,0817
	Treino E	88,9080	96,8062
	Treino F	90,2042	97,2200
U-Net (Noite)	Treino A	61,4172	83,2596
	Treino B	81,0013	91,6363
	Treino C	93,5216	97,5151
	Treino D	88,3482	95,3071
	Treino E	71,9142	87,6108
	Treino F	91,7143	96,7658
U-Net (Dia e Noite)	Treino A	79,6938	94,6180
	Treino B	84,1281	96,0150
	Treino C	88,1733	97,1669
	Treino D	86,7408	96,7713
	Treino E	81,4759	95,1977
	Treino F	88,0763	97,1405

Todos os trabalhos listados na Tabela 16 foram apresentados na Seção 3.1. A comparação entre os trabalhos deve considerar também outros parâmetros, como a base de dados utilizada e a quantidade de classes que o modelo é capaz de categorizar. Esses parâmetros estão dispostas na Tabela 17.

Tabela 17 – Bases de dados e quantidade de classes dos trabalhos relacionados

Referência	**Base de Dados**	**Número de classes**
Yousri, Elattar e Darweesh (2021)	*nuScenes*	2
Jebamikyous e Kashef (2021)	*CamVid*	32
Sarmah, Gogoi e Kalita (2022)	*CamVid*	32
Pham (2021)	*Cityscapes*	19
Lazuardi et al. (2019)	*Cityscapes*	5
U-Net	*Cityscapes* e ACDC	2

O primeiro trabalho de Yousri, Elattar e Darweesh (2021) apresentou os melhores resultados em termos de Acurácia e de Média IoU, superando os demais estudos. Os autores propõem um algoritmo automatizado de duas etapas para gerar automaticamente máscaras binárias de pavimentos asfálticos em imagens rasterizadas: identificação da região adaptativa e aprimoramento de características das faixas de trânsito. A identificação da região adaptativa tem a função de estimar a linha do horizonte utilizando o método de distribuição médio vertical, identificar as bordas da pavimentação asfáltica utilizando uma variante do método da Trasformada de Hough, e transformar a perspectiva da imagem para vista aérea.

Na etapa de aprimoramento de características, a imagem é convertida para o espaço de cores HSV, e a operação morfológica *top-hat* é aplicada para realçar as regiões mais brilhantes, correspondentes às faixas de trânsito. Essas regiões são utilizadas como máscaras iniciais para a segmentação, que serve de base para o treinamento de cinco modelos de redes neurais.

É importante notar que o algoritmo, em sua configuração atual, assume que as imagens de entrada apresentam a pavimentação asfáltica predominantemente livre de outros objetos. Essa limitação pode afetar a generalização do modelo para cenários mais complexos. A alta taxa de acerto relatada pelos autores pode estar diretamente relacionada a essa restrição, uma vez que a ausência de outros objetos facilita a identificação das faixas de trânsito.

É interessante observar também que as aplicações dos trabalhos citados apresentam diferentes prioridades. Em Pham (2021), a necessidade de processamento em tempo real impulsionou o desenvolvimento de um modelo mais rápido, mesmo que isso significasse uma leve redução na precisão. Por outro lado, Lazuardi et al. (2019) se concentrou em modelos compactos e eficientes em termos de memória, capazes de operar em sistemas embarcados com recursos computacionais limitados.

Os resultados obtidos neste trabalho superam o de todos os outros estudos restan-

tes. Essa superioridade pode ser atribuída, em parte, à abordagem de classificação binária adotada neste estudo, em contraste com a classificação multiclasse presente nos demais trabalhos, e pelos recursos computacionais utilizados.

A classificação multiclasse apresenta o desafio do desbalanceamento de classes (CHU; KIM; HAN, 2021), onde algumas categorias (como a "pavimentação asfáltica") são significativamente mais frequentes nas imagens de treinamento do que outras (como "motocicleta"). Esse desbalanceamento dificulta a aprendizagem de classes minoritárias, e, de fato, as Tabelas 16 e 17 demonstram a tendência de que o aumento do número de classes diminui a performance do modelo.

Os recursos computacionais desempenharam um papel fundamental na realização deste trabalho. A quantidade de memória RAM disponível foi essencial para carregar os grandes conjuntos de dados utilizados no treinamento do modelo, enquanto a utilização da placa de vídeo (GPU) com memória dedicada acelerou significativamente o processo de treinamento.

5.5 Considerações Parciais

Neste capítulo, foram apresentados os resultados de três ensaios, todos utilizando a mesma arquitetura *U-Net*, mas diferindo no método de treinamento. O primeiro ensaio avalia a invariância à condição de iluminação do mapa de texturas LBP. Foram realizados dois treinamentos (A e B) contendo apenas imagens diurnas. O Treino A é realizado com imagens rasterizadas e Treino B com os mapas de textura LBP. Os resultados demonstraram que a performance do Treino B foi superior à do Treino A quando testado com imagens noturnas.

O segundo ensaio foi realizado com caráter de controle experimental, para validar o treinamento da *U-Net* com imagens diurnas e noturnas. Foram realizados dois treinamentos (C e D) contendo imagens diurnas e noturnas. O Treino C é realizado com imagens rasterizadas e Treino D com os mapas de textura LBP. Os resultados de todos os testes foram muito semelhantes.

O terceiro ensaio adiciona o mapa de texturas LBP em uma quarta camada nas imagens de treino. São realizados também dois tipos de treinamento: Treino E, apenas com imagens diurnas, e Treino F, com imagens diurnas e noturnas. O Treino F apresentou resultados muito próximos aos do Treino C. Apesar disso, é preciso chamar a atenção para o Treino E que mostra índices maiores que os do Treino A no teste com imagens noturnas. Isso evidencia uma capacidade de generalização melhor ao se concatenar o mapa de texturas LBP.

A Tabela 16 faz uma comparação dos resultados obtidos com os publicados por outros trabalhos. É notório que a referência em Yousri, Elattar e Darweesh (2021) apre-

sentou taxas de Acurácia acima de 99% para algumas condições específicas. Esse trabalho emprega uma base de dados diferente (a *nuScenes*), além disso, é descrito um algoritmo próprio baseado em análises gráficas e geométricas para gerar as máscaras de segmentação. Dessa forma, as máscaras de segmentação não levam em conta a presença de objetos (como outros carros, motos, bicicletas e pessoas) na pavimentação asfáltica. Essa simplicidade nas imagens e máscaras utilizadas pode explicar as altas taxas de acurácia publicadas.

Na comparação com o restante dos trabalhos, as taxa de acurácia obtidos por este trabalho são mais altas, na faixa de 97%, nos casos em que o treinamento e os testes são realizados com ambos os tipos de imagens (dia e noite). Certamente que a abordagem de classificação binária, os recursos computacionais disponíves (memória para alocação das imagens e placa GPU para aceleração do treinamento e dos testes), e os ajustes dos parâmetros de treinamento, contribuiram para o bom desempenho da *U-Net* adotada.

O próximo capítulo traz as conclusões finais acerca dos resultados da pesquisa. Os resultados mais expressivos são salientados, mas também são destacadas as vantagens observadas no treinamento da *U-Net* com mapas de textura LBP. Além disso, é realizada uma análise comparativa dos resultados obtidos em trabalhos relacionados, e por fim são listados temas potenciais para investigações futuras.

6 CONCLUSÕES FINAIS

Este trabalho foi estruturado em cinco capítulos, e apresenta um estudo sobre a utilização de mapas de textura LBP para treinamento de *U-Net* a fim de realizar a segmentação semântica de pavimentações asfálticas. O Capítulo 1 introduz o contexto da Inteligência Artificial (IA) e justifica o uso de métodos de IA como solução para o desafio de segmentação semântica proposto. Ressaltando a flexibilidade e adaptabilidade das Redes Neurais Artificiais (RNAs) como técnicas de IA, o trabalho destaca sua capacidade de resolver problemas complexos quando a descrição detalhada em algoritmos tradicionais se torna impraticável devido à quantidade e variedade de informações que influenciam o resultado (RUSSELL; NORVIG, 2010).

Assim, reconhece-se que a segmentação semântica, para aplicações como a direção autônoma, é uma tarefa de nível elevado de dificuldade para os métodos tradicionais, mas que se torna possível graças a IA. No âmbito desta pesquisa, foi destacada a *U-Net* (RONNEBERGER; P.FISCHER; BROX, 2015) como uma ferramenta eficaz para realizar a segmentação semântica e identificação de áreas de pavimentação asfáltica em imagens rasterizadas e em mapas de texturas gerados algoritmo *Local Binary Pattern* (LBP). Os mapas de textura LBP são obtidos atráves do processamento de uma imagem rasterizada pelo descritor de texturas LBP (OJALA; PIETIKäINEN; HARWOOD, 1996), sendo a exploração de seus benefícios um diferencial deste trabalho. Seu uso mostrou uma melhora no desempenho da tarefa em comparação com imagens rasterizadas, no caso em que o modelo é testado com imagens em condições de iluminação diferentes das quais foi treinado.

O Capítulo 2 aprofunda a fundamentação teórica relevante para esta pesquisa. São abordados os conceitos fundamentais da estrutura do neurônio artificial, o processo de treinamento, e os métodos de interconexão de várias unidades de neurônios artificiais em camadas para a formação de RNAs. A combinação de todos esses tópicos posibilita a implementação da arquitetura *U-Net*. Além disso, este capítulo ainda discorre acerca do funcionamento do descritor de texturas LBP, ressaltando suas características e vantagens para este trabalho.

O interesse nos mapas de textura LBP adveio da constatação da dificuldade da segmentação semântica em se obter uma base de dados diversificada para o treinamento da rede. Essa base de dados é crucial para ajustar e afinar os parâmetros do modelo a fim de reconhecer os padrões de estruturas contidos em suas amostras (BISHOP, 2006). O sucesso desse processo permite que o modelo realize previsões coerentes para novas entradas, desde que os padrões de estruturas possam ser extraídos e utilizados para tomar decisões. Porém, se a entrada contiver padrões muito diferentes para os quais não há referência na base de

dados, o resultado pode ser prejudicado.

Ou seja, um modelo treinado exclusivamente com imagens capturadas durante o dia pode não apresentar uma resposta adequada quando confrontado com uma imagem capturada durante a noite. Dessa forma, surge a hipótese de que a propriedade de invariância em relação à iluminação do operador LBP pode ser aproveitada para aprimorar a precisão do modelo.

O Capítulo 3 analisa algumas publicações que também se propuseram a solucionar a questão da segmentação semântica de pavimentações asfálticas empregando métodos diversos de IA. Ainda neste capítulo, são apresentados dois outros artigos que utilizaram os mapas de texturas LBP para treinamento de RNAs em aplicações distintas. Os trabalhos em questão realizam uma comparação do desempenho dos modelos quando treinados com imagens rasterizadas e mapas de textura LBP, reportando um aprimoramento da Acurácia em favor dos mapas de textura LBP.

O Capítulo 4 lista os recursos materiais e métodos utilizados para o desenvolvimento deste trabalho. São descritas todas as bibliotecas de *software* e os equipamentos computacionais empregados. É importante destacar a relevância dos amplos recursos de *hardware* da plataforma *Colab* para a implementação e treinamento da *U-Net*, que, além de necessitar de uma quantidade substancial de memória RAM, também exige uma placa GPU para aceleração do processo de treinamento. Este capítulo também detalha a implementação do código da *U-Net*, duas métricas de avaliação de performance - Acurácia e média *Intersection over Union* (IoU) -, e três ensaios que são realizados para avaliar o desempenho do modelo com mapas de textura LBP e compará-lo com o de imagens rasterizadas.

O Capítulo 5 reporta os resultados dos ensaios propostos no Capítulo 4. Mostra-se que a segmentação semântica é alcançada pelo modelo tanto por imagens rasterizadas quanto pelos mapas de textura LBP. Os resultados da segunda parte do primeiro ensaio da seção 5.1, demonstram que a média IoU do Treino B para o cenário noturno (para o qual não foi treinada), foi de `81` %. Isso é significativamente maior que o apresentado pelo Treino A, realizado apenas com imagens rasterizadas diurnas, que atingiu `61,4`%. Apesar de o resultado não ter sido tão alto quanto o obtido nos testes com imagens diurnas, em que ambos os modelos apresentaram média IoU de aproximadamente `89` %, é possível inferir que o mapa de texturas LBP evidencia uma quantidade maior de informações estruturais na área da pavimentação asfáltica que auxiliam na predição. Ou seja, a capacidade de generalização do modelo é aumentada.

O terceiro ensaio 5.3 também demonstrou resultados muito semelhantes para o Treino F (treinado com imagens rasterizadas diurnas e noturnas e com o mapa de texturas LBP concatenado) quando comparado com o Treino C (treinado apenas com imagens rasterizadas diurnas e noturnas). Não obstante, é preciso chamar a atenção para a perfor-

mance do Treino E e compará-la com a do Treino A. Ambos foram treinados apenas com imagens diurnas, mas o mapa de texturas LBP foi adicionado nas imagens de treinamento do Treino E. Quando ambos foram submetidos ao teste de imagens noturnas, observou-se que o Treino E foi consideravelmente melhor, com 71,9% de média IoU, em relação ao Treino A, que apresentou 61,4 %, reforçando novamente a vantagem concedida pelo mapa de texturas LBP.

De forma geral, as redes treinadas puramente com o mapa de texturas LBP (Treinos B e D) apresentaram métricas menores (porém, muito próximas) em relação aos treinamentos que utilizaram apenas as imagens rasterizadas (Treinos A e C). Isto é, quando os testes consideram apenas imagens semelhantes às quais foram treinadas.

Na comparação dos resultados obtidos com os publicados por outros trabalhos (Tabela 16) verificou-se que os Treinos C e F implementados neste trabalho ficaram acima da maior parte dos outros modelos com Acurácia próxima de 97%. Em grande parte, os recursos de *hardware* utilizados (memória para alocação das imagens e placa GPU para aceleração do treinamento e dos testes) contribuiram para o bom desempenho da *U-Net* adotada. Além disso, deve-se ressaltar o esforço dedicado ao ajuste dos parâmetros do modelo, visando estabilizar a etapa de treinamento.

É necessário observar que a referência em Yousri, Elattar e Darweesh (2021), que apresentou taxas de Acurácia bastante elevadas, ultrapassando 99% em algumas situações. Dentre as razões que a base de treinamento utilizada (a *nuScapes*) em Yousri, Elattar e Darweesh (2021) foi diferente, e as máscaras de segmentação foram geradas de forma automatizada por um algoritmo baseado em análises gráficas e geométricas descrito no próprio trabalho. Dessa forma, as imagens utilizadas não contém a presença de outros objetos (como outros carros, motos, bicicletas e pessoas na) pavimentação asfáltica, simplificando a análise e predição, o pode explicar as altas taxas de acurácia publicadas.

Os resultados obtidos pelos treinamentos feitos com mapas de textura LBP foram muito próximos dos obtidos com os treinamentos com imagens rasterizadas, particularmente nos casos em que as imagens de teste não se distanciam do escopo do treinamento. No entanto, a principal contribuição deste trabalho reside na demonstração de que o treinamento com os mapas de textura LBP podem conferir ao modelo uma capacidade de generalização maior, reafirmando o que é citado em Zhang et al. (2017).

De fato, a Acurácia nos testes em imagens noturnas foi significativamente maior para os casos em que o os mapas de texturas LBP são usados, mas sem que haja imagens noturnas na etapa de treinamento. Essa abordagem pode representar uma redução de custos na montagem de bases de dados, possibilitando a diminuição do tamanho da base sem comprometer sua abrangência. Além disso, pode trazer benefícios em termos de segurança, pois o sistema se torna mais robusto em face de situações não contempladas no treinamento original.

6.1 Trabalhos Futuros

No âmbito da segmentação semântica com redes neurais artificiais, estudos futuros podem explorar o treinamento da rede em outros cenários climáticos desafiadores, como neblina, chuva e neve, contidas na base de dados ACDC. Também é interessante a expansão das bases de dados além do conjunto de classes de objetos para incluir carros, motocicletas, sinais e faixas de trânsito, bicicletas, pessoas, árvores, edificações e outras, que são elementos comuns no cenário de navegação em pavimentações asfálticas. Por fim, sugere-se a experimentação de diferentes arquiteturas neurais ou a implementação de modificações na estrutura da *U-Net*, buscando aprimorar e otimizaro desempenho do modelo.

DISSEMINAÇÃO

Principais trabalhos publicados ou submetidos pelo autor durante o programa.

1. YUI, M. K.; MELO, L. F. D. Leveraging LBP Texture Maps in U-Net for Road Semantic Segmentation. In: *2024 IEEE Transactions on Neural Networks and Learning Systems*. 2024.

2. YUI, M. K.; MELO, L. F. D. Evaluation of LBP Learning for U-Net Based Road Semantic Segmentation. In: *2024 39º Simpósio Sul de Microeletrônica*. 2024.

REFERÊNCIAS

AIZAN, J.; EZIN, E.; MOTAMED, C. *A Face Recognition Approach Based on Nearest Neighbor Interpolation and Local Binary Pattern*. : , 2016. 76-81 p. Disponível em: <https://doi.org/10.1109/SITIS.2016.21>.

AOTO, M.; WADA, Y.; NUMATA, Y. Development of an fpga controlled "mini-car"toward autonomous driving. In: *2018 International Conference on Field-Programmable Technology (FPT)*. : , 2018. p. 400–402.

BADRINARAYANAN, V.; KENDALL, A.; CIPOLLA, R. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. 2016.

BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

BRIDLE, J. S. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In: TOURETZKY, D. S. (Ed.). *Advances in Neural Information Processing Systems*. San Mateo CA: Morgan Kaufmann, California, 1990. v. 2, p. 211–217.

CHU, S.; KIM, D.; HAN, B. *Learning Debiased and Disentangled Representations for Semantic Segmentation*. 2021. Disponível em: <https://arxiv.org/abs/2111.00531>.

CORDTS, M. et al. The cityscapes dataset for semantic urban scene understanding. In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. : , 2016.

DROZDZAL, M. et al. *The Importance of Skip Connections in Biomedical Image Segmentation*. 2016.

DUBEY, S. R.; SINGH, S. K.; CHAUDHURI, B. B. A comprehensive survey and performance analysis of activation functions in deep learning. *CoRR*, abs/2109.14545, 2021. Disponível em: <https://arxiv.org/abs/2109.14545>.

DUMOULIN, V.; VISIN, F. A guide to convolution arithmetic for deep learning. 03 2016.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, , , v. 36, n. 4, p. 193–202, Apr 1980. ISSN 1432-0770. Disponível em: <https://doi.org/10.1007/BF00344251>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. : MIT Press, 2016. <http://www.deeplearningbook.org>.

GPU Platforms. <https://cloud.google.com/compute/docs/gpus>. Accessed: 2024-02-06.

GREESHMA, P. G. Different approaches for semantic segmentation. In: *2020 5th International Conference on Communication and Electronics Systems (ICCES)*. : , 2020. p. 938–943.

HAHNLOSER, R. H. R. et al. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, v. 405, n. 6789, p. 947–951, jun. 2000.

HEBB, D. *The Organization of Behavior: A Neuropsychological Theory*. : Wiley, 1949. (A Wiley book in clinical psychology). ISBN 9780471367277.

JEBAMIKYOUS, H.-H.; KASHEF, R. Deep learning-based semantic segmentation in autonomous driving. In: *2021 IEEE 23rd Int Conf on High Performance Computing Communications; 7th Int Conf on Data Science Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud Big Data Systems Application (HPCC/DSS/SmartCity/DependSys)*. : , 2021. p. 1367–1373.

KINGMA, D. P.; BA, J. *Adam: A Method for Stochastic Optimization*. 2017.

LAZUARDI, R. N. et al. A system of semantic segmentation on an autonomous vehicle. In: *2019 16th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. : , 2019. p. 786–789.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

LI-YONG, M. et al. A lane detection technique based on adaptive threshold segmentation of lane gradient image. In: *2018 4th Annual International Conference on Network and Information Systems for Computers (ICNISC)*. : , 2018. p. 182–186.

LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2015. p. 3431–3440. ISSN 1063-6919. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/CVPR.2015.7298965>.

LU, L. Dying relu and initialization: Theory and numerical examples. *Communications in Computational Physics*, Global Science Press, v. 28, n. 5, p. 1671–1706, jun. 2020. ISSN 1991-7120. Disponível em: <http://dx.doi.org/10.4208/cicp.OA-2020-0165>.

MALLIKARJUNA, P. B. et al. The kth-tips 2 database. In: . : , 2006. Disponível em: <https://api.semanticscholar.org/CorpusID:15987374>.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115–133, 1943.

MINSKY, M.; PAPERT, S. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

MüLLER, D.; SOTO-REY, I.; KRAMER, F. *Towards a Guideline for Evaluation Metrics in Medical Image Segmentation*. 2022.

NIE, X. et al. A novel vision based road detection algorithm for intelligent vehicle. In: *2019 IEEE International Conference on Power, Intelligent Computing and Systems (ICPICS)*. : , 2019. p. 504–507.

NOH, H.; HONG, S.; HAN, B. *Learning Deconvolution Network for Semantic Segmentation*. 2015.

OJALA, T.; PIETIKäINEN, M.; HARWOOD, D. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, v. 29, n. 1, p. 51–59, 1996. ISSN 0031-3203. Disponível em: <https://www.sciencedirect.com/science/article/pii/0031320395000674>.

O'SHEA, K.; NASH, R. *An Introduction to Convolutional Neural Networks*. 2015.

PADILLA, R.; NETTO, S. L.; SILVA, E. A. B. da. A survey on performance metrics for object-detection algorithms. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. : , 2020. p. 237–242.

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

PHAM, T. Semantic road segmentation using deep learning. In: *2020 Applying New Technology in Green Buildings (ATiGB)*. : , 2021. p. 45–48.

PHUEAKJEEN, W. et al. A study of the edge detection for road lane. In: *The 8th Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI) Association of Thailand - Conference 2011*. : , 2011. p. 995–998.

PURWONO, P. et al. Understanding of convolutional neural network (cnn): A review. v. 2, p. 739–748, 01 2023.

RAINIO, O.; TEUHO, J.; KLéN, R. Evaluation metrics and statistical tests for machine learning. *Scientific Reports*, v. 14, n. 1, p. 6086, Mar 2024. ISSN 2045-2322. Disponível em: <https://doi.org/10.1038/s41598-024-56706-x>.

RONNEBERGER, O.; P.FISCHER; BROX, T. U-net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2015. (LNCS, v. 9351), p. 234–241. (available on arXiv:1505.04597 [cs.CV]). Disponível em: <http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a>.

ROSENBLATT, F. *The perceptron - A perceiving and recognizing automaton*. Ithaca, New York, 1957.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, v. 323, n. 6088, p. 533–536, Oct 1986. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/323533a0>.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. [S.l.]: Prentice Hall, 2010.

SAKARIDIS, C.; DAI, D.; GOOL, L. V. ACDC: The adverse conditions dataset with correspondences for semantic driving scene understanding. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. : , 2021.

SARKER, I. H. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, v. 2, n. 6, p. 420, Aug 2021. ISSN 2661-8907. Disponível em: <https://doi.org/10.1007/s42979-021-00815-1>.

SARKER, I. H. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, v. 2, n. 3, p. 160, Mar 2021. ISSN 2661-8907. Disponível em: <https://doi.org/10.1007/s42979-021-00592-x>.

SARMAH, P.; GOGOI, A.; KALITA, S. Encoder decoder based deep learning architecture for video scene parsing. In: *2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA)*. : , 2022. p. 1–6.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding Machine Learning - From Theory to Algorithms.* : Cambridge University Press, 2014. I-XVI, 1-397 p. ISBN 978-1-10-705713-5.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, , , 09 2014.

TOOSI, A. et al. A brief history of ai: How to prevent another winter (a critical review). *PET Clinics*, Elsevier BV, , v. 16, n. 4, p. 449–469, out. 2021. ISSN 1556-8598. Disponível em: <http://dx.doi.org/10.1016/j.cpet.2021.07.001>.

WALI, R. *Xtreme Margin: A Tunable Loss Function for Binary Classification Problems*. 2022.

WERBOS, P.; JOHN, P. Beyond regression : new tools for prediction and analysis in the behavioral sciences. , , 01 1974.

WU, H.; ZHOU, J.; LI, Y. Deep generative model for image inpainting with local binary pattern learning and spatial attention. *IEEE Transactions on Multimedia*, v. 24, p. 4016–4027, 2022.

YAMASHITA, R. et al. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, v. 9, n. 4, p. 611–629, Aug 2018. ISSN 1869-4101. Disponível em: <https://doi.org/10.1007/s13244-018-0639-9>.

YOUSRI, R.; ELATTAR, M. A.; DARWEESH, M. S. A deep learning-based benchmarking framework for lane segmentation in the complex and dynamic road scenes. *IEEE Access*, v. 9, p. 117565–117580, 2021.

ZHANG, H. et al. A face recognition method based on lbp feature for cnn. In: *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*. [S.l.: s.n.], 2017. p. 544–547.

ZHANG, J.; NAGEL, H.-H. Texture-based segmentation of road images. In: *Proceedings of the Intelligent Vehicles '94 Symposium*. : , 1994. p. 260–265.

ZHAO, H. et al. *Pyramid Scene Parsing Network*. 2017.

Apêndices

APÊNDICE A – CÓDIGO DO PROJETO - IMPLEMENTAÇÃO EM *PYTHON*

```
import os
import glob
import zipfile
import numpy as np
import random

import cv2
from PIL import Image

from skimage.feature import local_binary_pattern
from matplotlib import pyplot as plt

from tensorflow.keras.metrics import MeanIoU
from tensorflow.keras.metrics import Accuracy
from tensorflow.keras.utils import normalize

from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D, concatenate, Conv2DTranspose, BatchNormalization, Dro
from keras.optimizers import Adam
from keras.layers import Activation, MaxPool2D, Concatenate
from keras.utils import plot_model

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler


W_SIZE = 512
H_SIZE = 256

dim = (W_SIZE, H_SIZE)

zip_ref = zipfile.ZipFile('/content/drive/MyDrive/Colab Notebooks/Datasets/Cityscapes/train/raw.zip', 'r') #Opens the zip
zip_ref.extractall('/tmp/cityscapes/train/') #Extracts the files into the /tmp folder
zip_ref.close()

image_names = glob.glob('/tmp/cityscapes/train/raw/*/*.png')
image_names.sort()

num_images = len(image_names)

image_names_subset = image_names[0 : num_images]

images = []
for img in image_names_subset:
  image_gray_ = cv2.imread(img,0)
  lbp_ = local_binary_pattern(image_gray_, P=8, R=1)
  images.append(lbp_)

image_dataset = np.array(images)
image_dataset = np.expand_dims(image_dataset, axis = 3)
image_dataset = image_dataset /255

zip_ref = zipfile.ZipFile('/content/drive/MyDrive/Colab Notebooks/Datasets/Cityscapes/train/mask.zip', 'r') #Opens the zi
```

```
zip_ref.extractall('/tmp/cityscapes/train/') #Extracts the files into the /tmp folder
zip_ref.close()

mask_names = glob.glob('/tmp/cityscapes/train/mask/*/*.png')
mask_names.sort()

num_images = len(mask_names)

mask_names_subset = mask_names[0:num_images]

masks = []
for img in mask_names_subset:
  mask_ = cv2.imread(img, 0)
  binary_mask_ = np.where(mask_ == 90, 1, 0) # Replace any value with 1
  masks.append(binary_mask_)

mask_dataset = np.array(masks)
mask_dataset = np.expand_dims(mask_dataset, axis = 3)

X_train, X_test, y_train, y_test = train_test_split(image_dataset, mask_dataset, test_size = 0.20)

IMG_HEIGHT = image_dataset.shape[1]
IMG_WIDTH  = image_dataset.shape[2]
IMG_CHANNELS = image_dataset.shape[3]

input_shape = (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)

# Inicio da implementação da arquitetura U-Net
inputs = Input(input_shape)

# Codificador Bloco 1
c1 = Conv2D(64, 3, padding="same")(inputs)
c1 = BatchNormalization()(c1)
c1 = Activation("relu")(c1)

c1 = Conv2D(64, 3, padding="same")(c1)
c1 = BatchNormalization()(c1)
c1 = Activation("relu")(c1)

p1 = MaxPool2D((2, 2))(c1)

# Codificador Bloco 2
c2 = Conv2D(128, 3, padding="same")(p1)
c2 = BatchNormalization()(c2)
c2 = Activation("relu")(c2)

c2 = Conv2D(128, 3, padding="same")(c2)
c2 = BatchNormalization()(c2)
c2 = Activation("relu")(c2)

p2 = MaxPool2D((2, 2))(c2)

# Codificador Bloco 3
c3 = Conv2D(256, 3, padding="same")(p2)
c3 = BatchNormalization()(c3)
c3 = Activation("relu")(c3)

c3 = Conv2D(256, 3, padding="same")(c3)
c3 = BatchNormalization()(c3)
c3 = Activation("relu")(c3)
```

```
p3 = MaxPool2D((2, 2))(c3)

# Codificador Bloco 4
c4 = Conv2D(512, 3, padding="same")(p3)
c4 = BatchNormalization()(c4)
c4 = Activation("relu")(c4)

c4 = Conv2D(512, 3, padding="same")(c4)
c4 = BatchNormalization()(c4)
c4 = Activation("relu")(c4)

p4 = MaxPool2D((2, 2))(c4)

# Bridge
b1 = Conv2D(1024, 3, padding="same")(p4)
b1 = BatchNormalization()(b1)
b1 = Activation("relu")(b1)

b1 = Conv2D(1024, 3, padding="same")(b1)
b1 = BatchNormalization()(b1)
b1 = Activation("relu")(b1)

# Decodificador Bloco 1
d1 = Conv2DTranspose(512, (2, 2), strides=2, padding="same")(b1)
d1 = Concatenate()([d1, c4])

d1 = Conv2D(512, 3, padding="same")(d1)
d1 = BatchNormalization()(d1)
d1 = Activation("relu")(d1)

d1 = Conv2D(512, 3, padding="same")(d1)
d1 = BatchNormalization()(d1)
d1 = Activation("relu")(d1)

# Decodificador Bloco 2
d2 = Conv2DTranspose(256, (2, 2), strides=2, padding="same")(d1)
d2 = Concatenate()([d2, c3])

d2 = Conv2D(256, 3, padding="same")(d2)
d2 = BatchNormalization()(d2)
d2 = Activation("relu")(d2)

d2 = Conv2D(256, 3, padding="same")(d2)
d2 = BatchNormalization()(d2)
d2 = Activation("relu")(d2)

# Decodificador Bloco 3
d3 = Conv2DTranspose(128, (2, 2), strides=2, padding="same")(d2)
d3 = Concatenate()([d3, c2])

d3 = Conv2D(128, 3, padding="same")(d3)
d3 = BatchNormalization()(d3)
d3 = Activation("relu")(d3)

d3 = Conv2D(128, 3, padding="same")(d3)
d3 = BatchNormalization()(d3)
d3 = Activation("relu")(d3)

# Decodificador Bloco 4
```

```
d4 = Conv2DTranspose(64, (2, 2), strides=2, padding="same")(d3)
d4 = Concatenate()([d4, c1])

d4 = Conv2D(64, 3, padding="same")(d4)
d4 = BatchNormalization()(d4)
d4 = Activation("relu")(d4)

d4 = Conv2D(64, 3, padding="same")(d4)
d4 = BatchNormalization()(d4)
d4 = Activation("relu")(d4)

# Se o número de classes for 1, use a função de ativação sigmoide
activation = 'sigmoid'

# Se o número de classes for maior que 1, use a função de ativação softmax
# activation = 'softmax'

outputs = Conv2D(1, 1, padding="same", activation=activation)(d4)
# Fim da implementação da arquitetura U-Net

# Instanciação do modelo
model = Model(inputs, outputs, name="U-Net")

# Inicialização dos parâmetros
model.compile(optimizer=Adam(learning_rate = 0.2e-3), loss='binary_crossentropy', metrics=['accuracy'])

# Resumo dos parâmetros
model.summary()

# Diagrama da estrutura
plot_model(model, show_shapes=True, show_layer_names=False)

# Treinamento do modelo
history = model.fit(X_train, y_train, batch_size = 16,
  verbose=1, epochs=10, validation_data=(X_test, y_test),
  shuffle=False)

# Função de custo do treinamento
loss = history.history['loss']

# Função de custo da validação
val_loss = history.history['val_loss']

epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Precisão de treinamento
acc = history.history['accuracy']

# Precisão de validação
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
```

```
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Teste da U-Net
y_pred=model.predict(X_test)
y_pred_thresholded = y_pred > 0.5

# Métrica de Intersecção Sobre a Área Média
n_classes = 2
IOU_keras = MeanIoU(num_classes=n_classes)
IOU_keras.update_state(y_pred_thresholded, y_test)
print(IOU_keras.result().numpy())

# Métrica de Precisão
accuracy_keras = Accuracy()
accuracy_keras.update_state(y_pred_thresholded, y_test)
print(accuracy_keras.result().numpy())

# Métrica de Precisão
test_img_number = random.randint(0, len(X_test)-1)
test_img = X_test[test_img_number]
ground_truth=y_test[test_img_number]
test_img_input=np.expand_dims(test_img, 0)
prediction = (model.predict(test_img_input)[0,:,:,0] > 0.5).astype(np.uint8)

# Exemplo de exibição de imagem utilizada
plt.figure(figsize=(16, 8))
plt.subplot(231)
plt.title('Testing Image')
plt.imshow(test_img[:,:,0], cmap='gray')
plt.subplot(232)
plt.title('Testing Label')
plt.imshow(ground_truth[:,:,0], cmap='gray')
plt.subplot(233)
plt.title('Prediction on test image')
plt.imshow(prediction, cmap='gray')

plt.show()
```

APÊNDICE B – SUMÁRIO DO MODELO U-NET

```
Model: "U-Net"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 256, 512, 3)]     0         []

 conv2d (Conv2D)             (None, 256, 512, 64)      1792      ['input_1[0][0]']

 batch_normalization (Batch  (None, 256, 512, 64)      256       ['conv2d[0][0]']
 Normalization)

 activation (Activation)     (None, 256, 512, 64)      0         ['batch_normalization[0][0]']

 conv2d_1 (Conv2D)           (None, 256, 512, 64)      36928     ['activation[0][0]']

 batch_normalization_1 (Bat  (None, 256, 512, 64)      256       ['conv2d_1[0][0]']
 chNormalization)

 activation_1 (Activation)   (None, 256, 512, 64)      0         ['batch_normalization_1[0][0]'
                                                                 ]

 max_pooling2d (MaxPooling2  (None, 128, 256, 64)      0         ['activation_1[0][0]']
 D)

 conv2d_2 (Conv2D)           (None, 128, 256, 128)     73856     ['max_pooling2d[0][0]']

 batch_normalization_2 (Bat  (None, 128, 256, 128)     512       ['conv2d_2[0][0]']
 chNormalization)

 activation_2 (Activation)   (None, 128, 256, 128)     0         ['batch_normalization_2[0][0]'
                                                                 ]

 conv2d_3 (Conv2D)           (None, 128, 256, 128)     147584    ['activation_2[0][0]']

 batch_normalization_3 (Bat  (None, 128, 256, 128)     512       ['conv2d_3[0][0]']
 chNormalization)

 activation_3 (Activation)   (None, 128, 256, 128)     0         ['batch_normalization_3[0][0]'
                                                                 ]

 max_pooling2d_1 (MaxPoolin  (None, 64, 128, 128)      0         ['activation_3[0][0]']
 g2D)

 conv2d_4 (Conv2D)           (None, 64, 128, 256)      295168    ['max_pooling2d_1[0][0]']

 batch_normalization_4 (Bat  (None, 64, 128, 256)      1024      ['conv2d_4[0][0]']
 chNormalization)

 activation_4 (Activation)   (None, 64, 128, 256)      0         ['batch_normalization_4[0][0]'
                                                                 ]

 conv2d_5 (Conv2D)           (None, 64, 128, 256)      590080    ['activation_4[0][0]']

 batch_normalization_5 (Bat  (None, 64, 128, 256)      1024      ['conv2d_5[0][0]']
 chNormalization)
```

```
activation_5 (Activation)   (None, 64, 128, 256)   0         ['batch_normalization_5[0][0]']

max_pooling2d_2 (MaxPooling2D)  (None, 32, 64, 256)   0         ['activation_5[0][0]']

conv2d_6 (Conv2D)           (None, 32, 64, 512)    1180160   ['max_pooling2d_2[0][0]']

batch_normalization_6 (BatchNormalization)  (None, 32, 64, 512)   2048      ['conv2d_6[0][0]']

activation_6 (Activation)   (None, 32, 64, 512)    0         ['batch_normalization_6[0][0]']

conv2d_7 (Conv2D)           (None, 32, 64, 512)    2359808   ['activation_6[0][0]']

batch_normalization_7 (BatchNormalization)  (None, 32, 64, 512)   2048      ['conv2d_7[0][0]']

activation_7 (Activation)   (None, 32, 64, 512)    0         ['batch_normalization_7[0][0]']

max_pooling2d_3 (MaxPooling2D)  (None, 16, 32, 512)   0         ['activation_7[0][0]']

conv2d_8 (Conv2D)           (None, 16, 32, 1024)   4719616   ['max_pooling2d_3[0][0]']

batch_normalization_8 (BatchNormalization)  (None, 16, 32, 1024)   4096      ['conv2d_8[0][0]']

activation_8 (Activation)   (None, 16, 32, 1024)   0         ['batch_normalization_8[0][0]']

conv2d_9 (Conv2D)           (None, 16, 32, 1024)   9438208   ['activation_8[0][0]']

batch_normalization_9 (BatchNormalization)  (None, 16, 32, 1024)   4096      ['conv2d_9[0][0]']

activation_9 (Activation)   (None, 16, 32, 1024)   0         ['batch_normalization_9[0][0]']

conv2d_transpose (Conv2DTranspose)  (None, 32, 64, 512)    2097664   ['activation_9[0][0]']

concatenate (Concatenate)   (None, 32, 64, 1024)   0         ['conv2d_transpose[0][0]',
                                                               'activation_7[0][0]']

conv2d_10 (Conv2D)          (None, 32, 64, 512)    4719104   ['concatenate[0][0]']

batch_normalization_10 (BatchNormalization)  (None, 32, 64, 512)   2048      ['conv2d_10[0][0]']

activation_10 (Activation)  (None, 32, 64, 512)    0         ['batch_normalization_10[0][0]']

conv2d_11 (Conv2D)          (None, 32, 64, 512)    2359808   ['activation_10[0][0]']

batch_normalization_11 (BatchNormalization)  (None, 32, 64, 512)   2048      ['conv2d_11[0][0]']
```

```
activation_11 (Activation)  (None, 32, 64, 512)    0         ['batch_normalization_11[0][0]
                                                             ']

conv2d_transpose_1 (Conv2D  (None, 64, 128, 256)   524544    ['activation_11[0][0]']
Transpose)

concatenate_1 (Concatenate  (None, 64, 128, 512)   0         ['conv2d_transpose_1[0][0]',
)                                                            'activation_5[0][0]']

conv2d_12 (Conv2D)          (None, 64, 128, 256)   1179904   ['concatenate_1[0][0]']

batch_normalization_12 (Ba  (None, 64, 128, 256)   1024      ['conv2d_12[0][0]']
tchNormalization)

activation_12 (Activation)  (None, 64, 128, 256)   0         ['batch_normalization_12[0][0]
                                                             ']

conv2d_13 (Conv2D)          (None, 64, 128, 256)   590080    ['activation_12[0][0]']

batch_normalization_13 (Ba  (None, 64, 128, 256)   1024      ['conv2d_13[0][0]']
tchNormalization)

activation_13 (Activation)  (None, 64, 128, 256)   0         ['batch_normalization_13[0][0]
                                                             ']

conv2d_transpose_2 (Conv2D  (None, 128, 256, 128)  131200    ['activation_13[0][0]']
Transpose)

concatenate_2 (Concatenate  (None, 128, 256, 256)  0         ['conv2d_transpose_2[0][0]',
)                                                            'activation_3[0][0]']

conv2d_14 (Conv2D)          (None, 128, 256, 128)  295040    ['concatenate_2[0][0]']

batch_normalization_14 (Ba  (None, 128, 256, 128)  512       ['conv2d_14[0][0]']
tchNormalization)

activation_14 (Activation)  (None, 128, 256, 128)  0         ['batch_normalization_14[0][0]
                                                             ']

conv2d_15 (Conv2D)          (None, 128, 256, 128)  147584    ['activation_14[0][0]']

batch_normalization_15 (Ba  (None, 128, 256, 128)  512       ['conv2d_15[0][0]']
tchNormalization)

activation_15 (Activation)  (None, 128, 256, 128)  0         ['batch_normalization_15[0][0]
                                                             ']

conv2d_transpose_3 (Conv2D  (None, 256, 512, 64)   32832     ['activation_15[0][0]']
Transpose)

concatenate_3 (Concatenate  (None, 256, 512, 128)  0         ['conv2d_transpose_3[0][0]',
)                                                            'activation_1[0][0]']

conv2d_16 (Conv2D)          (None, 256, 512, 64)   73792     ['concatenate_3[0][0]']

batch_normalization_16 (Ba  (None, 256, 512, 64)   256       ['conv2d_16[0][0]']
tchNormalization)

activation_16 (Activation)  (None, 256, 512, 64)   0         ['batch_normalization_16[0][0]
```

```
                                                                    ']

 conv2d_17 (Conv2D)          (None, 256, 512, 64)   36928     ['activation_16[0][0]']

 batch_normalization_17 (Ba  (None, 256, 512, 64)   256       ['conv2d_17[0][0]']
 tchNormalization)

 activation_17 (Activation)  (None, 256, 512, 64)   0         ['batch_normalization_17[0][0]
                                                               ']

 conv2d_18 (Conv2D)          (None, 256, 512, 1)    65        ['activation_17[0][0]']

==================================================================================================
Total params: 31055297 (118.47 MB)
Trainable params: 31043521 (118.42 MB)
Non-trainable params: 11776 (46.00 KB)
```

Leveraging LBP Texture Maps in U-Net for Road Semantic Segmentation

Mauricio Kendi Yui, *Department of Electrical Engineering, State University of Londrina*
Leonimer Flávio de Melo, *Department of Electrical Engineering, State University of Londrina*

***Abstract*—This paper investigates the advantages of using Local Binary Pattern (LBP) texture maps to train a U-Net model for road pavement semantic segmentation. LBP is a texture descriptor that captures local pixel variations in a raster image. By processing an image with LBP algorithm, texture maps are produced. These texture maps emphasize key features like edges, corners, and surface irregularities, while are also less susceptible to illumination intensity contained in the original image.**

The incorporation of LBP texture maps into the training process directly exposes the structural information within the images to the U-Net. This enables the model to achieve higher segmentation accuracy on unseen data with varying lighting conditions. For instance, a model trained on daytime images using LBP maps might outperform a model trained directly on daytime raster images when presented with nighttime images. This suggests the effectiveness of LBP texture maps in enhancing the generalizability of the model.

***Index Terms*—Road semantic segmentation, Local Binary Pattern (LBP), U-Net, deep learning, computer vision.**

I. Introduction

SEMANTIC segmentation aims to assign a specific class label (or semantic label) to each pixel in an image, effectively partitioning the image into meaningful regions [1]. It is a suitable solution for applications that demand sophisticated and contextualized understanding of visual content.

Street and road segmentation is a recurrent challenge, with renewed interest fueled by advancements in Artificial Intelligence (AI) and the development of autonomous vehicles. Recognizing the versatility of Deep Learning within AI, this work utilizes a U-Net architecture [2] for the semantic segmentation of road pavements. However, a major concern lies in assembling a comprehensive training dataset encompassing diverse environmental conditions.

To address this issue, the Local Binary Pattern (LBP) texture descriptor [3] is leveraged to extract LBP texture maps that captures important textural features, such as edges, corners, nodes, and surface smoothness and irregularities, from the dataset, while mitigating lighting intensity. By training the U-Net model on LBP texture maps pattern identification is simplified and generalizability and accuracy of predictions are enhanced. This approach has the potential to improve performance on unseen data, such as nighttime road scenes, even if the training dataset primarily consists of daytime images.

Fig. 1. Sample image from Cityscapes dataset (left) and its corresponding LBP texture map (right).

II. Related Work

A. Local Binary Pattern

The Local Binary Pattern (LBP) texture descriptor was introduced by T. Ojala et al. (1996) [3]. It was designed to extract local texture features from images for classification purposes. Its operator is computationally efficient, analyzing local variations in pixel intensity, generating a texture map highly sensitive to image details. The LBP value is calculated for each pixel in an image by equation (1).

$$\mathrm{LBP}_{(x_c,y_c)} = \sum_{n=0}^{N-1} s_{(i_n,i_c)} \cdot 2^n, \tag{1}$$

where (x_c, y_c) is the center pixel position with intensity i_c, N denotes the total number of neighboring pixels, i_n is the intensity of each neighboring pixel, and s is the sign function defined in equation (2).

$$s_{(i_n,i_c)} = \begin{cases} 1, \text{ if } i_n \geq i_c \\ 0, \text{ otherwise.} \end{cases} \tag{2}$$

The LBP texture map describes how the intensity of the center pixel (i_c) compares to its neighbors (i_n). Contrast changes relative to the center pixel is used to detect edges, corners, smooth or irregular surfaces at that point. Figure 1 displays an example of an image obtained from the Cityscapes dataset [4] and its corresponding LBP texture map.

Training neural networks with LBP texture maps has been shown to improve performance in some cases. As demonstrated by H. Zhang et al. (2017) [5], a Convolutional Neural Network (CNN) trained for facial recognition achieved a higher accuracy rate when utilizing the LBP texture map as input, compared to training with raster images.

Another work by H. Wu et al. (2022) [6] proposes a generative model with two distinct U-Nets concatenated for reconstructing images with missing pixels. The first U-Net is designed to predict structural information in the missing

regions, leveraging the LBP texture map of the input image. The resulting information guides the second U-Net, which utilizes both the original image and the reconstructed LBP map to fill in the missing pixels with appropriate colors.

B. Segmentation of Road Pavement

Road segmentation in raster images is a longstanding challenge, with existing literature offering solutions based on methods predating AI. Notably, graphical and geometric approaches, such as those outlined by W. Phueakjeen et al. (2011) [7], remain prevalent. These methods typically involve a combination of edge filtering and subsequent application of Hough Transform to identify traffic lane boundaries.

In Deep Learning context, Yousri et al. (2021) [8] developed a benchmarking framework for lane detection in various adverse wheater conditions. Five artificial neural networks were tested: SegNet, Modified SegNet, U-Net, ResUNet and ResUNet++. Their work also presented their own training and testing segmentation masks. The ResUNet++ model achieved the highest accuracy over all related work analyzed in this section, exceeding 99% in good weather conditions.

Authors H. Jebamikyous and R. Kashef (2021) [9] explored U-Nets and other architectures like Fully Convolutional Network (FCN) and SegNet for road segmentation. The most promising outcomes were achieved by a U-Net variant with a deeper structure (more convolutional and pooling layers) and dropout regularization. This variant has a larger number of parameters compared to the other models presented, this likely allowed it to capture more complex patterns and information in the road images.

However, it must be considered that a larger number of parameters, despite achieving better performance, probably comes at the cost of increased processing time due. This trade-off is highlighted by Pham (2021) [10] where a complex model, Pyramid Scene Parsing Network (PSPNet), achieved superior evaluation metrics but also had a slower runtime compared to simpler models like FCN and SegNet. Hence, it is essential to consider the balance between model complexity and resource constraints, especially when processing time and speed are critical.

In Sarmah et al. (2022) [11] the transfer learning technique for U-Nets is explored. Five models are evaluated: a standard U-Net, two variants where the encoder is replaced with the pre-trained convolutional structure of VGG-16 and VGG-19 networks, and versions of these models without pre-trained weights. It is reported that the U-Net with a pre-trained VGG-16 encoder achieved the best accuracy. This suggests that transfer learning can enhance performance by leveraging pre-trained optimized weights, potentially accelerating the training process and achieve better results compared to training a model from scratch.

III. Methodology

The LBP operator applied to a raster image generates an LBP texture map. Traditionally, texture classification tasks utilize histograms derived from this map, discarding the valuable spatial information of each pixel's position. Instead of using the histogram, this work proposes the direct use of the LBP texture map for training a U-Net for road semantic segmentation task. The LBP map emphasizes structural information within the image, as described in H. Zhang et al. (2017) [5], this potentially improves and facilitates the learning process. Additionally, lighting variations among a set of images are mitigated. This can lead to improved performance on tasks where illumination intensity is a concern.

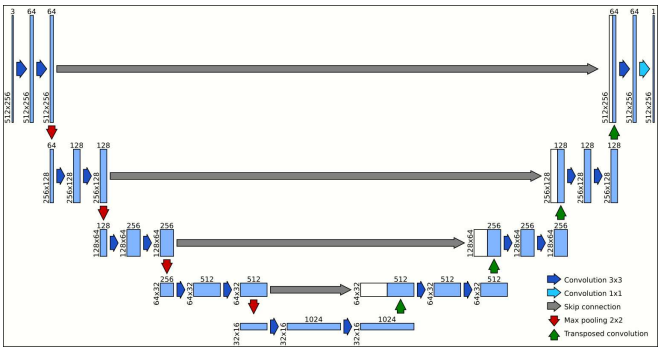


Fig. 2. U-Net architecture.

To validate the effectiveness of using LBP texture map for U-Net training, three experiments are conducted. All experiments utilize the same U-Net architecture, but differ in the training datasets. Details of each experiment are provided in Section IV. These experiments aim to evaluate results and demonstrate the benefits of incorporating LBP texture map.

A. U-Net Architecture

The U-Net model was published by Ronneberger et al. [2]. Originally designed for biomedical purposes, it was soon realized that it is powerful for general object segmentation tasks. The U-Net implemented in this work has a structure similar to that of the original work, and its architecture is depicted in Figure 2.

The U-Net consists of two main parts: an encoder (contracting path) on the left side and a decoder (expanding path) on the right side, connected by a bridge at the bottom. Images are fed into the encoder, which is responsible for learning abstract representations of the input. The encoder has four layers, each containing two convolutional filterings (blue arrows). Each convolutional filtering is followed by a Batch Normalization layer, for training stabilization, and a Rectified Linear Unit (ReLU) activation function. These filters extract features from the image at different scales. Following each convolutional layer, a max pooling operation (red arrow) reduces the spatial resolution of the feature maps while preserving important features. This downsampling process allows the network to capture increasingly complex features but can also lead to spatial information loss.

The decoder aims to translate the abstract representation into a segmentation mask. The structure has four layers mirroring the encoder structure. Each decoder layer upsamples feature maps via transposed convolutions (green arrows) to regain spatial resolution, followed by two convolutional filterings. Each convolutional filtering is also followed by a Batch

Normalization layer and a ReLU activation function. U-Net addresses the spatial information loss by incorporating skip connections (grey arrows). These connections directly copy and concatenate filtered feature maps from the corresponding encoder layers. This reintroduces precise location information from earlier stages, allowing the decoder to recover spatial details and generate more accurate segmentation masks. The last layer in the decoder performs a convolution with a sigmoid activation function (cyan arrow) to generate the output segmentation mask, representing a pixel-wise classification prediction.

B. Training

The code was developed using Python 3.1 on Google Research's Colaboratory platform. This platform provides access to hardware resources including an Intel(R) Xeon(R) CPU @ 2.20 GHz processor, 83.5 GB of RAM, and an A100 GPU with 40 GB of RAM. TensorFlow is an open-source library, and provides functions to implement deep neural networks. Keras is another open-source library that makes a Python interface available for accessing TensorFlow functionalities.

The U-Net parameters were configured with "Adam" as the optimizer, "Binary Cross Entropy" as the loss function, and a "batch size" of 16. The number of training "epochs" was determined through experimentation based on the model's behavior, ranging from 22 to 40 epochs, chosen to minimize loss and prevent accuracy degradation.

Two datasets were employed for training and testing: Cityscapes [4] and Adverse Conditions Dataset with Correspondences (ACDC) [12]. Cityscapes provides 2,974 daytime images captured in European cities for training, along with 500 images for testing. The ACDC dataset offers a significant amount of images in various adverse weather conditions, but for this work, only 400 nighttime images from various European cities were used for training and 106 for testing.

C. Evaluation Metrics

This work employs two common metrics for evaluating semantic segmentation performance, as described in Greeshma et al. (2020) [1]: Intersection over Union (IoU) and Accuracy. IoU measures the overlap between the predicted segmentation mask (or predicted area) and the ground truth mask (or target area), divided by the total number of pixels in both masks. It essentially measures the percentage of pixels correctly classified, and is defined in equation 3.

$$\text{IoU} = \frac{\text{Target Area} \bigcap \text{Predicted Area}}{\text{Target Area} \bigcup \text{Predicted Area}} \tag{3}$$

Evaluating the performance of semantic segmentation models on multiple images (N) often relies on mean Intersection over Union (mIoU), as defined in equation (4). This metric calculates the average IoU score across all N predictions, providing a more comprehensive assessment.

$$mIoU = \frac{1}{N} \cdot \sum_{1}^{N} IoU \tag{4}$$

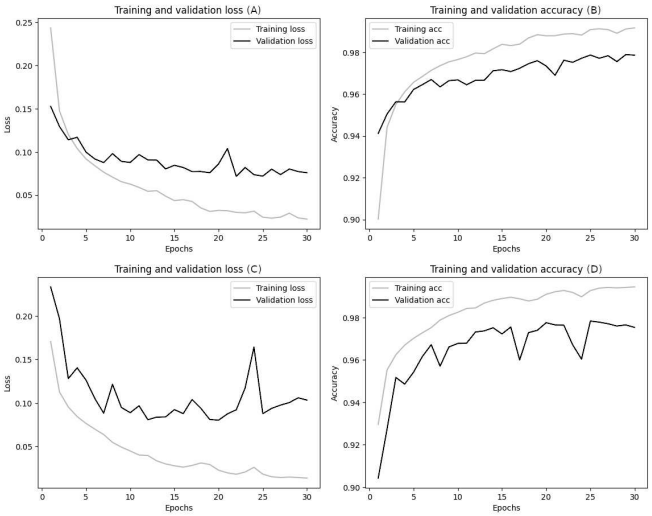


Fig. 3. (A) Training A loss. (B) Training A accuracy. (C) Training B loss. (D) Training B accuracy.

Accuracy is defined in equation 5, and it considers all correctly classified pixels across all classes. It is particularly useful when all classes have equal importance.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}, \tag{5}$$

where TP are true positive, TN are true negative, FP are false positive and FN are false negative predictions.

IV. Experiments and Results

This work investigates the impact of training a U-Net model with LBP texture maps compared to traditional raster images through three experiments.

A. Experiment 1

The first experiment aims to assess the robustness of the LBP texture map under variations in lighting conditions. Two different training configurations are employed: Trainings A and B. Training A used only daytime raster images from the Cityscapes dataset, and Training B used the corresponding LBP texture maps from the same dataset.

Training "epochs" was set to 30 epochs, and to mitigate instabilities during training, the Batch Normalization layer underwent hyperparameter tuning. In both Trainings A and B, the "epsilon" value was set to 0.02 and the "momentum" value to 0.90. Figure 3 displays accuracy and loss curves for Trainings A and B.

Figure 4 presents some of Trainings A and B predictions alongside the original image and the ground truth expected results for comparison.

The nighttime test set from the ACDC dataset was subjected to Trainings A and B prediction. Figure 5 displays some of the U-Net predictions to visually observe the degradation in performance.

Both test image sets from Cityscapes and ACDC datasets were combined and subjected to the U-Net model prediction. All results are presented in Table I.

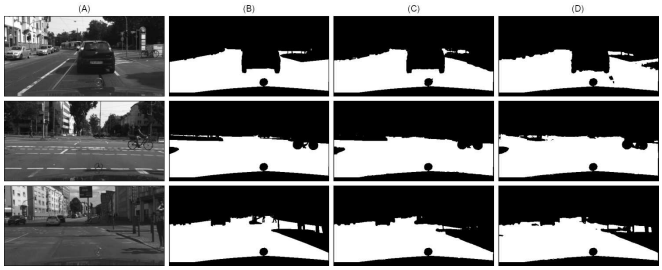


Fig. 4. (A) Original image. (B) Ground truth. (C) Training A prediction on daytime images. (D) Training B prediction on daytime images.

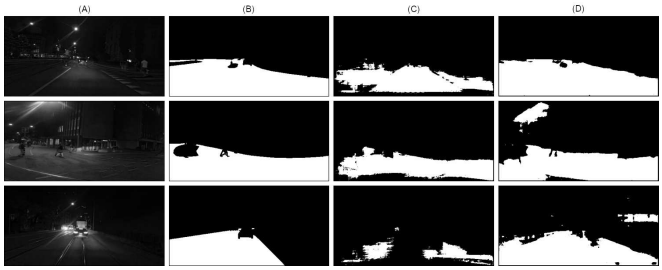


Fig. 5. (A) Original image. (B) Ground truth. (C) Training A prediction on nighttime images. (D) Training B prediction on nighttime images.

TABLE I
TRAININGS A AND B MEAN IOU AND ACCURACY

Training	Test	mIoU (%)	Accuracy (%)
A	daytime	89.5919	97.0260
	nighttime	61.4172	83.2596
	day and nighttime	79.6938	94.6180
B	daytime	89.3334	96.9433
	nighttime	81.0013	91.6363
	day and nighttime	84.1281	96.0150

TABLE II
TRAININGS C AND D MEAN IOU AND ACCURACY

Training	Test	mIoU (%)	Accuracy (%)
C	daytime	89.8027	97.0930
	nighttime	93.5216	97.5151
	day and nighttime	88.1733	97.1669
D	daytime	87.7668	97.0817
	nighttime	88.3482	95.3071
	day and nighttime	86.7408	96.7713

B. Experiment 2

The second experiment is done in order to validate proper U-Net functionality with nighttime images. Two different training configurations are employed: Trainings C and D. Training C used both daytime and nighttime raster images from Cityscapes and ACDC dataset, and Training D used the corresponding LBP texture maps from the same datasets. The same number of epochs and hyperparameters for the Batch Normalization layer described in Experiment 1 were used. Figure 6 displays accuracy and loss curves for Trainings C and D.

The trainings were subjected to the same tests described in Experiment 1. The results are presented in Table II.

Figure 7 demonstrates on some samples how predictions have improved and are now closer to the expected results.

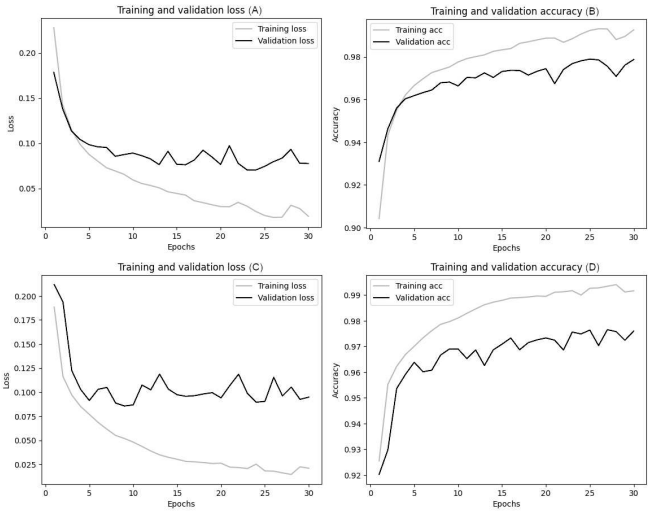


Fig. 6. (A) Training C loss. (B) Training C accuracy. (C) Training D loss. (D) Training D accuracy.

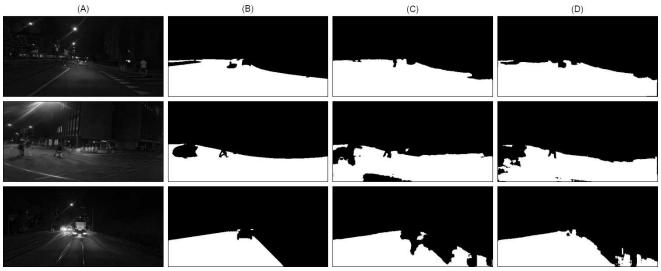


Fig. 7. (A) Original image. (B) Ground truth. (C) Training C prediction on nighttime images. (D) Training D prediction on nighttime images.

C. Experiment 3

The third experiment concatenates the LBP texture map as a fourth layer to the raster images, resulting in a four layer input: three layers representing colors and one for the LBP texture map. Two different training configurations are employed: Trainings E and F. Training E used only daytime images from Cityscapes dataset, and Training F used daytime and nighttime images from Cityscapes and ACDC datasets. The number of epochs and hyperparameters for the Batch Normalization layer were set as the following: 22 “epochs”, “epsilon” of 0.005 and “momentum” of 0.995 for Training E, and 40 “epochs”, “epsilon” of 0.01 and “momentum” of 0.99 for Training F. Figure 8 displays accuracy and loss curves for Trainings E and F.

The same tests in the previous experiments were conducted and the results are presented in Table III.

Figure 9 shows trainig F predictions on some nighttime samples. Despite training E exhibiting lower accuracy, it demonstrated better evaluation metrics than training A.

D. Results comparison

Many related works presented only the metric accuracy for their evaluation of their models. Given the large data volume, Table IV selected the best accuracy results from the cited

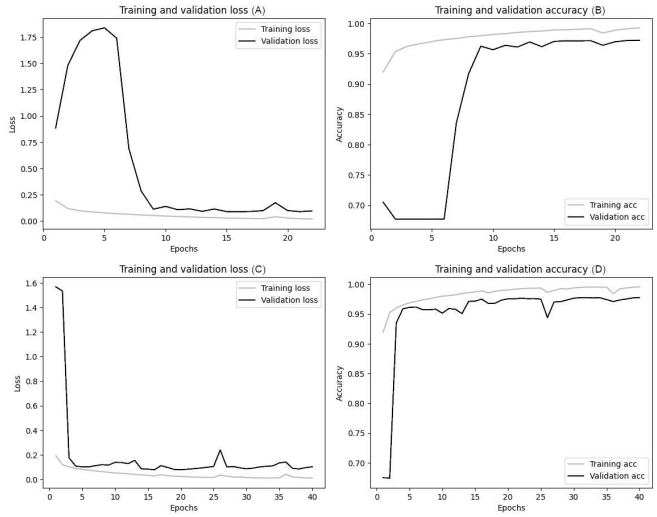


Fig. 8. (A) Training E loss. (B) Training E accuracy. (C) Training F loss. (D) Training F accuracy.

TABLE III
TRAININGS E AND F MEAN IOU AND ACCURACY

Training	Test	mIoU (%)	Accuracy (%)
E	daytime	88.9080	96.8062
	nighttime	71.9142	87.6108
	day and nighttime	81.4759	95.1977
F	daytime	90.2042	97.2200
	nighttime	91.7143	96.7658
	day and nighttime	88.0763	97.1405

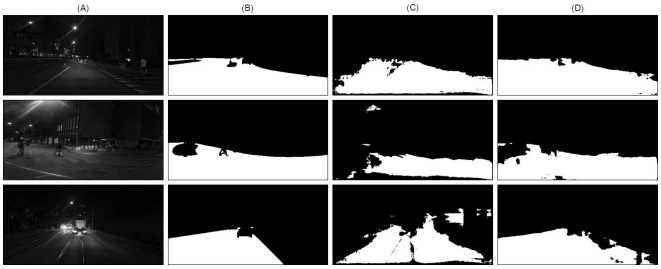


Fig. 9. (A) Original image. (B) Ground truth. (C) Training E prediction on nighttime images. (D) Training F prediction on nighttime images.

studies in subsection II-B for comparison with the findings of this work.

The results in Yousri et al. (2021) [8] surpass those obtained in this work. It is relevant to point that their work implemented their own segmentation algorithm based on graphical and geometrical analysis of the image, whereas this work handled scenarios with complex objects on the road. However, comparison with the remaining related works revealed that our methodology achieved higher accuracy rates. Several factors may contribute to this discrepancy. These also include variations in datasets, mask generation methods, but more importantly the fact that many related works opted for multi-class classification, while this work focused only in detecting which pixels belonged to road class and which did not. Additionally, the availability of powerful hardware resources, with increased memory and processing power, likely contributed to better performance of our approach.

TABLE IV
RESULTS COMPARISON

Reference	Model	Accuracy (%)
[8]	ResU-Net++ (Daylight)	99.1
	ResUNet (Daylight)	99.0
	U-Net (Daylight)	99.0
	ResU-Net++ (Night)	98.0
	ResUNet (Night)	97.1
	U-Net (Night)	97.9
[9]	U-Net ReLU	89.74
	Small U-Net Leaky ReLU	87.96
	Long U-Net (Dropout=0.7)	89.85
	SegNet	85.37
	FCN-16 (Dropout=50)	85.85
	FCN-8 (Dropout=50)	87.00
[10]	PSPNet	93.75
	FCN	87.61
	SegNet	92.89
[11]	U-Net	90.60
	U-Net VGG16 (Pre-trained)	91.53
	U-Net VGG19 (Pre-trained)	90.04
Ours	U-Net Training C	97.1669
	U-Net Training F	97.1405

V. CONCLUSION

Experiment 1 evaluated the U-Net's performance trained with daytime images under two configurations: using raster images (Training A) and using LBP texture maps (Training B). Notably, Training B achieved over 8% higher accuracy and nearly 20% higher mIoU when tested on nighttime images. This suggests that LBP texture maps effectively capture structural information on road pavements, leading to better generalization and improved predictions even in unseen low-light scenarios.

Although the highest performance overall was achieved in Training C, which used only raster images, the results were very close to Training F, which used the LBP texture map concatenated as a fourth layer onto the raster image. Interestingly, Training E, using daytime images with concatenated LBP maps, achieved over 4% higher accuracy and over 10% higher mIoU compared to Training A, which also used daytime images but only with raster data. This further reinforces the benefit of incorporating LBP texture maps, particularly for enhancing model performance across varying lighting conditions.

ACKNOWLEDGMENTS

The writing content of this work was revised by Gemini [13] for grammatically accurate sentences.

REFERENCES

[1] Greeshma, P.G. Different Approaches for Semantic Segmentation. *2020 5th International Conference On Communication And Electronics Systems (ICCES)*. pp. 938-943 (2020)

[2] Ronneberger, O., P.Fischer & Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing And Computer-Assisted Intervention (MICCAI)*. **9351** pp. 234-241 (2015), http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a, (available on arXiv:1505.04597 [cs.CV])

[3] Ojala, T., Pietikäinen, M. & Harwood, D. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*. **29**, 51-59 (1996), https://www.sciencedirect.com/science/article/pii/0031320395000674

[4] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S. & Schiele, B. The Cityscapes Dataset for Semantic Urban Scene Understanding. *Proc. Of The IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*. (2016)

[5] Zhang, H., Qu, Z., Yuan, L. & Li, G. A face recognition method based on LBP feature for CNN. *2017 IEEE 2nd Advanced Information Technology, Electronic And Automation Control Conference (IAEAC)*. pp. 544-547 (2017)

[6] Wu, H., Zhou, J. & Li, Y. Deep Generative Model for Image Inpainting With Local Binary Pattern Learning and Spatial Attention. *IEEE Transactions On Multimedia*. **24** pp. 4016-4027 (2022)

[7] Phueakjeen, W., Jindapetch, N., Kuburat, L. & Suvanvorn, N. A study of the edge detection for road lane. *The 8th Electrical Engineering/ Electronics, Computer, Telecommunications And Information Technology (ECTI) Association Of Thailand - Conference 2011*. pp. 995-998 (2011)

[8] Yousri, R., Elattar, M. & Darweesh, M. A Deep Learning-Based Benchmarking Framework for Lane Segmentation in the Complex and Dynamic Road Scenes. *IEEE Access*. **9** pp. 117565-117580 (2021)

[9] Jebamikyous, H. & Kashef, R. Deep Learning-Based Semantic Segmentation in Autonomous Driving. *2021 IEEE 23rd Int Conf On High Performance Computing & Communications; 7th Int Conf On Data Science & Systems; 19th Int Conf On Smart City; 7th Int Conf On Dependability In Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. pp. 1367-1373 (2021)

[10] Pham, T. Semantic Road Segmentation using Deep Learning. *2020 Applying New Technology In Green Buildings (ATiGB)*. pp. 45-48 (2021)

[11] Sarmah, P., Gogoi, A. & Kalita, S. Encoder Decoder Based Deep Learning Architecture For Video Scene Parsing. *2022 Second International Conference On Computer Science, Engineering And Applications (ICCSEA)*. pp. 1-6 (2022)

[12] Sakaridis, C., Dai, D. & Van Gool, L. ACDC: The Adverse Conditions Dataset with Correspondences for Semantic Driving Scene Understanding. *Proceedings Of The IEEE/CVF International Conference On Computer Vision (ICCV)*. (2021,10)

[13] AI, G. Gemini (Large Language Model). (2024)

Evaluation of LBP Learning for U-Net Based Road Semantic Segmentation

Mauricio Kendi Yui
Department of Electrical Engineering
State University of Londrina
Londrina, Paraná
Email: mauricio.yui1985@uel.br

Leonimer Flávio de Melo
Department of Electrical Engineering
State University of Londrina
Londrina, Paraná
Email: leonimer@uel.br

***Abstract*—Local Binary Pattern (LBP) is a texture descriptor used for capturing textural features such as edges, nodes, and surfaces in raster images, while improving robustness against varying lighting conditions. This paper proposes training a U-Net with the LBP texture maps extracted from the original dataset to perform semantic segmentation of road pavements.**

Comparative analysis between the model trained with LBP texture maps and the same model trained with original raster images reveals a higher accuracy of the former on untrained road pavement scenarios. This suggests the effectiveness of LBP texture extraction in enhancing the generalizability of the model.

I. Introduction

Semantic segmentation aims to delimit each element in an image and assign a class label or semantic label to every pixel belonging to that segmented region [1]. It is a suitable solution for applications that demand sophisticated and contextualized understanding of visual content.

Street and road segmentation is a recurrent problem. Many solutions have been proposed, recently reappearing due to the enthusiasm surrounding Artificial Intelligence (AI) and autonomous vehicles.

Recognizing the versatility of AI and the specialized capabilities of Deep Learning, this work employs a U-Net architecture [2] to perform semantic segmentation of asphalt pavements. However, a significant challenge in this approach lies in the compilation of a diverse training dataset that encompasses various environmental conditions.

To address this issue, the Local Binary Pattern (LBP) texture descriptor [3] is leveraged to extract LBP texture maps capturing important textural features from the dataset, while mitigating lighting variations.

Training the model with LBP texture maps simplifies pattern identification and enhances generalizability and accuracy of predictions. For instance, it improves performance in nighttime scenes, which were not included in the training dataset consisting solely of daytime samples.

II. Related Works

A. Local Binary Pattern (LBP)

The Local Binary Pattern (LBP) algorithm was introduced by T. Ojala et al. [3], designed to extract local texture features from images for classification purposes. Its operator is computationally efficient, producing a texture map highly sensitive to image details.

The LBP texture map is robust against lighting variations, and highlights structural elements in the image. These features collectively ease the identification and learning of underlying structural patterns by an artificial neural network. The LBP value is given by

$$LBP_{(x_c,y_c)} = \sum_{n=0}^{N-1} s_{(i_n,i_c)} \cdot 2^n, \tag{1}$$

in which each pixel, in turn, is assigned the center position (x_c, y_c) with intensity i_c, N is the total number of neighbor pixels, i_n is the intensity of the neighbor pixel, and s is the sign function defined as

$$s_{(i_n,i_c)} = \begin{cases} 1, \text{ if } i_n \geq i_c \\ 0, \text{ otherwise.} \end{cases} \tag{2}$$

The LBP texture map describes how each pixel compares to the surrounding pixels. Contrast changes relative to the center pixel are used to detect edges, corners, smooth or irregular surfaces at that point. Figure 1 displays an example of an image obtained from the CityScapes dataset [10] and its corresponding LBP texture map.

Fig. 1. Sample image from CityScapes [10] dataset (left) and its corresponding LBP texture map (right)

The use of the LBP texture map in training neural networks has been shown to enhance performance, as demonstrated by H. Zhang et al. [4]. A Convolutional Neural Network (CNN) trained for facial recognition achieved a higher accuracy rate when utilizing the LBP texture map as input, compared to training with raster images.

Another work by H. Wu et al. [5] presents a generative model with two distinct U-Nets concatenated to reconstruct

images containing areas with missing pixels. The first U-Net is designed to predict structural information in the missing regions, utilizing the LBP texture map of the input image. The second U-Net fills in the missing pixels with appropriate colors, using the input image along with its reconstructed LBP texture map as a guide for this process.

B. Segmentation of Asphalt Pavement

Road segmentation in raster images is a longstanding issue, with existing literature offering solutions based on methods predating AI. Notably, graphical and geometric approaches, such as those outlined by W. Phueakjeen et al. [6], remain prevalent. These methods typically involve a combination of edge filtering and subsequent application of Hough Transform to identify traffic lane boundaries.

In Deep Learning context, authors H. Jebamikyous and R. Kashef [7] presents a standard U-Net model, and four additional variants, each incorporating some adjustments. Other architectures are also explored such as Fully Convolutional Network (FCN) and SegNet. The most promising outcomes were observed in a U-Net variant that included a dropout regularization and an elongated structure with additional convolutional and pooling layers. The superior performance can be attributed to its elaborated architecture and larger number of parameters, enabling the identification of a wider range of patterns and information within images.

However, it is wise to consider that a larger number of parameters, despite achieving better performance, may also take longer to execute. In a study conducted by T. Pham [8], three models were tested: the Pyramid Scene Parsing Network (PSPNet), FCN, and SegNet. While PSPNet had higher evaluation metrics, its runtime was also longer compared to the other two. Hence, it is paramount to attend to resource constraints when selecting models, particularly in applications where processing time is critical.

Finally, P. Sarmah et al. [9] evinced that transfer learning techniques may also be applied to U-Nets. Five U-Nets were evaluated: a standard model, in which the encoder was replaced by the convolutional structure of the VGG-16 network with pre-trained weights, an identical model to the previous one but without pre-trained weights, a model where the encoder was replaced by the convolutional structure of the VGG-19 network with pre-trained weights, and an identical model to the previous one but without pre-trained weights. The U-Net network with pre-trained weights from the VGG-16 model exhibited the highest accuracy. Transfer learning techniques may serve as a means to expedite the training process and leverage optimally adjusted weights from other models.

III. Proposed Approach

The LBP texture descriptor is conventionally applied to raster image to extract the histogram of the texture map. While the histogram describes the texture features, it discards pixel position information. This work proposes using the LBP texture map for training a U-Net, as the texture map emphasizes pattern information in the image. This approach can improve and facilitate learning, while also mitigating the effects of lighting variations. Therefore, higher evaluation metrics are validated by comparing to the same U-Net trained with the original raster images.

A. The Adopted U-Net Model

The U-Net model was published by Ronneberger et al. [2]. Originally designed for biomedical purposes, it was soon realized that it is powerful for general object segmentation. The U-Net implemented in this work has a structure similar to that of the original work, and its diagram is depicted in Figure 2.

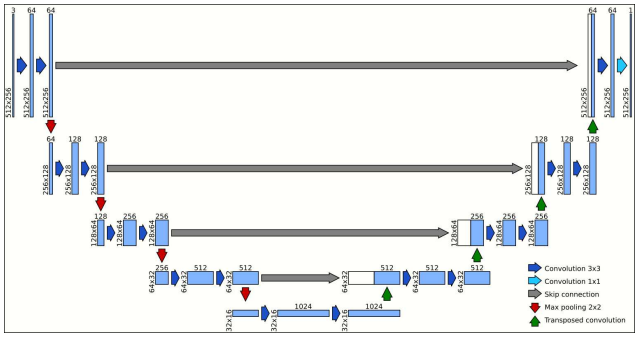


Fig. 2. U-Net architecture

The U-Net architecture (Figure 2) consists of an encoder (left side) and a decoder (right side), connected through a bridge at the bottom. Images are fed into the encoder, which has the task of learning abstract representations and producing feature maps. The encoder has four layers, each containing two convolutional filterings (blue arrows) with a Rectified Linear Unit (ReLU) activation function followed by a max pooling layer (red arrow). The max pooling layer downsamples and delivers the feature maps to the next layer. The downsampling process, while beneficial for capturing complex features, can also lead to spatial information loss.

The decoder aims to generate a segmentation mask from the abstract representation. It mirrors the encoder's structure with four layers. Each layer in the decoder upsamples the feature maps from the previous layer through a transposed convolution (green arrow) and performs two convolutional filterings. U-Net addresses the spatial information loss by incorporating skip connections (grey arrows). These connections directly copy filtered feature maps from corresponding encoder layers and concatenate them to the feature maps in the decoder. This reintroduces precise location information from the earlier stages, allowing the decoder to recover spatial details and generate more accurate segmentation masks.

The final layer in the decoder undergoes a last convolution with a sigmoid activation function (cyan arrow) to generate the output segmentation mask, representing a pixel-wise classification prediction.

B. Training

Code was developed using the Python programming language, version 3.1, on the Colab (or Colaboratory) platform provided by Google Research, a division of Alphabet Inc. Hardware resources include an Intel(R) Xeon(R) CPU @2.20GHz processor, 83.5 GB of RAM, and an A100 GPU with 40 GB of RAM. TensorFlow, an open-source library, provides functions for manipulating deep neural networks. Keras, also open-source library, offers a Python interface for accessing TensorFlow libraries.

The U-Net parameters were configured with "Adam" as the optimizer, "Binary Cross Entropy" as the loss function, and a batch size of 16. The number of training epochs was determined through experimentation based on the model's behavior according to the training method, ranging from 20 to 60 epochs, chosen to minimize loss and prevent accuracy degradation.

Two datasets were employed: CityScapes [10] and ACDC [11]. The CityScapes dataset provides 2974 images for training and 500 images for testing - these were captured in European cities during the daytime. 400 images from the ACDC dataset were used for training and 106 for testing - these were captured during night-time in various other European cities.

C. Evaluation Metrics

Two common metrics, as described in P. G. Greeshma [1], are used in this paper for evaluating semantic segmentation: Intersection over Union (IoU) and Accuracy. IoU represents the number of pixels common to both masks divided by the total number of pixels in both masks, as given by equation

$$\text{IoU} = \frac{\text{Target Area} \bigcap \text{Predicted Area}}{\text{Target Area} \bigcup \text{Predicted Area}}. \tag{3}$$

Accuracy is defined by equation

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}, \tag{4}$$

where TP are true positive, TN are true negative, FP are false positive and FN are false negative predictions. Accuracy measures all correctly identified classes and is particularly useful when all classes have equal importance.

The first experiment evaluates the robustness of LBP texture maps to lighting variations. Two U-Net models are trained: one using only daytime raster images from the CityScapes dataset (Training A), and another using the corresponding LBP texture maps from the same dataset (Training B). This comparison allows us to assess whether LBP features offer any advantages under consistent daytime lighting conditions.

The night-time test set from the ACDC dataset was subjected to the U-Net model prediction. Figure 4 displays some of the U-Net predictions to visually observe the degradation in performance.

Both test image sets from CityScapes and ACDC datasets were combined and subjected to the U-Net model prediction. All results are displayed in Table I

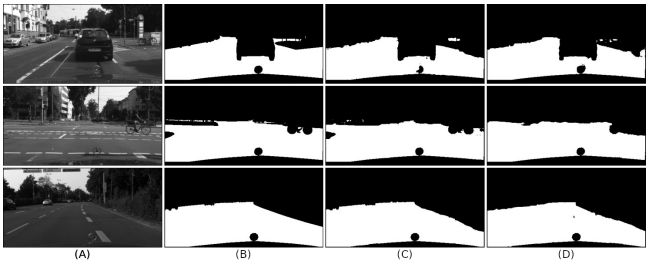


Fig. 3. (A) Original image. (B) Ground truth. (C) Training A prediction on daytime images. (D) Training B prediction on daytime images

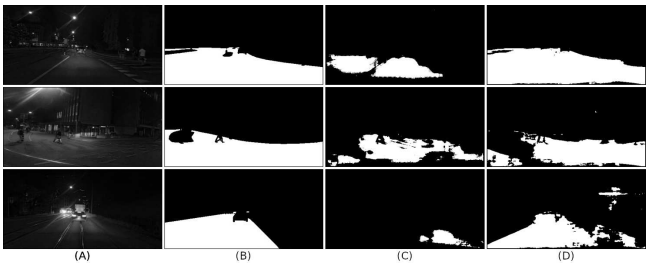


Fig. 4. (A) Original image. (B) Ground truth. (C) Training A prediction on night-time images. (D) Training B prediction on night-time images

Both IoU and accuracy exhibit a decline when tested with night-time images, attributable to the absence of night-time samples in the training data, however, in Training B, the decrease is notably less pronounced.

D. Experiment 2

The second experiment serves as a control to validate proper U-Net functioning. The U-Net undergoes two trainings: training C was conducted using raster images, while training D utilized the LBP texture maps. Both daytime and night-time images from CityScapes and ACDC datasets were used in the training processes of this experiment, and the same tests as in the previous experiment were done. The results were recorded in Table II.

Figure 5 visually demonstrates on some samples how the U-Net predictions have improved and are now closer to the expected results.

E. Experiment 3

In the third experiment, the LBP texture map is appended as a fourth layer to the raster images, thereby resulting in

TABLE I
TRAININGS A AND B MEAN IOU AND ACCURACY

Training	Input	Test	Mean IoU (%)	Accuracy (%)
A	RGB	daytime	89,5374	97,0086
A	RGB	night-time	51,8287	79,8503
A	RGB	day and night-time	77,8988	94,0073
B	LBP	daytime	89,1117	96,8720
B	LBP	night-time	78,0000	90,3514
B	LBP	day and night-time	83,1885	95,7314

TABLE II
TRAININGS C AND D MEAN IOU AND ACCURACY

Training	Input	Test	Mean IoU (%)	Accuracy (%)
C	RGB	daytime	89,9839	97,1505
C	RGB	night-time	93,1967	97,3697
C	RGB	day and night-time	88,2541	97,1888
D	LBP	daytime	88,4193	96,6471
D	LBP	night-time	89,5519	95,8343
D	LBP	day and night-time	85,8022	96,5049

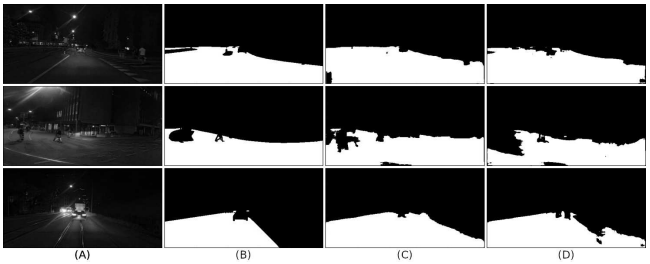


Fig. 5. (A) Original image. (B) Ground truth. (C) Training C prediction. (D) Training D prediction

training and testing inputs with four layers: three representing colors and one representing the LBP texture map. The U-Net undergoes two additional trainings: training E was conducted solely with images from the CityScapes dataset, while training F utilized images from both the CityScapes and ACDC datasets. The same tests as in the previous experiment were conducted. The results were recorded in Table III.

TABLE III
TRAININGS E AND F MEAN IOU AND ACCURACY

Training	Input	Test	Mean IoU (%)	Accuracy (%)
E	RGB + LBP	daytime	88,4135	96,6451
E	RGB + LBP	night-time	59,2926	83,4107
E	RGB + LBP	day and night-time	78,8377	94,3302
F	RGB + LBP	daytime	90,2532	97,2353
F	RGB + LBP	night-time	92,3085	96,9799
F	RGB + LBP	day and night-time	88,2609	97,1906

Figure 6 illustrates how the U-Net predictions for training F are accurate and reliable on some samples. Despite training E exhibiting lower accuracy, it has demonstrated better results than training A.

F. Results comparison

The accuracy was the only common metric published in all related works. Table IV displays the best accuracy results from related works for comparison with the results presented in previous subsections.

IV. CONCLUSION

In Experiment 1, two models were trained using daytime images, one with raster images and the other with LBP texture maps. The results demonstrated that the model trained with

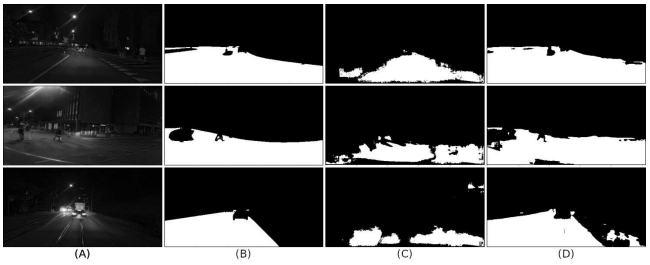


Fig. 6. (A) Original image. (B) Ground truth. (C) Training E prediction. (D) Training F prediction

TABLE IV
RESULTS COMPARISON

Reference	Model	Accuracy (%)
[7]	U-Net ReLU	89,74
	Small U-Net Leaky ReLU	87,96
	Long U-Net (Dropout=0,7)	89,85
	SegNet	85,37
	FCN-16 (Dropout=50)	85,85
	FCN-8 (Dropout=50)	87,00
[8]	PSPNet	93,75
	FCN	87,61
	SegNet	92,89
[9]	U-Net	90,60
	U-Net VGG16 (Pre-trained)	91,53
	U-Net VGG19 (Pre-trained)	90,04
Ours	U-Net Training D (LBP)	96,5049
	U-Net Training F (RGB + LBP)	97,1906

LBP texture maps exhibited over a 10% higher accuracy when tested with night-time images. This underscores the effectiveness of LBP texture maps in capturing structural information on asphalt pavement, enabling better generalization and aiding predictions even in untrained scenarios.

The highest performance overall was achieved in Experiment 3 where the LBP texture map was integrated as a fourth layer onto the raster image. This approach also improved accuracy when trained with only daytime images and tested with night-time images, emphasizing the advantage of incorporating LBP texture maps in enhancing predictive performance across diverse conditions.

Comparison with related works revealed that our methodology achieved higher accuracy rates. Several factors may contribute to this discrepancy, including differences in datasets, mask generation methodologies, and the utilization of multi-class classification in other models. Additionally, the availability of greater hardware resources, such as increased memory and processing power, likely contributed to the superior performance of our approach.

REFERENCES

[1] Greeshma, P.G. Different Approaches for Semantic Segmentation. *2020 5th International Conference On Communication And Electronics Systems (ICCES)*. pp. 938-943 (2020)

[2] Ronneberger, O., P.Fischer & Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. *Medical Image Computing And Computer-Assisted Intervention (MICCAI)*. **9351** pp. 234-241 (2015), http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a, (available on arXiv:1505.04597 [cs.CV])

[3] Ojala, T., Pietikäinen, M. & Harwood, D. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*. **29**, 51-59 (1996), https://www.sciencedirect.com/science/article/pii/0031320395000674

[4] Zhang, H., Qu, Z., Yuan, L. & Li, G. A face recognition method based on LBP feature for CNN. *2017 IEEE 2nd Advanced Information Technology, Electronic And Automation Control Conference (IAEAC)*. pp. 544-547 (2017)

[5] Wu, H., Zhou, J. & Li, Y. Deep Generative Model for Image Inpainting With Local Binary Pattern Learning and Spatial Attention. *IEEE Transactions On Multimedia*. **24** pp. 4016-4027 (2022)

[6] Phueakjeen, W., Jindapetch, N., Kuburat, L. & Suvanvorn, N. A study of the edge detection for road lane. *The 8th Electrical Engineering/ Electronics, Computer, Telecommunications And Information Technology (ECTI) Association Of Thailand - Conference 2011*. pp. 995-998 (2011)

[7] Jebamikyous, H. & Kashef, R. Deep Learning-Based Semantic Segmentation in Autonomous Driving. *2021 IEEE 23rd Int Conf On High Performance Computing & Communications; 7th Int Conf On Data Science & Systems; 19th Int Conf On Smart City; 7th Int Conf On Dependability In Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*. pp. 1367-1373 (2021)

[8] Pham, T. Semantic Road Segmentation using Deep Learning. *2020 Applying New Technology In Green Buildings (ATiGB)*. pp. 45-48 (2021)

[9] Sarmah, P., Gogoi, A. & Kalita, S. Encoder Decoder Based Deep Learning Architecture For Video Scene Parsing. *2022 Second International Conference On Computer Science, Engineering And Applications (ICCSEA)*. pp. 1-6 (2022)

[10] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S. & Schiele, B. The Cityscapes Dataset for Semantic Urban Scene Understanding. *Proc. Of The IEEE Conference On Computer Vision And Pattern Recognition (CVPR)*. (2016)

[11] Sakaridis, C., Dai, D. & Van Gool, L. ACDC: The Adverse Conditions Dataset with Correspondences for Semantic Driving Scene Understanding. *Proceedings Of The IEEE/CVF International Conference On Computer Vision (ICCV)*. (2021,10)