Sistema FIEB | SENAI CIMATEC

PELO FUTURO DA INOVAÇÃO

Centro Universitário SENAI CIMATEC

Programa de Pós-Graduação em Modelagem Computacional e Tecnologia Industrial

Doutorado em Modelagem Computacional e Tecnologia Industrial

Tese de Doutorado

Desenvolvimento de um modelo para identificar, categorizar e mapear dados precisos da qualidade de placas de trânsito de vias públicas através de câmeras veiculares.

Apresentada por: Vanessa Ferreira Dalborgo
Orientador: Prof.Dr. Roberto Luiz Souza Monteiro
Coorientador: Prof.Dr. Thiago B. Murari

Salvador, 2024

Vanessa Ferreira Dalborgo

Desenvolvimento de um modelo para identificar, categorizar e mapear dados precisos da qualidade de placas de trânsito de vias públicas através de câmeras veiculares.

Tese apresentada ao Programa de Pós-Graduação em Modelagem Computacional e Tecnologia Industrial do Centro Universitário SENAI CIMATEC como requisito parcial para a obtenção do título de Doutor em Modelagem Computacional e Tecnologia Industrial.

Orientador: Prof.Dr. Roberto Luiz Souza Monteiro
Coorientador: Prof.Dr. Thiago B. Murari

Salvador, 2024

Senai Cimatec

Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial Doutorado em Modelagem Computacional e Tecnologia Industrial.

A Banca Examinadora, constituída pelos professores abaixo listados, leram e recomendam a aprovação da Tese de Doutorado, intitulada "Desesenvolvimento de um modelo para identificar, categorizar e mapear dados precisos da qualidade de placas de trânsito de vias públicas através de câmeras veiculares", apresentada no dia 21 de Fevereiro de 2024, como requisito para a obtenção do título de Doutor em Modelagem Computacional e Tecnologia Industrial.

Orientador: _____

Prof. Dr. Roberto Luiz Souza Monteiro
SENAI CIMATEC

Coorientador: _____

Prof. Dr. Thiago B. Murari
SENAI CIMATEC

Membro interno da Banca: _____

Prof. Dr. Hernane Borges de Barros Pereira
SENAI CIMATEC

Membro interno da Banca: _____

Prof. Dr. Marcelo Albano Moret Simões Gonçalves
SENAI CIMATEC

Membro externo da Banca: _____

Prof. Dr. Marcos Figueiredo
UNEB - Universidade do Estado da Bahia

Membro externo da Banca: _____

Prof. Dr. José Garcia Vivas Miranda

UFBA - Universidade Federal da Bahia

Agradecimentos

Primeiramente agradeço a Deus.

Ao meu esposo Esmeraldo Ferreira Campos Filho, pelo amor, carinho, compreensão e tolerância durante todo este período da pesquisa; ao meu filho Leonardo Dalborgo Campos pela paciência e colaboração e à minha filha Isabella Dalborgo Campos por compreender os momentos de dedicação à pesquisa.

Agradeço a Ford Motor Company pelo apoio no desenvolvimento desta pesquisa.

Meus sinceros agradecimentos ao Colegiado do Doutorado em Modelagem Computacional e Tecnologia Industrial do SENAI CIMATEC por acreditarem no meu potencial, e especialmente para o meu coorientador Thiago B Murari e meu Orientador Roberto Monteiro pelo apoio e incentivo.

Resumo

O reconhecimento de placas de trânsito é uma das muitas utilidades possibilitadas por sistemas embarcados. Através do uso de câmeras veiculares, é possível capturar e classificar placas de trânsito com o auxílio de Inteligência Artificial, especificamente, Redes Neurais Convolucionais. Além de tratar o problema básico abordado na literatura, que consiste em detecção e classificação do tipo da placa de trânsito, a presente pesquisa abordou uma metodologia para criação de bancos de dados e modelos para avaliar a qualidade das placas, o que resulta na possibilidade de mapear esses dados através da implementação de tal sistema. Com isso, é possível melhorar a gestão da manutenção das placas de trânsito realizada pelo governo ou concessionária contratada. Como principais resultados obteve-se um conjunto de dados com 7805 imagens e 17493 placas brasileiras, além de modelos de detecção e classificação de objetos treinados com estes dados e validados com desempenho significativo. Outro resultado importante desta pesquisa é a aplicação e análise de técnicas de aumento de dados, o que se mostrou capaz de aumentar a robustez dos modelos a situações em que os dados coletados não são suficientes.

Palavras-chave: Placas de Trânsito, Banco de dados, Redes Neurais, Visão Computacional.

Abstract

The recognition of traffic signs is one of the many utilities enabled by embedded systems. Through the use of vehicle cameras, it is possible to capture and classify traffic signs with the aid of Artificial Intelligence, specifically Convolutional Neural Networks. In addition to addressing the basic problem solved in the literature, which consists of detecting and classifying the type of traffic sign, the present project achieved a methodology for creating databases and models to evaluate the quality of the signs, which results in the possibility of mapping this data through the implementation of such a system. With this, it is possible to improve the management of traffic sign maintenance carried out by the government or contracted agency. The main results obtained were a dataset with 7805 images and 17493 Brazilian plates, as well as object detection and classification models trained with this data and validated with good performance. Another important result of this research is the application and analysis of data augmentation techniques, which proved capable of increasing the robustness of the models to situations where the collected data is not sufficient.

Keywords: Traffic Signs Recognition, Dataset, Neural Networks, Computer Vision.

Sumário

Lista de Figuras

Lista de Siglas e Abreviações

ADAS - Advanced Driver Assistance Systems (Sistemas Avançados de Assistência ao Condutor)
ANN - Artificial Neural Network (Rede Neural Artificial)
API - Application Programming Interface (Interface de Programação de Aplicativos)
AV - Autonomous Vehicles (Veículos Autônomos)
BRTSD - Brazilian Traffic Signs Dataset (Conjunto de Dados de Placas de Trânsito Brasileiras)
BTSD - Belgian Traffic Sign Dataset (Conjunto de Dados de Sinais de Trânsito Belga)
CET - Companhia de Engenharia de Tráfego
CNT - Confederação Nacional do Transporte
CNN - Convolutional Neural Network (Rede Neural Convolucional)
COCO - Common Objects in Context (Objetos Comuns em Contexto)
CPU - Central Processing Unit (Unidade Central de Processamento)
CV - Computer Vision (Visão Computacional)
CVAT - Computer Vision Annotation Tool (Ferramenta de Anotação de Visão Computacional)
DDP - Distributed Data Parallel (Dados Paralelos Distribuídos)
DL - Deep Learning (Aprendizado Profundo)
DNIT - Departamento Nacional de Transportes Terrestres
DITS - Dataset of Italian Traffic Signs (Conjunto de Dados de Sinais de Trânsito Italianos)
FCN - Fully-Connected Network (Rede Totalmente Conectada)
GAN - Generative Adversarial Networks (Redes Adversariais Generativas)
GTSRB - German Traffic Sign Recognition Benchmark (Conjuntos de Dados de Sinais de Trânsito Alemães para Classificação)
HSV - Hue Saturation Value (Valor de Matiz e Saturação)
IA - Inteligência Artificial
ICF - Integral Channel Features (Recursos de Canal Integral)
ILSVRC - ImageNet Large-Scale Visual Recognition Challenge (Desafio de Reconhecimento Visual em Grande Escala ImageNet)
IOT - Internet of Things (Internet das Coisas)
MBST - Manual Brasileiro de Sinalização de Trânsito
MNIST - Modified National Institute of Standards and Technology (Instituto Nacional de Padrões e Tecnologia Modificado)
OpenCV - Open-source Computer Vision Library (Biblioteca de Visão Computacional de Código Aberto)
R-CNN - Regions with Convolutional Neural Networks (Regiões com Redes Neurais Convolucionais)
RPN - Regional Proposal Network (Rede de Propostas Regionais)
SETRAM - Secretaria Executiva de Transporte e Mobilidade Urbana
SSD - Single-Shot Detector (Detector de Único Disparo)
TFRecord - Tensorflow Record
TSR - Traffic Sign Recognition (Reconhecimento de Placas de Trânsito)
VOC - Visual Object Classes (Classes de Objetos Visuais)
YOLO - You Only Look Once
ZCA - Zero Component Analysis (Análise de Componentes Zero)

Capítulo Um

Introdução

O sistema rodoviário constitui o principal meio de transporte para escoamento de produção e circulação de pessoas no mundo. Problemas decorrentes de seu estado de conservação e manutenção acarretam uma série de implicações sociais e econômicas. No Brasil, a forte dependência do sistema rodoviário pelos sistemas logísticos para escoamento de produção, assim como para a circulação de pessoas, prejudica a conservação e manutenção da infraestrutura rodoviária no país.

Apesar da sua evidente relevância, os investimentos efetuados nos últimos anos não foram suficientes para proporcionar condições adequadas de segurança e qualidade às rodovias. Conforme o levantamento do Departamento Nacional de Infraestrutura de Transportes (DNIT) (2023), em 2021, o orgão priorizou o trabalho de manutenção das rodovias e, assim, cobriu mais de 94% dos 53,6 mil quilômetros pavimentados da malha federal com contratos de manutenção. O Brasil conta com 1,7 milhões de quilômetros de estradas, porém, apenas 13%, ou 221.820 quilômetros, estão pavimentados, e destes, 58,2% apresentam algum tipo de deficiência.

As rodovias quando em mau estado de conservação, ocasionam uma série de implicações sociais e econômicas. Para as cadeias logísticas, a má condição das placas de transito e as irregularidades do pavimento acarretam em maior custo operacional dos veículos. Além disso, os problemas do sistema rodoviário implicam em maiores gastos com manutenção dos veículos, com o consumo de combustível e tempo de viagem, além de afetar a segurança e conforto de seus usuários.

A inspeção e manutenção de placas de transito é um elemento-chave dos sistemas de gerenciamento, e os resultados trazidos proporcionam bases para que o processo de manutenção seja executado com baixa eficiência. O método mais simples de inspeção é a identificação visual de defeitos conduzidos por especialistas qualificados. No entanto, existem vários problemas dessa abordagem, e incluem altos custos, subjetividade e não receptibilidade da avaliação, além de exposição a possíveis incidentes de trânsito durante as inspeções. As deficiências acima implicam a necessidade de usar sistemas diferentes para concluir o processo de identificação automática e avaliação de defeitos. Quando os sistemas mencionados estão em uso, falhas como as mencionadas são significativamente limitadas ou mesmo eliminadas, mas podem ser substituídas por outras que pertencem à eficiência da identificação, precisão da descrição e avaliação de defeitos, bem como desempenho em tempo real (STANIEK; CZECH, 2018).

O mapeamento da qualidade das placas de transito existente em vias publica é de grande importância para que o poder público possa cobrar as empresas responsáveis para agilizar o processo de manutenção das mesmas. Desta forma, a concepção de uma aplicação capaz para indicar as coordenadas geográficas e defeitos correspondentes à essas placas, se mostra interessante a

diversos propósitos. Com a disponibilização deste tipo de dado, empresas responsáveis pela manutenção de rodovias podem utilizá-lo como critério para planejamento de futuras manutenções, definindo-se onde e quando realizar, tendo, portanto, maior controle das falhas, ganho na eficiência e redução de custos operacionais (STANIEK; CZECH, 2018).

Relatos da imprensa mostram números que indicam uma situação onerosa para os cofres públicos e de grande insegurança para as pessoas no trânsito. Por exemplo, o Jornal de Brasília (2017) relatou em 2017 que o custo médio de manutenção de placas de trânsito era de R$ 28 mil por mês, envolvendo em média 429 placas/mês. Ainda segundo o Jornal de Brasília (2017), a manutenção de placas danificadas era feita a um custo unitário de R$ 120 na época, mesmo com o reaproveitamento das chapas das placas. No caso de placas pichadas, este custo unitário era de R$ 45. Embora os dados mencionados sejam referentes ao ano de 2017, a situação atual é semelhante, conforme pode ser observado na notícia apresentada por Cardoso (2023), do *website* Metrópoles. Percebe-se, portanto, que trata-se de um problema atual, sendo de grande importância buscar alternativas para tornar a manutenção de placas de trânsito mais eficiente, reduzindo custos e o tempo de reparo, além de garantir um trânsito mais seguro.

A indústria automotiva, por sua vez, tem investido em sistemas tecnológicos para aumentar a segurança nos ambientes de trânsito. De acordo com Galvani (2019), um veículo pode ser classificado com base no nível de automação presente em uma aplicação chamada *Advanced Driver Assistance Systems* (ADAS), que é projetada para usar câmeras, radares, sistemas embarcados com conectividade com a internet e outras tecnologias que fornecem recursos que auxiliam o motorista ou até mesmo controlam o veículo automaticamente em tarefas como estacionar, frear e reconhecer placas de trânsito. Dentre as várias funções presentes no contexto do ADAS, destaca-se o reconhecimento de placas de trânsito.

Muitos estudos de placas de trânsito envolvendo métodos de IA foram recentemente relatados na literatura. Em Cao et al. (2019) e Silva et al. (2021), a etapa de detecção é baseada nas cores e formas das placas, enquanto as classes são previstas por uma Rede Neural Convolucional (CNN). No entanto, de acordo com Gu e Si (2022), Zhu e Yan (2022) e Triki, Karray e Ksantini (2023), essa abordagem não é confiável em termos de desafios TSR como a oclusão, que é o foco deste artigo. Por outro lado, modelos de aprendizado profundo bem conhecidos, como Faster R-CNN (REN et al., 2017), *You Only Look Once* (YOLO) (BOCHKOVSKIY; WANG; LIAO, 2020) e *Single Shot Detector* (SSD) (LIU et al., 2016) são investigados em Zhou, Zhan e Fu (2021), Gu e Si (2022), Zhu e Yan (2022). Esses modelos fornecem como resultado final tanto a detecção quanto a classificação (reconhecimento) de objetos, sendo mais estáveis do que as outras abordagens de detecção mencionadas e permitindo amplas possibilidades no reconhecimento de objetos, uma vez que um conjunto de dados com imagens e coordenadas dos objetos a serem reconhecidos esteja disponível (GU; SI, 2022; TRIKI; KARRAY; KSANTINI, 2023).

Existem conjuntos de dados bem conhecidos e disponíveis publicamente para detecção e classificação de sinais de trânsito. Alguns deles são os *benchmarks* de conjuntos de dados de sinais

de trânsito alemães para classificação (GTSRB, *German Traffic Sign Recognition Benchmark*) (STALLKAMP et al., 2012) e detecção (GTSDB) (HOUBEN et al., 2013), o conjunto de dados de sinais de trânsito belga (BTSD) (TIMOFTE; ZIMMERMANN; GOOL, 2014) e o conjunto de dados de sinais de trânsito italianos (DITS) (YOUSSEF et al., 2016). No entanto, existe a necessidade de criar conjuntos de dados com cenários mais complexos, uma vez que os conjuntos de dados existentes estão saturados com métodos CNN atingindo mais de 90% de precisão (SAADNA; BEHLOUL, 2017). Além disso, alguns dos tipos de sinais de trânsito nesses conjuntos de dados europeus não existem ou têm padrões diferentes em outros países como o Brasil. Esse é um dos fatores que reforça a necessidade de um conjunto de dados de referência para cada país ou região, conforme discutido por (MAGNUSSEN et al., 2020).

Em situações reais, o reconhecimento de placas de trânsito é desafiado por uma variedade de fatores, como oclusão (ou obstrução), mudanças de iluminação, apagamento, condições climáticas, etc. (WALI et al., 2019; MAGNUSSEN et al., 2020; MANNAN et al., 2022). Dentre esses fenômenos, a oclusão e apagamento merecem atenção especial. A presença de obstáculos à frente de um sinal pode dificultar o seu reconhecimento devido à visibilidade reduzida. Este é um problema muito comum que pode ser causado por pedestres, árvores, outros veículos etc. (REHMAN et al., 2017). Os sistemas anteriores apresentam soluções relevantes para esse problema, mas têm escopo muito limitado (MANNAN et al., 2022). Por exemplo, quando um sinal de trânsito tem alto nível de oclusão, pode ser irreconhecível tanto para humanos quanto para sistemas tecnológicos, levando a um tráfego inseguro.

Uma das possíveis formas de oclusão é aquela causada por vegetação, sendo relativamente comum no Brasil (DALBORGO et al., 2023). O impacto deste fenômeno em TSR tem sido um tópico de interesse entre os pesquisadores, quer seja abordado explicitamente ou não. Em Mathias et al. (2013), os autores demonstram um alto desempenho na detecção de placas de trânsito usando a abordagem *Integral Channel Features* (ICF). Entre os conjuntos de dados utilizados em Mathias et al. (2013), há o Belgian Traffic Signs Dataset (conjunto de dados de placas de trânsito belgas), que inclui instâncias de placas ocultas (incluindo ocultas por vegetação) e danificadas, as quais os autores identificam como amostras desafiadoras, uma vez que são a principal causa de detecções falhas. Da mesma forma, em Balali e Golparvar-Fard (2014), autores introduzem um *pipeline* para a detecção e classificação de placas de trânsito, trabalho no qual a oclusão causada por vegetação é também dada como um desafio. Os exemplos de trabalhos citados deixam claro que o problema é relevante e frequentemente abordado em pesquisas. É evidente, todavia, que não há trabalho que diretamente classifique a condição de oclusão por vegetação em placas de trânsito.

No contexto de *Autonomous Vehicles* (AV), alguns trabalhos investigam a oclusão de objetos de forma bastante genérica, tratando desde a detecção de placas de trânsito até a detecção de pedestres e veículos. O ambiente de condução autônoma tende a ser particularmente complexo, uma vez que os diferentes tipos de objetos de interesse criam muitos cenários específicos nos quais um modelo deve se especializar. Além da oclusão causada por objetos estáticos (como postes de iluminação e prédios), os objetos-alvo podem interocluir ou até mesmo autoocluir, conforme

analisado em Gilroy, Jones e Glavin (2021). No artigo Kim et al. (2018), é discutido como os métodos anteriores são propensos a serem afetados por oclusões e, em seguida, uma nova solução baseada em aprendizado profundo é proposta e apresentada para ser robusta para esses cenários, superando o bem estabelecido *Faster R-CNN*. Os autores em Mostafa et al. (2022), por outro lado, criam um conjunto de dados totalmente novo contendo várias instâncias ocultas, estudando o problema e analisando os recursos de detecção de arquiteturas de última geração.

É comum que ocorra, com o passar do tempo, uma gradual deterioração de placas de trânsito devido a intempéries naturais. Assim, a evolução para a condição de apagamento (ou envelhecimento) é uma questão de tempo nos casos onde não são realizadas manutenções. Esta característica é impactante no processo de classificação (WALI et al., 2019), sendo portanto estudada de modo a se criar modelos robustos a tal condição. Li et al. (2015), por exemplo, propõem uma abordagem que integra segmentação de imagem baseada em invariantes de cor e histograma de gradientes orientados, resultando num modelo essencialmente invariante a situações comumente desafiadoras, como apagamento e oclusões. Zhu et al. (2016), diferentemente, com o objetivo de superar adversidades que incluem o apagamento de cores em placas de trânsito, propõem a aplicação de um modelo baseado em *EdgeBox* e rede neural convolucional, que é validado em um conjunto de dados publicamente disponível: Swedish Traffic Signs Dataset (conjunto de dados de placas de trânsito Suecas). Por outro lado, percebe-se que, assim como no caso da vegetação, embora esta condição seja, de fato, estudada e considerada em abordagens de detecção e classificação de placas de trânsito, não existem trabalhos na literatura direcionados a desenvolver modelos que diretamente reconheçam este estado de apagamento.

Para criar um conjunto de dados eficaz, é importante garantir a variedade de imagens, entretanto, alguns desafios podem surgir. A distância entre cada placa, por exemplo, pode ser muito grande, aumentando o tempo e a dificuldade de aquisição dessas imagens. É senso comum na comunidade de desenvolvimento de IA que o *data augmentation* é um recurso poderoso para aumentar o tamanho dos conjuntos de dados e melhorar o desempenho das CNNs, recurso este que é explorado neste trabalho.

1.1 Definição do problema

A cada dia que passa a tecnologia está evoluindo para *Internet of Things* (IOT), Cidades Inteligentes e produtos que exploram a inteligência artificial, como veículos autônomos, em que a presença de câmeras veiculares e computação embarcada com conectividade podem representar uma revolução na forma como os carros são usados comercialmente, e transformando-os como sensores para salvar vidas, com uma abordagem robusta e baseada em evidências levantada pelas IAs para evitar mortes e acidentes nas rodovias do Brasil e no mundo.

Um desses usos é a manutenção de estradas, a inteligência artificial pode ser usada para identificar pontos de interesse para os proprietários dessas estradas. Estrategicamente esta tecnologia pode

ser utilizada para identificação de placas de trânsito danificadas o qual é um problema para o governo e Concessionárias de estradas e rodovias por todo o Brasil. O modelo atual estabelece que o funcionário deva verificar pessoalmente, de tempos em tempos, as placas de trânsito instaladas. Esse processo é cansativo e caro, então a Inteligência Artificial deve ser aplicada para reduzir os custos humanos e monetários. Devido ao fato de as empresas responsáveis pela manutenção das estradas não possuírem um mapeamento sobre as placas de transito existentes, acabam realizando a manutenção com uma frequência irregular e, muitas vezes, sem critério de escolha sobre onde ou quando as inspeções devem ser realizadas.

Portanto, a definição do problema consiste em estado de "desinformação"das placas de trânsito, alto custo e tempo para manutenção das mesmas gerando insegurança viária.

Capturar imagens de diferentes placas de trânsito usando uma câmera pode ser eficaz, mas leva muito tempo para construir um conjunto de dados. As placas de trânsito não são totalmente padronizadas em diferentes países, portanto, a quantidade de imagens disponíveis ao público pode ser escassa dependendo da área de interesse. O Brasil, por exemplo, apresenta significativamente menos conjuntos de dados de imagens de sinais de trânsito em comparação com os Estados Unidos ou a Europa. Por esta definição de problema a criação de um banco de dados com imagens de placas de trânsito brasileiras se fez necessário para resultados expressivos. Embora Junges, Paula e Aguiar (2019) e Silva et al. (2021) relatem a construção de um *dataset* desse tipo, até onde sabemos não há nenhum conjunto de dados brasileiro disponível publicamente.

1.2 Objetivo

Desenvolver um modelo para mapear dados da qualidade de placas de vias públicas, utilizando técnicas de visão computacional (CV, *Computer Vision*) para detectar e classificar placas de trânsito em imagens obtidas por câmeras veiculares. A classificação deve ser realizada com relação ao tipo (pare, lombada, velocidade máxima permitida, passagem de pedestres a frente etc.) e à qualidade de placas de trânsito, sendo boa ou ruim (informação visível ou não pelo motorista).

1.3 Objetivos Específicos

Os objetivos específicos desta pesquisa são:

1. Analisar o estado da arte das ferramentas para visão computacional e *deep learning*;
2. Criar um de conjunto de dados com imagens de placas de trânsito brasileiras e anotações do tipo e condição de cada placa para treinar modelos de detecção e classificação de objetos. O *dataset* desenvolvido será referido como *Brazilian Traffic Signs Dataset* (BRTSD);

3. Desenvolver um conjunto de dados utilizando processamento computacional de imagens;
4. Treinar e avaliar o modelo de mapeamento da qualidade de placas em vias públicas com o auxílio da base de dados desenvolvida;
5. Simular a captura e reconhecimento de placas com defeitos;

1.4 Importância da pesquisa

Muitas referências como Hou, Hao e Chen (2017) e Mannan et al. (2022) pesquisaram modelos distintos entre classificação dos tipos de placas e classificação dos defeitos da placa, utilizando conjunto de dados apenas com imagens Europeias. Temos diferenças das placas de trânsito entre Brasil e países europeus, o que torna o modelo proposto como tecnologia inovadora.

Além da criação do conjunto de dados de imagens de placas brasileiras teremos pesquisa para aumentar o conjunto de dados através da aplicação de filtros nas imagens coletadas o que poderá ser um diferencial relevante.

A pesquisa foi importante no desenvolvimento e estudo técnico de um modelo algorítmico para capturar em tempo real placas de trânsito defeituosas, disponibilizando os dados para a venda, gerando um novo modelo de negócio. E a aplicação deste modelo poderá ser realizada para outros estudos dentro de AI e veículos autônomos.

A solução proposta possívelmente gera consequências positivas para a sociedade nos seguintes pontos:

- Aumento da segurança no trânsito com redução de acidentes;
- Aumento da eficiência de sistemas veiculares autônomos;
- Redução na ocorrência de multas;

1.5 Motivação

- Compreender os sistemas de Redes Neurais com a aplicação de IA para capturar imagens de placas de trânsito em tempo real e fazer a separação dos dados com aplicação na indústria automotiva.
- Desenvolver a criação de um conjunto de dados de placas de trânsito brasileira com um processo desafiador e inovador.
- Com esta pesquisa, teremos o potencial de redução de acidentes de trânsito e poderemos salvar vidas.

1.6 Limites e limitações

As limitações desta pesquisa estão listadas a seguir.

- O conjunto de dados criado, BRTSD, não contempla todos os tipos de placa encontrados no Manual Brasileiro de Sinalização de Trânsito devido à ausência de alguns desses tipos nas imagens analisadas;
- BRTSD é atualmente desbalanceado em relação aos dois atributos anotados: tipo e condição das placas de trânsito. Há, por exemplo, um excesso de placas do tipo proibido estacionar e de condição boa, enquanto existe uma menor quantidade para as placas de velocidade e com condição apagada. Isto impacta na capacidade de generalização do modelo.
- Por apresentarem características e significados semelhantes, alguns tipos de placa foram agrupados em uma única classe para garantir um melhor número de amostras para os estudos;
- Foram utilizados apenas os hiper-parâmetros padrão da biblioteca de detecção de objetos;
- Não foram feitas análises em tempo real ou de desempenho quanto ao tempo de resposta dos modelos obtidos;
- Na metodologia de *data augmentation* não foi suficientemente analisado o impacto da variação da quantidade de instâncias artificiais adicionadas ao *dataset*.
- Alguns detalhes da pesquisa sobre a construção do conjunto de dados não estão presentes neste relatório devido a um processo de patente que está em andamento.

Maiores detalhes são fornecidos no Capítulo 3.

1.7 Organização do Documento

O restante do documento apresenta outros quatro capítulos e está estruturado da seguinte forma:

- Capítulo 2 - Referencial teórico:

  Apresenta os principais temas abordados durante a pesquisa, como visão computacional, redes neurais artificiais, redes neurais convolucionais. Além disso, são apresentadas as principais ferramentas utilizadas no desenvolvimento do *dataset* e dos modelos de visão computacional.

- Capítulo 3 – Metodologia:

São apresentadas as estruturas do *dataset* e dos modelos de visão computacional, bem como os procedimentos e parâmetros utilizados para a obtenção destes. Também são expostos os processos de *data augmentation* aplicados.

- Capítulo 4 – Resultados:

  A versão do final do *dataset* (BRTSD) é apresentada, bem como os resultados para metodologias de detecção de objetos com YOLO, classificação com CNN *multilabel* e aplicação dos métodos de *data augmentation* ao BRTSD.

- Capítulo 5 - Considerações finais:

  São feitas as considerações finais, sendo retomado o estudo realizado e apresentadas as conclusões do trabalho. Por fim, as Atividades Futuras de Pesquisa são sugeridas.

Capítulo Dois

Referencial Teórico

Neste capítulo serão apresentados os principais temas envolvidos no desenvolvimento da pesquisa.

2.1 Visão computacional (CV)

As aplicações de CV se desenvolveram bastante com a evolução das Redes Neurais Convolucionais (CNN, *Convolutional Neural Networks*) e dos hardwares. A CNN é um tipo de rede neural que busca simular o funcionamento da visão animal aplicando operações de convolução em imagens. As principais vantagens proporcionadas pelas CNNs são a extração de características das imagens e redução da dimensionalidade sem perda das principais informações (GéRON, 2019). Dessa maneira, a metodologia da pesquisa consiste em aplicar métodos baseados em CNNs para a obtenção do objetivo apresentado anteriormente.

Existem três tipos de tarefas que são mais comuns em CV: classificação, detecção e segmentação. Essas tarefas estão ilustradas na Figura 2.1. A classificação é um processo de definição/organização de objetos e geralmente consiste em definir uma única classe para uma imagem de entrada, a qual deve apresentar uma única instância (ou uma instância em destaque) a ser classificada. A tarefa de detecção consiste em fazer a localização e classificação de uma ou mais instâncias em uma imagem. Como resultado, são adicionadas caixas limitadoras retangulares em torno do objeto detectado (localização) e, próximo a essas caixas, é indicada a classe do objeto (classificação) e o grau de precisão associado. Já a segmentação, é o processo de atribuir classes aos *pixels* da imagem. Ao invés de localizar um objeto através de caixas retangulares, são geradas máscaras sobre os *pixels* que pertencem a uma instância (KHAN et al., 2018).

Nesta pesquisa foram abordadas as tarefas de classificação e detecção. A classificação de imagens foi implementada nesta pesquisa utilizando CNNs através da biblioteca *PyTorch* (STEVENS; ANTIGA; VIEHMANN, 2020), enquanto para a detecção de objetos foi explorada a biblioteca yolov5 (JOCHER, 2020), cuja implementação também é baseada no uso do *PyTorch*.

2.2 Rede Neural Artificial (ANN)

Para compreender em detalhes as Redes Neurais Convolucionais (CNNs), é essencial primeiro dominar as bases das Redes Neurais Artificiais (RNAs), frequentemente referidas pela sigla ANNs, correspondente a *Artificial Neural Networks* no idioma inglês. As ANNs operam com lógica mais elementar e são frequentemente um componente fundamental das CNNs. Além disso, o processo

Figura 2.1: Tarefas da visão computacional
Fonte: Manual Brasileiro de Sinalização de Trânsito e *Open Images Dataset*

de treinamento das ANNs segue uma lógica semelhante.

As ANNs são modelos computacionais que emulam o funcionamento dos sistemas neurais encontrados na biologia. Elas são parte do que é amplamente conhecido como aprendizado de máquina, que, por sua vez, constitui um ramo da inteligência artificial. Segundo a definição de Haykin (2009), as ANNs são compostas por unidades de processamento simples (nodos), distribuídas de forma paralela e interconectadas, com a capacidade de armazenar conhecimento e torná-lo acessível para utilização. São intrínsecos aos modelos ANN uma natureza não linear, uma característica de aprendizado a partir do ambiente, entre outros fatores, que os tornam adequados a diversos problemas da engenharia e ciências (SILVA; SPATTI; FLAUZINO, 2010).

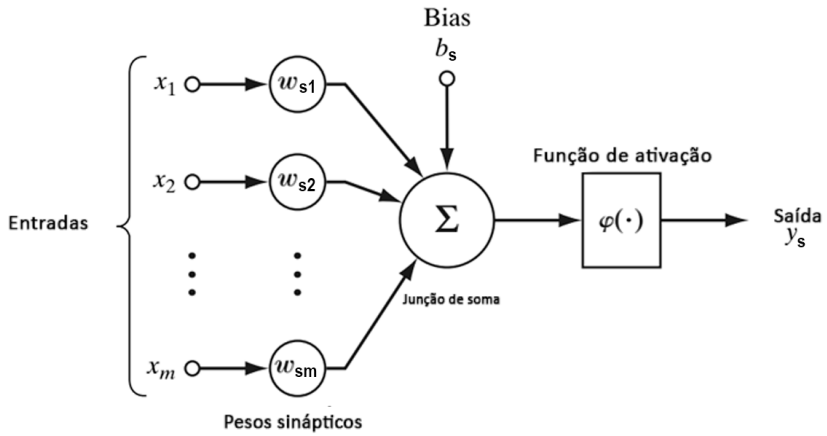
Os nodos que compõem uma ANN são denominados neurônios artificiais. A arquitetura de ANN mais simples compreende apenas um neurônio, e é denominada Perceptron de Rosenblatt (HAYKIN, 2009), que por si só pode ser considerado como um modelo de neurônio, sendo ilustrado na Figura 2.2. Nela, é ilustrado um neurônio s e m entradas (e sinapses), que são intermediadas por pesos sinápticos w. O parâmetro b_s corresponde ao *bias*, que graficamente representa um deslocamento da curva resposta y_s do neurônio.

A saída y_s de um neurônio s pode ser dada pela equação:

$$y_s = \varphi\left(\sum_{j=1}^{m} w_{sj}x_j + b_s\right), \tag{2.1}$$

sendo $\varphi(.)$ a função de ativação definida. Em termos práticos, o b_s pode ser visto como um peso sináptico que realiza o elo de ligação entre uma entrada fixa de valor 1 e o neurônio s, sendo também ajustado durante o treinamento de uma ANN. Além disso, a escolha da função de ativação $\varphi(.)$ depende da forma como a ANN está sendo aplicada, sendo frequentemente não

Figura 2.2: Perceptron de Rosenblatt



Fonte: Adaptado de Haykin (2009)

linear. Neste trabalho, por outro lado, o bloco de ANN aplicado terá neurônios com função de ativação linear – isto é, a saída de cada unidade computacional (neurônio) será igual à combinação linear $\sum_{j=1}^{m} w_{sj}x_j + b_s$, que é também chamada de potencial de ativação v (HAYKIN, 2009). Observa-se, ainda, que b_s tem o efeito de realizar uma transformação afim em v.

A maior parte das aplicações com ANNs, entretanto, envolve uma arquitetura com múltiplas camadas neurais. Neste caso, pode-se referir à ANN como um Perceptron Multicamadas. Quando o arranjo das ANNs consiste em uma arquitetura onde os nodos de entrada se projetam em direção à camada neural de saída, conforme ilustrado na Figura 2.3, pode-se dizer que trata-se de uma *feedforward network* (em português, rede alimentada adiante) (HAYKIN, 2009). Nesta configuração, os neurônios de uma camada interagem com todos os nodos da camada anterior, havendo conexão apenas entre camadas adjacentes, sem atalhos até a saída ou realimentações. As camadas entre nodos de entrada e camadas neurais de saída são denominadas camadas neurais escondidas (SILVA; SPATTI; FLAUZINO, 2010).

O treinamento de redes neurais consiste num ajuste dos seus parâmetros livres, isto é, dos pesos sinápticos e *biases* da arquitetura. Ele pode ocorrer de diferentes formas, mas uma das abordagens mais comuns e que representa o estado da arte é a de treinamento supervisionado por meio de um algoritmo de otimização, utilizado para minimizar uma função custo $\mathcal{F}$ que quantiza o erro da(s) saída(s). Uma das técnicas mais fundamentais é a descida de gradiente, frequentemente associada ao *deep learning*, consistindo num método que ajusta os pesos do modelo gradativamente até a convergência em uma solução ótima (KHAN et al., 2018). Denotando o vetor de parâmetros livres da rede neural como θ, a equação de ajuste de pesos do método descida de gradiente é dada pela equação:

$$\theta_t = \theta_{t-1} - n\nabla_\theta \mathcal{F}(\theta_t), \tag{2.2}$$

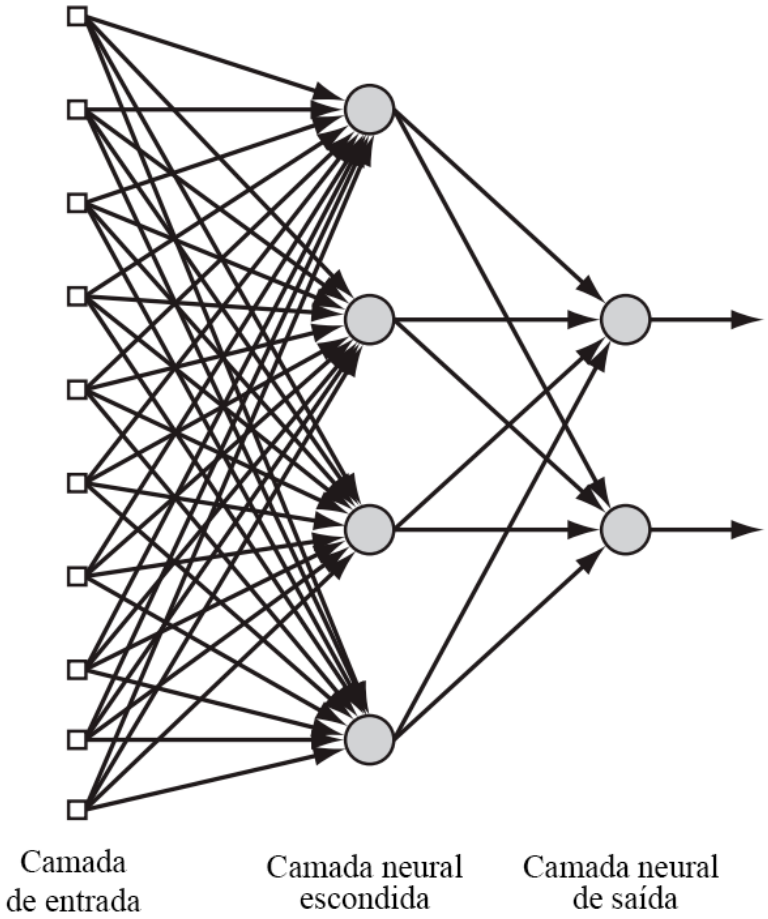


Figura 2.3: Rede alimentada adiante de múltiplas camadas
Fonte: Adaptado de Haykin (2009).

onde t é a iteração (ou época) atual, n um parâmetro ajustável denominado taxa de aprendizado, e $\nabla_\theta \mathcal{F}(\theta_t)$ o gradiente da função custo $\mathcal{F}(.)$. Vários métodos de otimização são variantes da descida de gradiente, incluindo o Adam (*Adaptive Moment Estimation*), caracterizado por combinar elementos da descida de gradiente com *momentum* e taxas de aprendizado adaptativas (KHAN et al., 2018).

2.3 Rede Neural Convolucional (CNN)

Nesta seção, são exploradas as Redes Neurais Convolucionais, que, conforme discutido anteriormente, representam uma extensão significativa das ANNs. Principalmente no contexto de tarefas de processamento de imagens, estas redes desempenham um papel crucial. A estrutura básica das CNNs é composta por camadas convolucionais, *pooling*, *flattening* e totalmente conectadas (equivalentes a ANNs), conforme ilustrado na Figura 2.4. As camadas Convolucionais são responsáveis por extrair e agrupar características como traços horizontais e verticais da imagem. Isso é feito através da aplicação de filtros de convolução sobre a imagem de entrada.

Filtros de convolução são matrizes que representam essas características e cujos elementos constituem os pesos que são ajustados durante o treinamento da rede. Vários desses filtros são aplicados em uma camada convolucional, de tal maneira que sua saída é um conjunto de matrizes (paralelepípedos na Figura 2.4) em que cada matriz consiste em um mapa de características.

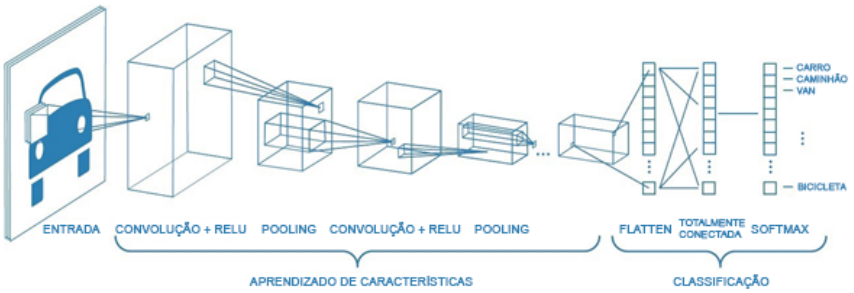


Figura 2.4: Exemplo de arquitetura de CNN
Fonte: Adaptado de MathWorks (2021).

Suponha por exemplo a seguinte matriz de entrada (imagem) I com dimensão 4x4 pixels:

$$I = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

E um filtro (kernel) K com dimensão 2x2:

$$K = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

A operação de convolução entre I e K pode ser realizada da seguinte forma:

1: Selecione a primeira submatriz da matriz de entrada I de mesma dimensão do filtro K, destacada em negrito a seguir:

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & 3 & 4 \\ \mathbf{5} & \mathbf{6} & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

2: Realize a multiplicação elemento a elemento entre a submatriz do Passo 1 e K, somando os resultados:

$$\begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = 1*1 + 2*2 + 5*3 + 6*4 = 44$$

3: Deslize o filtro K para a direita e repita a multiplicação e a soma:

$$\begin{bmatrix} 2 & 3 \\ 6 & 7 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = 2*1 + 3*2 + 6*3 + 7*4 = 54$$

4: Continue deslizando o filtro para a direita e para baixo até percorrer toda a matriz de entrada.

Cada posição corresponderá a novas multiplicações e somas. A matriz resultante completa após a convolução será:

$$\begin{bmatrix} 44 & 54 & 64 \\ 84 & 94 & 104 \\ 124 & 134 & 144 \end{bmatrix}$$

Nas camadas de *pooling* é feita a redução da dimensão da saída fornecida pela camada convolucional para condensar os mapas de características extraídos da imagem (KHAN et al., 2018). Não existem pesos associados aos neurônios nessa camada, apenas é computado o valor máximo ou a média dos elementos de uma submatriz da entrada. Por exemplo, aplicando max pooling com uma janela de 2x2 e deslocamento 2 tem-se:

$$I = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & \mathbf{6} & 7 & \mathbf{8} \\ 9 & 10 & 11 & 12 \\ 13 & \mathbf{14} & 15 & \mathbf{16} \end{bmatrix}$$

$$Max\ pooling\ (I) = \begin{bmatrix} 6 & 8 \\ 14 & 16 \end{bmatrix}$$

A camada de *flattening* transforma o conjunto de matrizes fornecidas pelo *pooling* em uma representação vetorial. Isso é necessário para que os dados extraídos pelas camadas anteriores possam ser fornecidos como entrada para uma ANN tradicional com neurônios totalmente conectados, a qual é responsável por aplicar pesos e fornecer como saída probabilidades associadas a cada classe de objeto.

O treinamento das CNNs ocorre de forma semelhante à descrita na Seção 2.2. Entre as principais diferenças, destaca-se a prática de *Transfer Learning*, frequentemente aplicado a CNNs e que consiste numa abordagem que adapta e aplica um conhecimento anteriormente adquirido numa tarefa relacionada em uma nova tarefa (KHAN et al., 2018). A adaptação a partir de um modelo pré-treinado é denominada pela literatura "*fine-tuning*", sendo principalmente útil quando se deseja treinar um modelo com muitas camadas, mas havendo em mãos poucos dados disponíveis para um contexto de muitas nuances.

A estrutura da rede neural, incluindo as camadas apresentadas acima e suas configurações, é implementada nesse trabalho utilizando a biblioteca *PyTorch* para lidar com situações mais complexas do que os exemplos apresentados acima. Esses exemplos são apenas inseridos para um melhor entendimento do leitor sobre as operações empregadas em CNNs.

A arquitetura CNN descrita pode realizar tarefas de classificação de imagens, que consistem em definir uma única classe para uma imagem de entrada, que deve ter uma única instância (ou uma instância em destaque) para ser classificada e a localização do objeto na imagem não é realizada. No entanto, a simples classificação das imagens não se enquadra no objetivo do modelo

algorítmico proposto, pois as imagens obtidas por câmeras veiculares podem conter mais de uma placa. Os métodos Faster R-CNN e YOLO, apresentados a seguir, são soluções para esse tipo de situação. Estes métodos são baseados no uso de CNNs, mas têm operações adicionais para localizar os objetos e, por isso, o *dataset* foi desenvolvido com foco nessa tarefa. A vantagem de desenvolver um *dataset* para detecção é que, a partir das anotações das coordenadas dos objetos, fica fácil gerar um *dataset* só para classificação também.

2.4 *Faster R-CNN*

De acordo com Ren et al. (2016), o *Faster R-CNN* é um detector de objetos que evoluiu dos algoritmos R-CNN (GIRSHICK et al., 2013) e Fast R-CNN (GIRSHICK, 2015), onde o termo R indica uma operação com base em regiões de possíveis objetos . É um método de dois estágios, em que no primeiro deles são feitas uma varredura na imagem e a proposição de caixas limitadoras, ou regiões de interesse, onde possivelmente existem objetos. Esse procedimento é feito por um tipo de CNN, chamado *Region Proposal Network* (RPN), que recebe como entrada um mapa de características oriundo de camadas convolucionais e aplica uma janela deslizante com caixas delimitadoras pré-determinadas, chamadas de âncoras, que são classificadas como contendo ou não um objeto. No segundo estágio, cada região de interesse passa por uma tarefa de classificação através de redes totalmente conectadas.

A Figura 2.5 mostra a arquitetura do Faster R-CNN, onde é possível observar que um único conjunto de camadas convolucionais é responsável por servir a rede RPN e a tarefa de classificação, tornando a tarefa de detecção mais rápida em relação aos métodos antecessores.

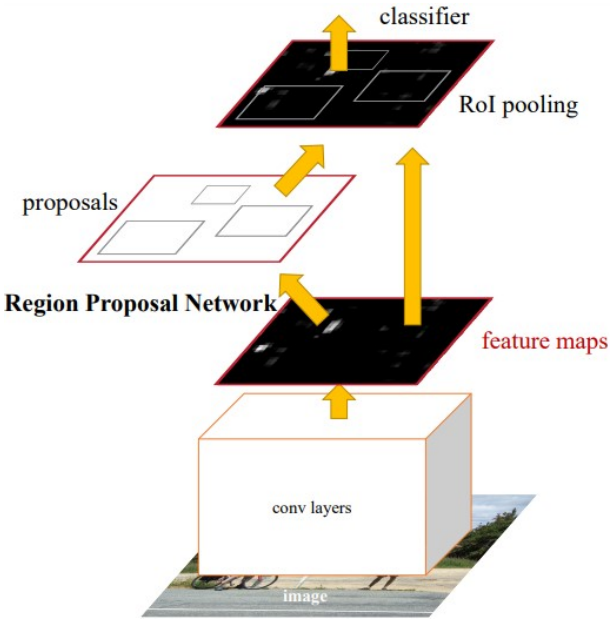


Figura 2.5: Arquitetura Faster R-CNN
Fonte: Ren et al. (2016)

De acordo com Redmon et al. (2015) o Faster R-CNN é mais lento que o YOLO, porque este é

um método de único estágio, como será apresentado na sequência. No entanto, o Faster R-CNN tende a ser mais preciso se ambos os métodos se basearem na mesma arquitetura de CNN.

2.5 YOLO

YOLO (REDMON et al., 2015), sigla para *You Only Look Once*, é uma arquitetura de rede neural moderna que realiza tarefas de detecção de objetos. Diferente de outras arquiteturas como *Faster R-CNN*, o YOLO é um detector de estágio único que prevê todas as caixas delimitadoras e as classifica em uma única etapa (REDMON et al., 2015). Assim, o YOLO pode ser usado em aplicações em tempo real, como detecção de placas de trânsito.

YOLO aborda a detecção de objetos como um problema de regressão. Conforme mostrado na Figura 2.6, ele funciona dividindo a imagem de entrada em uma grade $S \times S$ e prevendo B caixas delimitadoras com pontuações de confiança e C probabilidades de classe para cada célula da grade. Essa pontuação de confiança representa a certeza do algoritmo de que a caixa contém um objeto, e as regiões de alta pontuação possuem caixas com linhas mais grossas na Figura 2.6 (REDMON et al., 2015). Assim, cruzando as caixas delimitadoras previstas e as informações de confiança com o mapa de probabilidade de classe prevista, o YOLO produz suas saídas: uma caixa delimitadora e uma probabilidade de classe para cada detecção.

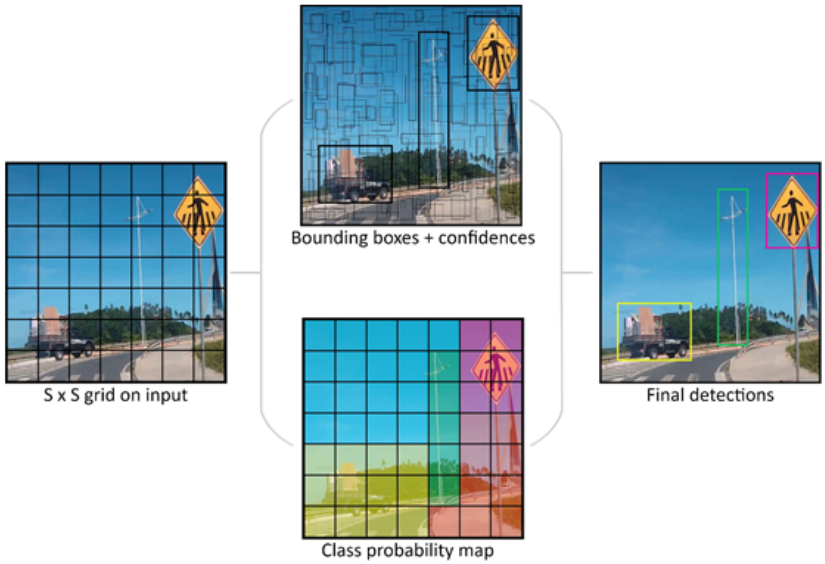


Figura 2.6: Exemplificação do processo de detecção de um modelo YOLO
Fonte: Baseado em Redmon et al. (2015).

De acordo com Redmon et al. (2015), cada *bounding box* B apresenta cinco parâmetros: x, y, w, h e confiança. As coordenadas x e y indicam, com relação aos limites da célula, o centro do *bounding box*. A largura w e a altura h são dados em função da imagem completa. A confiança

corresponde à interseção sobre a união (IoU)[1] entre a predição e os objetos anotados. Dessa maneira, as predições realizadas correspondem a um tensor de tamanho $S \times S \times (B \times 5 + C)$.

Ainda segundo Redmon et al. (2015), a *loss function* do YOLO é dada pela equação (2.3)

$$L = L_{loc} + L_{conf} + L_{cls} \tag{2.3}$$

sendo L_{loc} responsável por medir erros de detecção e dada pela equação (2.4)

$$L_{loc} = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \tag{2.4}$$

enquanto L_{conf} é dada pela equação (2.5) e avalia a confiança das detecções

$$L_{conf} = \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2 \tag{2.5}$$

e, por último, L_{cls} calcula erros de classificação de acordo com a equação

$$L_{cls} = \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \tag{2.6}$$

em que o símbolo de circunflexo denota uma variável predita pelo modelo, 1_i^{obj} é igual a 1 se um objeto aparece na célula i , 1_{ij}^{obj} indica que a j-ésima caixa delimitadora na i-ésima célula é responsável por essa predição, 1_{ij}^{noobj} é o complementar de 1_{ij}^{obj}, $\lambda coord$ e $\lambda noobj$ são pesos para os termos de coordenadas e confiança, respectivamente, $p_i(c)$ é a probabilidade de a i-ésima célula conter um objeto da classe c.

2.6 Dataset para detecção de objetos

O principal componente de um projeto de detecção de objetos é um conjunto de dados (*dataset*) com imagens e anotações referentes à localização dos objetos e aos atributos que estes possuem. É a partir deste *dataset* que o modelo de rede neural aprende a localizar e classificar os objetos de interesse.

[1] Essa medida será melhor explicada na metodologia.

Normalmente, essa tarefa de anotação é realizada com o auxílio de algum software como *Label Studio* (TKACHENKO et al., 2023), *Labelme* (WADA, 2023) ou *Computer Vision Annotation Tool* (CVAT) (CORPORATION, 2022), os quais possuem interface gráfica para desenho e atribuição de rótulos a caixas delimitadoras (*bounding boxes*), conforme ilustrado na Figura 2.7.

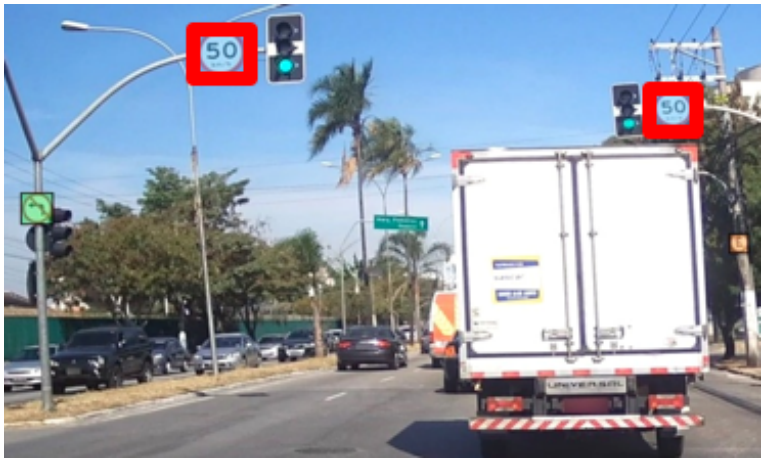


Figura 2.7: Exemplificação da anotação de placas de trânsito

Fonte: Autoria própria.

2.7 Ferramentas para visão computacional

Nesta seção serão apresentadas as principais ferramentas utilizadas na pesquisa, justificando o uso de cada uma. A pesquisa é baseada na utilização da linguagem de programação Python.

2.7.1 *FiftyOne*

FiftyOne (MOORE; CORSO, 2020) é uma biblioteca Python que reúne recursos para auxiliar os desenvolvedores a analisar e gerenciar o *dataset*. Possui uma interface gráfica com visualização das imagens em grade que permite visualizar varias imagens simultaneamente e identificar erros de anotação de maneira mais fácil. Na Figura 2.8, por exemplo, foi aplicado um filtro para mostrar somente as placas de proibido virar à esquerda e a função *to_patches()*, que recorta os *bounding boxes* das imagens exibindo apenas os objetos de interesse. No caso ilustrado, observe como é fácil identificar que existe uma placa de proibido estacionar que está com a classe errada. Realizar essa revisão abrindo imagem por imagem seria uma tarefa mais demorada e, provavelmente, com desempenho inferior.

Integrada a essa interface, o *FiftyOne* disponibiliza um grande número de funções *Python* que facilitam as interações com as imagens e anotações do *dataset* podendo, por exemplo:

- adicionar *tags* à imagens ou objetos específicos;

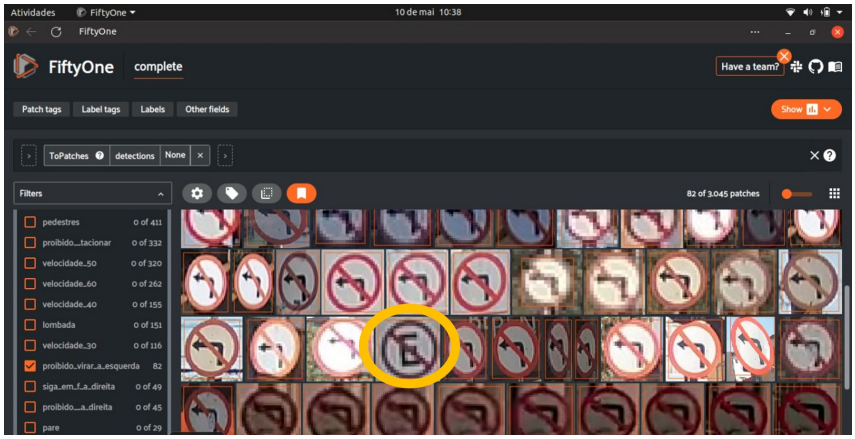


Figura 2.8: Interface de usuário do *FiftyOne*
Fonte: Autoria própria.

- filtrar imagens ou objetos que possuam classes ou *tags* específicas;
- aplicar métodos para analisar a similaridade entre imagens ou objetos;
- importar/exportar *datasets* em diversos formatos utilizados em visão computacional;
- calcular métricas de visão computacional para os modelos em análise;
- integrar os *datasets* a ferramentas de anotação como o CVAT e o *Label Studio*, auxiliando o processo de correção das anotações;

O principal elemento da biblioteca *FiftyOne* é a classe *Dataset*. Através dela é possível carregar, modificar, visualizar e avaliar *datasets* a partir de diferentes formatos de anotações. O objeto *Dataset* criado em Python reúne todas as informações referentes ao *dataset* anotado, como o caminho das imagens, as coordenadas e a classe de todas os objetos em cada imagem. *FiftyOne* permite o acesso a essas informações através da linguagem *Python* e também a partir da sua interface gráfica. O *FiftyOne* também possui métodos integrados que fazem o *download* e carregam uma coleção (*zoo*) de *datasets* públicos, como o COCO e o *Open Images*.

Um tutorial com mais detalhes da biblioteca *FiftyOne* se encontra no Apêndice B.

2.7.2 OpenCV

OpenCV (*Open source Computer Vision Library*) é uma biblioteca de funções de programação voltada para computação visual em tempo real. Originalmente desenvolvida pela Intel, a biblioteca é multi-plataforma e opera sob licença *open-source* Apache 2. Desde 2011, *OpenCV* tem suporte para operações em tempo real processadas com a utilização de GPUs. *OpenCV* possui uma variada gama de aplicações, desde pré-processamento de imagens até aplicação em inteligência artificial. No estudo da tese, *OpenCV* é utilizado extensivamente na captura e processamento

de imagens e, também, no processo de manipulação das imagens para aumentar o conjunto de dados disponível.

No sistema proposto, *OpenCV* é utilizado para redimensionar as imagens capturadas de maneira uniforme antes de ocorrer o processo de aprendizagem da rede neural, a biblioteca também é capaz de reduzir os atributos das imagens somente ao necessário para realização do processo de aprendizagem. Imagens podem possuir diversos atributos como cores, posição, tamanho, forma. *OpenCV* proporciona uma série de ferramentas que permite a manipulação dessas imagens, como por exemplo: reduzir o número de cores para uma escala de cinza, dessa forma, o tamanho e a posição relativa dos objetos na imagem são mantidos, caso a cor não seja um atributo importante no processo de classificação. Outra aplicação da Biblioteca é na aplicação de suas funções de mapeamento de *pixels* em um *bitmap*, essas técnicas são especialmente utilizadas no refinamento de segmentação de imagens, processo extensivamente utilizado em códigos como o *Mask-RCNN*.

2.7.3 Computer Vision Annotation Tool - CVAT

Considerando que para fins de detecção de sinais de trânsito existe um grande número de classes e a arquitetura yolov5 recomenda 10.000 ou mais instâncias por classe (JOCHER, 2020), esse processo de anotação pode levar muito tempo. Além disso, erros de anotação, por exemplo, espaço entre um objeto e seu delimitador, rotulagem errada, e objetos sem rótulos, devem ser evitados, o que requer atenção e revisões.

Nesta pesquisa, optou-se por utilizar o CVAT, um software gratuito fornecido pela Intel para anotação de vídeo e imagem (CORPORATION, 2022). Ele suporta anotações para classificação de imagens, detecção de objetos e segmentação por caixas de desenho, polígonos, polilinhas e/ou formas de pontos em imagens carregadas. CVAT possui uma interface fácil de usar com recursos integrados, como atalhos, zoom, rastreadores e detectores automáticos de objetos, para acelerar o processo e garantir boas anotações. Além disso, o CVAT permite um trabalho colaborativo, e o número de imagens a serem anotadas ou revisadas pode ser distribuído em uma equipe de anotadores para agilizar todo o processo e possibilitar a realização de projetos de grande porte.

O CVAT exporta as anotações em vários formatos frequentemente usados com visão computacional: *PASCAL VOC*, YOLO, COCO, *TFRecord*, etc. Assim, o mesmo conjunto de dados pode ser usado com diferentes arquiteturas de visão computacional para fins de comparação.

Se o conjunto de dados estiver sendo criado do zero, um fluxo de trabalho CVAT útil é criar um conjunto de dados primário anotando algumas imagens, usar métodos de aumento de dados, treinar esse conjunto de dados aumentado com uma arquitetura de visão computacional e carregar o modelo treinado resultante no CVAT para ajudar com anotações de novas imagens para aumentar o conjunto de dados. Esse fluxo de trabalho pode ser repetido até que o modelo esteja com a precisão necessária.

2.7.4 PyTorch

Com os avanços no campo de *deep learning* e a sua popularização, poderosas bibliotecas relacionadas vêm surgindo e se aprimorando. Entre as emergentes, se destaca o *PyTorch*, uma biblioteca de código aberto programável em *Python* que enfatiza flexibilidade e simplicidade, fornecendo todos os blocos construtivos que são necessários para desenvolver redes neurais (como as ANNs e CNNs) e treiná-las (PASZKE et al., 2019).

Características relevantes sobre o *PyTorch* englobam: possibilidade de realização de computações aceleradas por meio de GPUs, o que permite realizar operações muito mais rapidamente que uma CPU (*Central Processing Unit*, ou Unidade Central de Processamento, em português); o fornecimento de recursos que permitem otimização numérica em expressões genéricas frequentemente usadas em *deep learning*, como o *autograd*.

PyTorch trabalha primariamente com o tipo de dado Tensor para armazenar números, vetores e matrizes, os quais são operados via funções convenientemente definidas, se assemelhando com a popular biblioteca *NumPy*. Os módulos essenciais desta biblioteca podem ser acessados via *torch.nn*. Tem-se, por exemplo, a classe básica *torch.nn.Module*, segundo a qual todas as arquiteturas de redes neurais são criadas via *PyTorch*. Esta biblioteca torna simples a execução de processos que não são comumente aplicados facilmente no Python, como processamento paralelo. Com o *PyTorch*, pode-se efetivamente organizar dados de forma paralela, de acordo com diferentes possíveis estratégias de amostragem, através do instanciamento da classe *DataLoader* (STEVENS; ANTIGA; VIEHMANN, 2020).

Capítulo Três

Metodologia

Neste capítulo é apresentada a metodologia empregada em cada etapa da pesquisa: criação do *dataset* e obtenção dos modelos de detecção de objetos.

3.1 Conjunto de dados (dataset)

Essa seção apresenta os aspectos teóricos e técnicos referentes à criação do conjunto de dados (*dataset*) necessário para a aplicação de TSR. A metodologia e as ferramentas utilizadas nesta tarefa são a base deste trabalho e podem ser empregadas em outros trabalhos de visão computacional. A criação do dataset é necessária devido a diferenças de formatos, cores, símbolos e classes das placas nacionais em relação a datasets públicos como os datasets da Alemanha. Uma análise dessa situação é apresentada no Apêndice A.

Baseado na metodologia de *human-in-the-loop* para anotação de Wu et al. (2022), foi estabelecido um fluxo de trabalho para acelerar a criação do *dataset* conforme apresentado no fluxograma da Figura 3.1. Esse processo consiste em criar um conjunto de dados primário anotando algumas imagens manualmente, treinar esse conjunto de dados com uma arquitetura de visão computacional e utilizar o modelo resultante para fazer as novas anotações, que devem ser corrigidas pela equipe sempre que necessário. Repete-se o processo à medida que se obtenha novas imagens, atualizando o modelo e o *dataset* final a partir da validação de ambos.

A seguir serão detalhadas as metodologias empregadas em cada uma das etapas do fluxograma da Figura 3.1.

3.1.1 Atributos e classes

Basicamente, o TSR consiste em análise de imagens para localizar placas de trânsito e classificá-las corretamente. Como um diferencial em relação à literatura atual, esta pesquisa apresenta modelos que, além de classificar placas de trânsito de acordo com o seu tipo, também são capazes de indicar a condição dos objetos encontrados. Assim, é necessário que esses dois atributos (tipo e condição) estejam disponíveis para cada placa presente no *dataset*. Cada um desses atributos tem um conjunto de classes específico. O tipo de uma placa pode ser, por exemplo, 'semáforo à frente' ou 'proibido estacionar', enquanto a condição pode ser 'boa' ou 'apagada'.

O caso mais complexo associado ao reconhecimento de placas de trânsito seria um veículo comple-

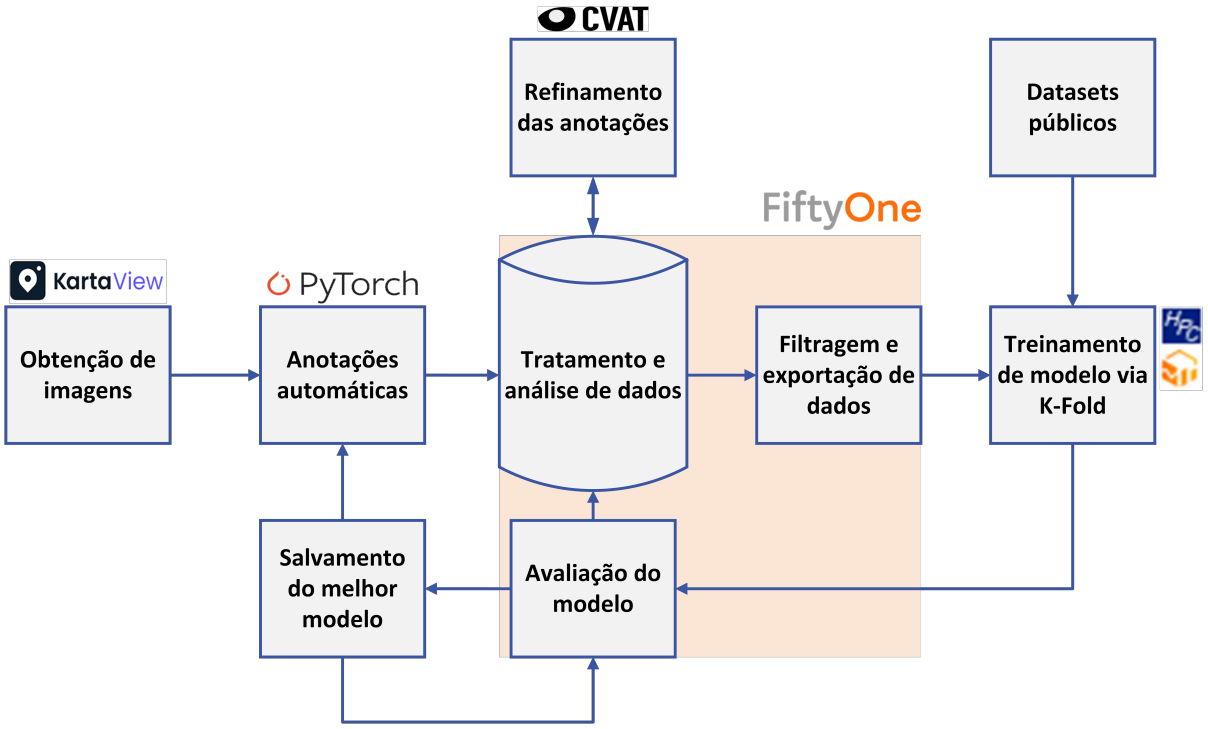


Figura 3.1: Fluxograma geral para criação do dataset e treinamento de modelos de detecção.
Fonte: Autoria própria.

tamente autônomo. Nessa aplicação, é necessário que o veículo reconheça precisamente qualquer tipo de placa presente no código de trânsito da localidade em que trafega. No entanto, durante o período da pesquisa realizada, alguns tipos de placas do MBST não foram encontrados e outros foram encontrados em pequena quantidade. Por isso, o escopo da aplicação é limitado às classes encontradas, havendo necessidade de ampliação do *dataset* para aplicações mais complexas como a citada anteriormente.

A escolha das classes de condição foi feita a partir do estudo dos serviços de manutenção de trânsito. Além disso, condições naturais também danificam muito as placas com ferrugem, apagamento e crescimento da vegetação. O *dataset* possui as seguintes classes referentes à condição da placa: boa, apagada e oclusão por vegetação. Vale ressaltar que os *datasets* publicamente disponíveis, como o GTSDB, não possuem anotações referentes a esse tipo de atributo.

Com relação ao tipo da placa, o *dataset* conta com as classes apresentadas na Figura 3.2, sendo elas: a) altura máxima, b) animais, c) curva, d) duplo sentido, e) estacionamento, f) estreitamento, g) exclusivo ciclistas, h) exclusivo ônibus, i) lombada, j) pare, k) pedestres, l) preferencia, m) proibido circulação, n) proibido estacionar, o) proibido parar e estacionar, p) proibido retornar a esquerda, q) proibido virar a direita, r) proibido virar a esquerda, s) rotatória, t) rotatória a frente, u) semáforo, v) sentido obrigatório, w) sentido proibido, x) siga em frente ou a direita, y) siga em frente ou a esquerda, z) velocidade 10, aa) velocidade 20, ab) velocidade 30, ac) velocidade 40, ad) velocidade 50, ae) velocidade 60, af) velocidade 70, ag) velocidade 80, ah) velocidade 90, ai) velocidade 100 e aj) outros tipos.

Ainda sobre a Figura 3.2, algumas classes como curva (c), estreitamento (f) e pedestres (k) concentram sinalizações que possuem significados diferentes de acordo com o Manual Brasileiro de Sinalização de Trânsito (MBST), mas, para garantir uma boa quantidade de amostras por classe, foram agrupadas dessa maneira por possuírem características físicas semelhantes (cores, desenhos, formatos etc.) e por representarem informações que provocam níveis de alerta parecidos ao motorista. A classe outros tipos (aj) refere-se a tipos de placas que não estavam definidas no CVAT ou *FiftyOne* durante as anotações e que têm poucas instâncias. Essas placas precisam receber pelo menos uma classe genérica para a correta validação dos modelos e para que possam ser utilizadas em trabalhos futuros como classes específicas.

Figura 3.2: Amostras dos tipos placas do dataset.
Fonte: Autoria própria.

Com relação à placas cobertas por vegetação, é necessário que pelo menos parte da placa esteja visível para que ela possa ser anotada. A Figura 3.3 mostra as imagens presentes no conjunto de dados. Nas placas a), b) e c) é fácil determinar que estamos lidando com sinais de proibição de

estacionamento, apesar da obstrução. Nesse caso a placa recebe anotação do tipo 'proibido estacionar' e condição 'vegetacao', por exemplo. A razão pela qual são classificadas como vegetação, embora a vegetação identificada causadora de oclusão seja bastante esparsa do ponto de vista mostrado, é que, de outras perspectivas, ela também pode ser consideravelmente densa e tornar esses sinais irreconhecíveis, conforme discutido em Hirt et al. (2022). Além disso, a detecção de tal caso permitiria antecipar a manutenção, evitando assim que a vegetação cresça e obstrua completamente a placa. Ser capaz de identificar sinais de trânsito parcialmente obstruídos, portanto, constitui o cenário ideal para o método proposto. Já nas placas d) e f), a densidade da vegetação é tão alta que os sinais são quase irreconhecíveis. Para esses casos são atribuídas a classe 'vegetacao' como tipo e condição.

a) b) c) d) e) f) g) h)

Figura 3.3: Amostras de placas cobertas por vegetação presentes no dataset desenvolvido.
Fonte: Autoria própria.

As placas apagadas seguem uma metodologia semelhante. As placas totalmente irreconhecíveis recebem classe 'apagada' como tipo e condição, enquanto as reconhecíveis são anotadas com o tipo adequado e condição 'apagada'. A Figura 3.4 apresenta algumas situações de placas apagadas presentes no *dataset*.

a) b) c) d) e) f) g) h)

Figura 3.4: Amostras de placas apagadas presentes no dataset desenvolvido.
Fonte: Autoria própria.

3.1.2 Aquisição de imagens

As imagens iniciais do *dataset* foram obtidas com equipamentos proprietários dos membros desta pesquisa. No entanto, foi identificada uma plataforma de mapeamento, conhecida como *KartaView*, que disponibiliza imagens de trânsito gravadas por usuários para download. Foi desenvolvido um código que utiliza os modelos treinados nos *datasets* anteriores para analisar imagens através da API (*Application Programming Interface*) do *KartaView* e as imagens que tenham os objetos de interesse são armazenadas com pre-anotações. Dessa maneira, o processo de anotação torna-se mais rápido ao mesmo tempo em que é possível verificar os resultados fornecidos pelo modelo.

Um aspecto importante do *KartaView* é que as imagens fornecidas são processadas para tornar irreconhecível o rosto dos indivíduos e as placas dos carros.

O código desenvolvido encontra-se no Algoritmo 5 do Apêndice C, que pode ser resumido nas etapas a seguir:

1. Carrega os modelos de detecção e classificação;
2. Busca uma lista de trajetos no KartaView;
3. Seleciona um trajeto da lista do item anterior e cria um dataset do *FiftyOne* com a numeração do trajeto;
4. Acessa a lista com as URLs de cada imagem do item anterior;
5. Abre cada imagem da lista anterior com a biblioteca PIL e aplica os modelos do item 1. Caso seja detectada alguma placa na imagem, a imagem é salva no diretório do trajeto e adicionada com as detecções no dataset *FiftyOne* criado no item 3;
6. Depois de analisada todas as imagens do item anterior, volta ao passo 3 selecionando um novo trajeto.

3.1.3 Anotações

Existem muitos padrões de anotação, podendo ser utilizados formatos de arquivos como .txt, .json, .xml, .csv etc. Neste trabalho optou-se por utilizar o padrão *FiftyOneImageDetectionDataset* (MOORE; CORSO, 2020) com arquivos em formato .json por permitir armazenar uma maior quantidade de informações sobre cada objeto, o que é de grande importância já que a pesquisa envolve a classificação de mais de um atributo das placas de trânsito. Além disso, esse padrão é nativo da ferramenta *FiftyOne*, utilizada durante a pesquisa para o gerenciamento do *dataset*.

Conforme mencionado anteriormente, nesta pesquisa cada placa no *dataset* possui dois atributos: tipo e condição. Assim, as ferramentas *FiftyOne* e CVAT devem ser estruturadas para essa situação. No caso do *FiftyOne*, as anotações estão registradas em um *field* de nome *detections*, onde o tipo e a condição das placas podem ser armazenados, por exemplo, em *detections.detections.type* e *detections.detections.condition*, respectivamente. Já no CVAT, define-se um conjunto de classes e em cada classe podem ser incluídos atributos customizados. Assim, o conjunto de classes contém os tipos de placa e as condições são atributos destes.

As tarefas de anotação no CVAT são criadas e configuradas a partir do *FiftyOne*, incluindo o *upload* de imagens e anotações. O arquivo *label_ schema.json* no repositório da pesquisa deve ser importado no código *Python* e passado como parâmetro ao utilizar o método *dataset.annotate()*.

Esse arquivo contém as listas dos tipos e condições de placas e outros parâmetros para configurar automaticamente a criação da tarefa no CVAT, de modo que ela apresente o layout da Figura 3.5.

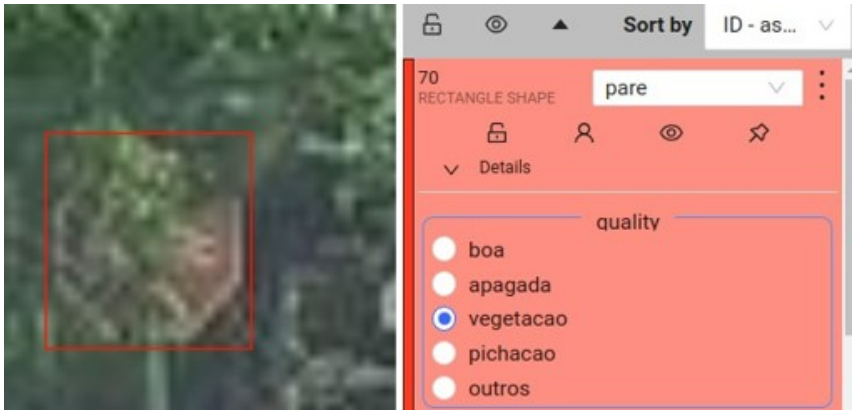


Figura 3.5: Layout de anotações no CVAT.
Fonte: Autoria própria.

O processo de anotação ocorre em duas etapas. Inicialmente, um modelo pré-treinado é utilizado para gerar anotações automáticas. Esta etapa é dedicada a diminuir o trabalho humano. Corrigir conjuntos de dados pré-anotados é mais rápido e confiável do que anotar todas as imagens desde o início. O modelo usado para pré-anotação é substituído a cada nova iteração do fluxograma da Figura 3.1. À medida que cada lote de anotações expande o conjunto de dados, o modelo evolui. A segunda etapa, melhoria de anotação, é feita com o software CVAT para corrigir os erros de classe e as incompatibilidades de caixa delimitadora feitas pela IA na etapa anterior ou durante o processo de validação dos modelos.

Depois de concluídas as as melhorias no CVAT, as anotações podem ser atualizadas no *dataset* do *FiftyOne* e exportadas no formato *FiftyOneImageDetectionDataset* para a pasta *Dataset* no repositório da pesquisa.

3.1.4 Descarte de imagens

Como o processo de aquisição de imagens é automatizado, podem surgir algumas situações em que placas de trânsito em uma determinada imagem podem ser difíceis de serem reconhecidas, tanto por humanos quanto por computadores. Essas imagens devem ser evitadas no *dataset*, pois podem atrapalhar o treinamento e validação dos modelos. As situações levadas em conta nesta pesquisa são:

- Placas cortadas pela borda da imagem como na Figura 3.6;
- Placas muito distantes/pequenas;
- Placas borradas/embaçadas por falta de foco da câmera.

Figura 3.6: Exemplo de placa irreconhecível cortada pela borda da imagem.
Fonte: Autoria própria.

Uma única placa não reconhecível em alguma dessas situações já inviabiliza o uso da imagem que a contém para fins de detecção de objetos. As placas pequenas e as próximas às bordas da imagem podem ser encontradas com uma rotina em Python que utilize as informações dos *bounding boxes*, como no Algoritmo 1 do Apêndice C.

Já as placas borradas/embaçadas, são encontradas com a ajuda do *FiftyOne* no modo *Patches-View* ou durante a anotação no CVAT.

3.1.5 Validação do dataset

De acordo com Joseph (2022), durante um projeto de *deep learning*, o *dataset* pode ser dividido em três grupos diferentes: treinamento, validação e teste. O primeiro é utilizado para ajustar os pesos da rede neural a partir do erro de inferência sobre este próprio conjunto de dados, ou seja, a rede aprende a partir do *dataset* de treinamento. O conjunto de validação é utilizado para avaliar as métricas do modelo em dados 'não vistos' no treinamento a fim de escolher os pesos que melhor se ajustem a esse conjunto de dados ou analisar a mudança de hiper-parâmetros da rede. Por último, o grupo de teste é voltado para uma avaliação final do modelo sem nenhum tipo de viés, mesmo aquele indireto relacionado à escolha dos pesos durante a validação.

O modelo escolhido a partir do grupo de validação pode ajudar a corrigir erros de anotação nesse próprio conjunto ou em outro de teste, pois esse modelo pode identificar, por exemplo, objetos não anotados ou com atribuição errada de classe. Não se pode dizer o mesmo do conjunto de treinamento, visto que os pesos da rede serão ajustados para atender os dados desse grupo,

mesmo que ele contenha erros.

Uma opção para validar todo o *dataset* consiste em utilizar a técnica *K-fold* de validação cruzada, em que o *dataset* a ser treinado é dividido em k partes (ou *folds*) distintas e treinado também k vezes. Conforme ilustrado na Figura 3.7, o primeiro treinamento é realizado utilizando o *Fold* 1 em azul como validação e os outros *folds* em verde são de treino. No segundo treinamento, o *Fold* 2 passa a ser o de validação e os *folds* 1, 3, 4 e 5 de treino. O procedimento se repete até que todos os *folds* sejam validados. Dessa maneira, é possível avaliar o *dataset* inteiro baseado nos modelos obtidos, fazendo as correções necessárias.

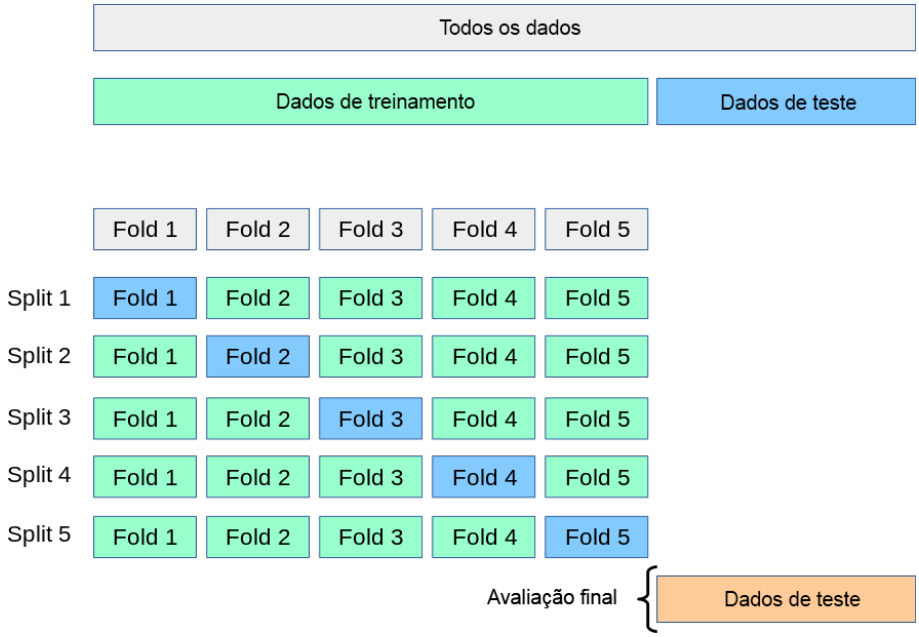


Figura 3.7: Estrutura da validação cruzada *K-fold*.
Fonte: Adaptado de https://scikit-learn.org/stable/modules/cross_validation.html

3.1.6 Data augmentation (Aumento de dados)

O processo de treinamento de um sistema de inteligência artificial envolve o uso de um conjunto robusto de dados para identificar ou classificar. A precisão desse sistema depende em grande parte de duas variáveis: a quantidade de dados usados para treinamento e a qualidade dessas informações. O desempenho melhora com a inclusão de dados de baixa qualidade, se os dados tiverem informações úteis para classificação (WANG; KLABJAN, 2016). Por exemplo, *templates* baseados em texto tiveram melhor desempenho quando o *Google* forneceu um conjunto de dados de trilhões de palavras, mesmo que as páginas não fossem filtradas e apresentassem muitos erros (HALEVY; NORVIG; PEREIRA, 2009). Quando a quantidade de informação é muito pequena ou quando a informação passada é apresentada em um contexto limitado que não abrange os diferentes cenários em que a rede será utilizada, há tendência de ocorrência do *overfitting*.

Overfitting é a incapacidade de um sistema de IA generalizar os dados usados no treinamento

para classificar dados ainda não vistos. Essa ocorrência é facilmente visualizada pela alta precisão nos dados de treinamento e pelo baixo número de acertos nos dados de teste (YING, 2019). Um exemplo disso é uma rede treinada com imagens de placas de trânsito coletados em uma área rural, mas não consegue reconhecer as mesmas placas em uma área urbana. Uma possível razão para isso é porque todas as imagens de placas de treinamento possuem vegetação ao fundo, a rede considera que a vegetação contém informações e tem um desempenho ruim quando aplicado a imagens com prédios ao fundo.

Vários métodos têm sido propostos para reduzir o impacto do *overfitting* (WANG; KLABJAN, 2016). Por exemplo, o aprendizado de transferência usa uma rede convolucional que foi previamente treinada em dados com valores ou estruturas semelhantes aos dados do problema que serão usados na etapa de treinamento. Desta forma, alguns parâmetros de rede são calibrados para resolver o problema específico e menos dados são necessários para atingir uma alta taxa de acerto.

Outra maneira de garantir a precisão desse conjunto é ter muitas informações coletadas em todas as circunstâncias possíveis. Para o sistema que identificará placas de trânsito defeituosas, isso significa ter imagens de todos os tipos de placas de trânsito que podem ser coletados em diferentes condições de iluminação (noite, dia, crepúsculo, etc.), diferentes condições climáticas (sol, chuva, neblina, etc.), diferentes qualidades de foto (alta resolução, distorcida pela velocidade, desfocada pela trepidação da câmera, etc.).

Devido às limitações de tempo e recursos, é impraticável capturar fotos em todas essas condições. A captura por meio de mapas, métodos de *web scraping* ou captura de vídeo pela própria pesquisadora levaria a muitas placas de trânsito capturadas por dia, com boa qualidade, em tempo claro e com boa resolução. Isso levaria a um sistema tendencioso de reconhecimento de placas "boas" de trânsito, quando o objetivo principal da construção do conjunto de dados é criar uma base capaz de reconhecer placas defeituosas. Para atingir esse objetivo, é importante que haja um equilíbrio nos dados, com uma quantidade de placas de trânsito defeituosas na mesma ordem de grandeza da quantidade de placas de trânsito em bom estado.

Falhas devido à escassez de dados são bastante comuns. Normalmente, à medida que o problema se torna cada vez mais especializado, aumenta a dificuldade de obter uma quantidade razoável de dados para resolvê-lo (WANG; KLABJAN, 2016). Por exemplo, na indústria médica a falta de dados dificulta a classificação do tipo de câncer (VASCONCELOS; VASCONCELOS, 2017). Além disso, os pequenos *players* do setor de IA geralmente não têm recursos para capturar esses dados ou até mesmo comprá-los (WANG; KLABJAN, 2016).

Superar a escassez de dados é o objetivo das técnicas de aumento de dados, um campo de estudo de maneiras de aumentar a quantidade de dados usados no treinamento. Este não é um campo de estudo novo, sua primeira aplicação bem sucedida pode ser considerada a introdução de defeitos dentro do conjunto de dados MNIST (BAIRD, 1992), que mostrou resultados positivos em vários

problemas como: uso de conhecimento especializado (VASCONCELOS; VASCONCELOS, 2017), classificação de relacionamento (XU et al., 2016) e mais usual, classificação de imagens (WONG et al., 2016). A precisão na classificação das imagens aumentou, mesmo com o uso apenas de transformações tradicionais, em todas as tarefas que foram avaliadas (WANG; KLABJAN, 2016)

Atualmente, os métodos existentes para aumento de imagens podem ser amplamente separados em duas categorias: métodos de caixa preta, nos quais redes neurais profundas são usadas para gerar novos dados, e métodos de caixa branca mais tradicionais, nos quais as transformações sobre dados coletados são descritas explicitamente (MIKOŁAJCZYK; GROCHOWSKI, 2018).

Os métodos de caixa branca são os mais populares e os mais testados na literatura (MIKOŁAJCZYK; GROCHOWSKI, 2018). Nesses métodos, uma cópia da imagem de entrada é submetida a uma combinação de transformações para alterar as informações da imagem de entrada (BJERRUM, 2017). Essas informações geralmente estão relacionadas à geometria e cores da imagem (WANG; KLABJAN, 2016). Por exemplo, as mudanças geométricas na imagem são obtidas através de transformações afins e podem ser reflexão, deslocamento, rotação, zoom, escala ou cisalhamento. Além disso, as alterações relacionadas às paletas de cores da imagem são distorção, nitidez, desfoque, equilíbrio de cores, contraste, brilho e alterações de matiz ou combinação de imagens (GALDRAN et al., 2017; WANG; KLABJAN, 2016).

Os métodos tradicionais de aumento de dados devem ser suficientes para começar a construir um conjunto de dados robusto de imagens de placas de trânsito, pois podem gerar variações significativas nos elementos que dificultam a detecção de placas de trânsito: condições da estrada e condições das placas de trânsito. A solução escolhida para aumentar os dados obtidos a partir de informações capturadas pela pesquisadora e retiradas da internet foi utilizar um processo de aumento de dados de caixa branca. As operações serão realizadas nos dados disponíveis para gerar imagens artificiais que serão utilizadas durante o treinamento. Essas operações, também chamadas de máscaras, podem ser aplicadas sobre toda a imagem (como no caso de mudança de iluminação ou clima) ou apenas sobre a área da imagem onde uma placa é identificada (como em ferrugem ou obstrução por vegetação). Quando apenas a área da placa de trânsito é alterada, é necessário identificar os *pixels* que lhe pertencem. Isso é feito importando as anotações já feitas na imagem original para o programa de ampliação e passando a máscara apenas na sub-imagem limitada pela anotação.

Os métodos de aumento de dados serão desenvolvidos utilizando primariamente a biblioteca *OpenCV* na linguagem *Python*. Esta biblioteca foi criada pela Intel e é completa tanto em pacotes de software quanto em integração de hardware para visão computacional. Três dos métodos desenvolvidos são exemplificados na Figura 3.8 (gerada via Algoritmo 6 do Apêndice C).

Figura 3.8: Exemplos de métodos de modificação da região da placa. Da esquerda para a direita: imagem original, placa apagada/envelhecida artificialmente e placa obstruída por vegetação artificial

Fonte: Autoria própria.

3.2 Criação e desenvolvimento dos modelos

Conforme mencionado anteriormente, um sistema veicular para reconhecimento de placas de trânsito necessita aplicar tanto detecção quanto classificação de objetos. Inicialmente, o foco da pesquisa era o desenvolvimento do *dataset* em paralelo com a obtenção de modelos de detecção utilizando o método YOLO de acordo com o fluxograma da Figura 3.1 e as anotações das placas com defeito eram únicas, ou seja, todas as placas das Figuras 3.3 e 3.4 eram classificadas somente como vegetacao ou apagada. Essa abordagem levanta a possibilidade de detectar as placas com necessidade de manutenção e foi publicada pela equipe da pesquisa em Dalborgo et al. (2023), embora esse artigo apresente apenas a classe *vegetacao* como má condição. Com o desenvolvimento do *dataset* e absorção de conteúdos relacionados, resolveu-se investigar a classificação simultânea do tipo e da condição da placa. Para isso, foram consideradas as duas abordagens a seguir:

1. YOLO com dupla anotação, ou seja, cada placa tem dois *bounding boxes* coincidentes, mas um classificado com o tipo e outro com a condição;
2. CNN multilabel, que possui duas FCNs depois da camada de *flattenning* da CNN, sendo cada FCN responsável por classificar um atributo da placa.

Para a primeira abordagem utilizou-se o *framework* yolov5 (JOCHER, 2020). Já a CNN multilabel foi desenvolvida com a biblioteca *PyTorch* e não realiza detecções, mas, como será visto mais adiante, essa estrutura sempre fornece uma classe para cada atributo, enquanto a primeira abordagem pode não realizar uma anotação dupla nos objetos. Então, a CNN multilabel necessita de um detector, que pode ser o próprio yolov5, como uma etapa inicial.

3.3 Treinamento do yolov5

A abordagem de treinamento é ajustar os modelos padrão yolov5 listados na Tabela 3.1 com seu número de parâmetros. Todos esses modelos são pré-treinados no conjunto de dados COCO.

Tabela 3.1: Quantidade de parâmetros das arquiteturas do yolov5

Arquitetura	**Parâmetros (M)**
yolov5n	1.9
yolov5s	7.2
yolov5m	21.2
yolov5l	46.5
yolov5x	86.5

De acordo com Jocher (2020), para treinamento do modelo é necessário obter o repositório do yolov5 e instalar as bibliotecas necessárias:

```
git clone https://github.com/ultralytics/yolov5  # clone
cd yolov5
pip install -qr requirements.txt
```

Para treinamento utilizando validação cruzada o *dataset* precisa estar estruturado da seguinte forma:

```
<dataset_dir>/
    dataset.yaml
    images/
        fold0/
            <uuid1>.<ext>
            <uuid2>.<ext>
            ...
        fold1/
            <uuid3>.<ext>
            <uuid4>.<ext>
            ...
        ...
    labels/
        fold0/
            <uuid1>.txt
            <uuid2>.txt
            ...
        fold1/
            <uuid3>.txt
            <uuid4>.txt
            ...
        ...
```

onde <uuid#> é o nome do arquivo, .<ext> é a extensão da imagem (.jpg, .png) e dataset.yaml é um arquivo contendo classes e o caminho dos arquivos. O Algoritmo 8 do Apêndice C é utilizado para separar as *folds* e exportar o *dataset* na estrutura acima. Já o treinamento é executado pelo Algoritmo 7, que gerencia os treinamentos preparando os arquivos do *dataset* para que as *folds* de validação sejam trocadas corretamente.

Em Dalborgo et al. (2023), uma versão anterior do *dataset* desenvolvido nesta pesquisa foi treinado por 1200 épocas com o modelo yolov5m. Foi verificado que as perdas (*loss*) foram mínimas entre 500 e 600 épocas, o que implica além de 600 épocas não ocorrem melhorias no desempenho do modelo. Esse valor de 600 épocas ainda é adotado nos resultados apresentados no próximo capítulo. Cada treinamento é realizado com hiper-parâmetros padrão yolov5.

3.4 CNN Multilabel

Nesta seção, é detalhada a metodologia desenvolvida para o treinamento e avaliação de uma CNN *multilabel*. São descritos os métodos envolvidos no processo de treinamento e na criação e utilização de um dataset apropriado, além da arquitetura aplicada.

3.4.1 Arquitetura do modelo

O modelo de CNN *Multilabel* utilizado é baseado na arquitetura da *ResNet-50*, referência em tarefas de visão computacional. Especificamente, à estrutura central do *ResNet-50* (o *backbone*) são incorporadas duas camadas totalmente conectadas: uma para a classificação do tipo da placa, a outra para classificação da condição, conforme ilustração na Figura 3.9.

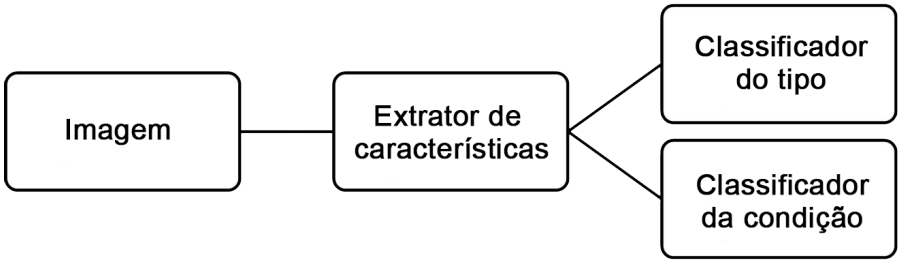


Figura 3.9: Ilustração concisa da arquitetura aplicada para a CNN *multilabel*, onde os blocos *Sign Type* e *Sign Condition* representam a saída para cada FCN
Fonte: Autoria própria.

Sendo assim, aproveita-se do bloco de *Feature Maps* advindo da *ResNet-50* para eficientemente extrair características de imagens, então classificando-as por meio de duas camadas lineares separadas. Além disso, entre o bloco de *Feature Maps* e classificadores, é aplicado *flattening*

(achatamento), responsável por transformar tensor multidimensional num tensor unidimensional, e uma camada de *dropout* (taxa de 0.2) como uma medida de regularização. O código para a definição do modelo de CNN *multilabel* utilizado é exibido no Apêndice C, Algoritmo 9 do Apêndice C.

3.4.2 Definição do dataset

Para o treinamento e avaliação do modelo CNN *multilabel*, é criado um *dataset* consistindo numa coleção de imagens de placas de trânsito, cada uma associada às *labels* (rótulos) de tipo e condição, de forma a adequar este conjunto à arquitetura descrita na Seção 3.4.1. Assim, é criada uma classe de *dataset PyTorch*. Esta é responsável por ler um *dataset*, a partir de um diretório ou via *FiftyOne*, e retornando, para cada amostra, um tensor respectivo e o tipo e condição da placa de trânsito.

Como medida de pré-processamento, algumas operações são feitas antes de fornecer as imagens ao modelo CNN. Técnicas padrão são aplicadas, como o redimensionamento, para manter um tamanho consistente das instâncias, conversão para RGB, normalização, entre outros. Além disso, o *dataset* é, por padrão, aumentado com a aplicação transformações afins, que tem como objetivo aumentar a robustez do modelo treinado.

3.4.3 Treinamento e validação

Para o treinamento, é implementada a classe *Trainer*, que recebe entradas como *dataloaders* de treino e validação, o modelo instanciado, entre outros.

O processo de treinamento é visualizado com uso do módulo *Tensorboard*, que registra a variação das métricas com o passar das épocas e permite realizar análises posteriores. Todos os treinamentos são executados com o uso de múltiplas GPUs através da abordagem DDP (*Distributed Data Parallel*) do *PyTorch*.

Os códigos de treinamento e avaliação podem ser visualizados nos Algoritmos 10, 11 e 9 do Apêndice C.

Figura 3.10: Tipos de erros e acertos
Fonte: Autoria própria.

3.5 Métricas de avaliação

3.5.1 Classificação

São utilizadas as métricas *precision*, recall, *F1-score* e *accuracy* para análise do modelo. Para o cálculo dessas métricas, é necessário quantificar primeiro os acertos e erros do modelo em um dataset. A Figura 3.10 mostra alguns exemplos com as possibilidades de erros e acertos das inferências realizadas por um modelo. O verdadeiro positivo (TP) se refere à predição correta da classe à qual um objeto pertence. Um verdadeiro negativo (TN) indica que um modelo acertou o que o objeto não é. Um falso positivo (FP) é gerado quando o modelo errou a classe à qual o objeto pertence. Por último, um falso negativo (FN) corresponde a um erro do modelo em inferir que o objeto não pertence a uma classe, mas na verdade pertence.

A precisão é calculada por meio da equação (3.1) e indica se as predições do modelo são boas ou ruins nas classes do conjunto de dados.

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

Recall é dado pela equação (3.2) e avalia se o modelo encontra todos os objetos de interesse.

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

F1-score é a média harmônica de precisão e *recall*, calculada pela equação (3.3).

$$F_1 = 2 * \frac{precision \times recall}{precision + recall} \tag{3.3}$$

Definindo *support* como sendo a quantidade de amostras a serem validadas, é possível calcular TP e FP conhecendo-se o *precision* e o *recall* e através das seguintes equações.

$$support := TP + FN \tag{3.4}$$

$$TP = recall \times support \tag{3.5}$$

$$FP = \frac{TP}{precision} - TP \tag{3.6}$$

Precision, *Recall* e F1-score são calculados para cada classe. Essas métricas são calculadas através do método *classification_report* do pacote python *scikit-learn* (PEDREGOSA et al., 2011). Este método também retorna a média micro, a média macro e a média ponderada de cada uma dessas métricas. Eles estão definidos da seguinte forma:

- Micro: Esta abordagem calcula métricas de maneira semelhante a (3.1), (3.2) e (3.3), mas considera as contagens totais de TPs, FNs e FPs em vez de contagens por classe. Quando a lista de classes de predição corresponde à lista de classes do conjunto de dados e apenas uma predição é feita por amostra, as micro médias de *precision*, *recall* e *f1-score* são iguais e correspondem à accuracy dada pela equação 3.7.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\text{observações corretas}}{\text{todas as observações}} \tag{3.7}$$

- Macro: Primeiro, calcula as métricas (3.1), (3.2) e (3.3) para cada classe. Em seguida, é obtida a média não ponderada de cada métrica nas classes C, como na equação (3.8).

$$\text{média } macro = \frac{1}{C} \sum_{i=1}^{C} \text{métrica}_i \tag{3.8}$$

- Ponderada: Este método primeiro calcula as métricas (3.1), (3.2) e (3.3) para cada classe e, em seguida, multiplica esses valores pelos correspondentes contagem de amostras da classe N_i e calcula a média dos resultados sobre o número total de amostras S, conforme demonstrado na equação (3.9).

$$\text{média } \textit{ponderada} = \frac{1}{S}\sum_{i=1}^{C} \text{métrica}_i \times N_i \tag{3.9}$$

Para determinar um conjunto ideal de pesos ao longo das épocas de treinamento para o classificador multilabel desenvolvido, é utilizado o Hamming Loss (HL) conforme descrito em (3.10) (PAL; SELVAKUMAR; SANKARASUBBU, 2020). O HL quantifica a proporção de previsões incorretas em relação ao produto de N e L, onde N significa o número de amostras no conjunto de dados e L denota o número de rótulos. Consequentemente, seleciona-se o modelo com o menor valor de HL do conjunto de validação.

$$Hamming\ loss = \frac{1}{N \times L}\sum_{i}^{N}\sum_{l}^{L}(y_{il} \neq \hat{y}_{il}) \tag{3.10}$$

3.5.2 Detecção

Além de *precision*, recall e *F1-score* conforme já apresentado acima para a classificação, as métricas utilizadas para avaliar modelos de detecção de objetos incluem *average precision* (AP) e *mean average precision* (mAP) (PADILLA; NETTO; SILVA, 2020). No caso de detecção, um TP é caracterizado por coordenadas de caixa delimitadora e atribuição de classe corretas, caso contrário, um FN e/ou falso positivo FP é gerado. TNs não são observados porque representam todas as caixas delimitadoras que não contém nenhum objeto de interesse em uma imagem, e há muitos deles.

Uma caixa delimitadora de previsão está correta quando suas coordenadas estão próximas das coordenadas da caixa delimitadora dos objetos anotados ou *ground truth* (GT). Nas tarefas de detecção de objetos esta condição é comumente avaliada através da interseção sobre união (IoU) das caixas delimitadoras previstas e GT, conforme representado por (3.11). O IoU tem valores que variam de 0 (*caixa delimitadora* com intersecção nula em relação à referência) e 1 (exata coincidência entre caixa delimitadora e a referência, isto é, o *ground truth*).

$$IoU = \frac{area(B_{gt} \cap B_p)}{area(B_{gt} \cup B_p)} \tag{3.11}$$

onde B_{gt} é o *bounding box* anotado e B_p é o *bounding box* predito. O valor limiar (*threshold*) de IoU de um modelo detector define o valor mínimo para que a detecção seja considerada. Isto é, não atingir o valor limiar significa em descarte da detecção (visto que o objeto não é eficientemente localizado).

Na Figura 3.11 são apresentados GTs em verde e predições em vermelho. Cinco situações são observadas conforme descritas a seguir:

Figura 3.11: Exemplos de detecções
Fonte: Autoria própria.

1. IoU aceitável, mas atribuição de classe errada: FN para o GT e um FP para a predição;
2. IoU não aceitável e atribuição de classe errada: FN para o GT e um FP para a predição;
3. Predição sem objeto correspondente: FP para a predição;
4. IoU aceitável e atribuição de classe correta: TP;
5. GT sem predição: FN para o GT.

Para cada objeto previsto, o detector fornece uma pontuação de confiança, que mostra a certeza do modelo de que uma caixa delimitadora contém um objeto de uma classe específica.

O AP é calculado de acordo com a avaliação de estilo COCO (*Common Objects in Context*) por meio da biblioteca *FiftyOne*, calculando correspondências para 10 limites de IoU de 0,5 a 0,95 com etapas de 0,05 e média (precisão, recuperação) de pontos sobre as pontuações de confiança. mAP é a média dos valores de AP em todas as classes.

3.6 Data augmentation para tarefa de classificação

Os modelos de *machine learning* dependem de uma quantidade razoável de amostras para um treinamento seguro e eficaz. O aumento do número de instâncias para as diferentes classes melhora o desempenho do modelo, desde que as amostras utilizadas sejam representativas do ambiente de aplicação. No entanto, coletar muitas amostras relacionadas pode ser um processo demorado, inconveniente ou até mesmo impossível. Nesses casos, técnicas de *data augmentation* surgem como uma possibilidade para aumentar o conjunto de dados, gerando amostras novas

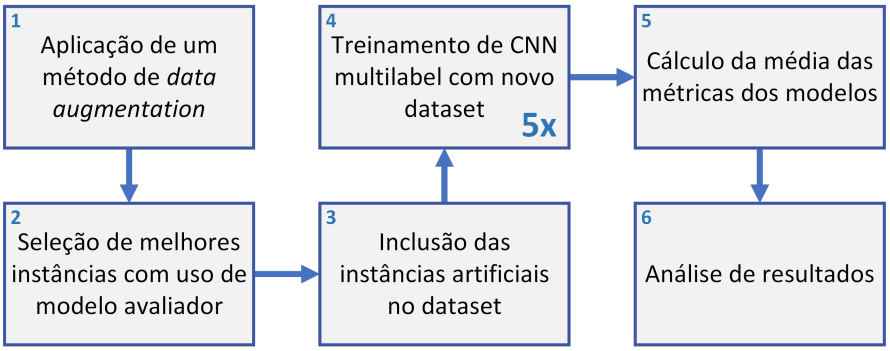


Figura 3.12: Encadeamento de etapas para a aplicação e avaliação dos métodos de *data augmentation* desenvolvidos

Fonte: Autoria própria.

a partir de amostras originais. Nesta pesquisa, o objetivo é classificar placas de trânsito de acordo com sua condição, e cada condição de placa deve estar bem representada nos dados de treinamento para que o modelo generalize de forma precisa (BRINK et al., 2017). É importante ressaltar que o uso de dados não representativos pode prejudicar o desempenho do modelo. Portanto, neste estudo, exploramos o impacto do *data augmentation* na adição de instâncias artificialmente modificadas no treinamento do modelo de classificação de placas de trânsito.

O processo prestes a ser descrito corresponde, como um todo, ao fluxograma da Figura 3.12. Esta sequência de passos é seguida de forma individual para cada método.

3.6.1 Seleção das melhores instâncias artificiais

Para selecionar os melhores exemplos de placas artificiais obtidas via *data augmentation*, utilizamos um modelo avaliador previamente treinado com dados reais. O modelo avaliador é utilizado para identificar as placas artificiais que podem ser classificadas com alta confiança, indicando quais delas são realistas o suficiente para serem utilizadas no treinamento de outras redes, posteriormente permitindo aumentar o tamanho do *dataset*. Este processo de seleção corresponde aos blocos 1 e 2 no fluxograma da Figura 3.12. Os passos envolvidos nesta abordagem são melhor descritos a seguir:

1. Aplicar um dos métodos de *data augmentation* em placas compatíveis.
2. Utilizar um modelo avaliador previamente treinado e validado com dados reais para selecionar exemplos que se parecem mais com placas reais.
3. Filtrar as placas classificadas pelo modelo avaliador que apresentarem alta confiança em relação à classificação, tanto em relação à condição quanto ao tipo da placa, por meio do *FiftyOne*.

4. Realizar análise das placas artificiais filtradas para efetuar o último ajuste fino, descartando qualquer caso que não pareça adequado para o treinamento, ainda que tenha sido detectado pelo modelo avaliador.

5. Adicionar a seleção de placas ao conjunto de dados de treinamento de um novo modelo e comparar as métricas de desempenho obtidas com as métricas do modelo treinado anteriormente no conjunto de dados original, sem o uso de *data augmentation*.

3.6.2 Procedimentos para a avaliação dos métodos de data augmentation

Devido à natureza aleatória da inicialização dos pesos dos modelos e da camada de *dropout*, existente entre *feature maps* e as FCNs (conforme descrito na Seção 3.4.1), modelos treinados com o mesmo *dataset* tendem a ter resultados diferentes, ainda que possuam exatamente os mesmos hiper-parâmetros. Esta variabilidade tende a aumentar quanto menor for o conjunto de dados utilizado. Neste cenário, se faz necessário criar estratégias para se assegurar de que cada método seja testado suficientemente.

Propõe-se que cada dataset testado seja utilizado em 5 treinamentos sequenciais. Desta forma, se explora melhor a variabilidade dos resultados, o que permite calcular medidas estatísticas como média, assegurando maior confiança na análise dos resultados. Tal procedimento também deve ser aplicado no *dataset* original, no qual não se considera o uso dos métodos de *data augmentation* descritos nesta seção, permitindo uma comparação justa entre as diferentes versões do *dataset* postas em prova.

Em suma, cada treinamento resultará em um relatório que possui métricas de *recall* respectivas, tanto para o tipo quanto para a condição das placas. Os 5 relatórios para cada *dataset* são utilizados como um todo para realizar uma análise estatística das métricas coletadas (Algoritmo 12 do Apêndice C). Estes procedimentos correspondem aos blocos 4 e 5 do fluxograma na Figura 3.12.

3.6.3 Métodos de data augmentation desenvolvidos

Nesta seção, o sequenciamento de passos necessário para a aplicação de cada algoritmo de *data augmentation* desenvolvido é detalhadamente descrito. Estes métodos representam o bloco 1 do fluxograma na Figura 3.12.

3.6.3.1 Método 1: Sobreposição de vegetação artificial

O Método 1 consiste em simular a oclusão das placas de trânsito por meio da sobreposição de imagens de vegetação artificial em placas reais em boas condições. Seu algoritmo pode ser acessado no Apêndice C (Algoritmo 2). Para aplicá-lo, são seguidos os seguintes passos:

1. Filtrar imagens de placas reais em boas condições e com área de *bounding box* maior que 40^2 *pixels*, utilizando a biblioteca *FiftyOne*.
2. Selecionar aleatoriamente uma das imagens filtradas no passo 1 e escolher uma das 16 vegetações para sobrepor.
3. Redimensionar a vegetação selecionada, utilizando a biblioteca *OpenCV*, para adequar as dimensões às da placa, utilizando as informações do *bounding box* da própria placa.
4. Sobrepor a imagem de vegetação na região do *bounding box* da placa de trânsito.
5. Aplicar duas vezes, aleatoriamente, filtro de borrão gaussiano ou mediano na região de *bounding box*
6. Repetir os passos 2 a 5 até iterar sobre todas as placas reais obtidas no passo 1.

Para melhorar a qualidade das instâncias criadas a partir do Método 1, é aplicado um pós-processamento na região do *bounding box*, por meio da utilização de dois filtros de borrão gaussiano ou mediano, que têm como objetivo eliminar serrilhamentos da vegetação artificial e tornar a imagem mais realista. O passo de pós-processamento é executado após a sobreposição da imagem de vegetação na placa de trânsito, como descrito no passo 5.

3.6.3.2 Método 2: Reuso de instâncias com vegetação

O Método 2 consiste em utilizar os registros de placas reais ocultas por vegetação para aumentar a quantidade de instâncias da classe. O processo envolve a substituição de placas em outras amostras de imagem. Seu algoritmo pode ser acessado no Apêndice C (Algoritmo 3). Abaixo está a descrição do processo executado por ele:

1. Instanciar um objeto/*view* como a filtragem das amostras que possuam uma placa real em boas condições, utilizando o *FiftyOne*.
2. Instanciar um objeto/*view* como a filtragem de todas as instâncias de placas reais ocultas por vegetação, utilizando o *FiftyOne*.
3. Copiar uma imagem dentre as filtradas no passo 1, gerando uma duplicata temporária dos dados.

4. Selecionar aleatoriamente uma placa oculta por vegetação dentre as filtradas no passo 2.
5. Redimensionar o recorte da placa selecionada no passo 4, utilizando o *OpenCV*, para que suas dimensões sejam iguais às da placa real selecionada no passo 3.
6. Substituir a cópia da placa real, do passo 3, pelo recorte da placa oculta obtida no passo 5.
7. Aplicar, de forma aleatória e utilizando o *OpenCV*, efeitos como borrão gaussiano, borrão de movimento (utilizando um *kernel*) e borrão mediano ao novo exemplo de placa oculta por vegetação obtido no passo 6. Este pós-processamento deve ser executado duas vezes.

3.6.3.3 Método 3: Apagamento/envelhecimento artificial

Com este método, busca-se aplicar ajustes de contraste, saturação e brilho, além de ruídos e borrões mediano e gaussiano, para simular placas que estejam com apagamento total ou parcial. Neste caso, os efeitos são aplicados a uma região elíptica delimitada pelo *bounding box* da respectiva placa, o que tende a selecionar com precisão a região da mesma (normalmente redonda, mas tendo formato elíptico na maior parte das perspectivas). Apesar da escolha de uma máscara elíptica, ela também é conveniente para diferentes formatos, como o da placa "Pare", por exemplo. O algoritmo desenvolvido pode ser acessado no Apêndice C (Algoritmo 4) e é descrito em detalhes abaixo:

1. Filtrar, via *FiftyOne*, amostras que possuam apenas uma placa real em boas condições (ou seja, que já não esteja apagada/avariada) e com área de *bounding box* maior que 40^2 *pixels*.
2. Para a placa do passo 1, definir, via *OpenCV*, uma máscara elíptica de forma que a região da elipse seja delimitada pelo *bounding box* de cada placa selecionada.
3. Através da máscara definida, copiar a região espacial aproximada das placas selecionadas.
4. Aplicar ruído aleatório a todos os *pixels* da região do passo 3.
5. Converter escala de cor da imagem para HSV e reduzir a saturação aleatoriamente dentro de um determinado intervalo de possíveis valores.
6. Converterescala de cor da imagem para RGB e aplicar uma transformação linear, utilizando dois parâmetros (*alpha* e *beta*), que variam aleatoriamente dentro de um intervalo, para reduzir contraste da região da placa e aumentar o brilho, tornando mais difícil a visualização da mesma.
7. Aplicar ruído gaussiano na região elíptica (região aproximada da placa).
8. Definir uma nova máscara elíptica com 60% das dimensões iniciais para especificar a região central da placa.

9. Utilizar a máscara do passo 8 para borrar, através de borrão gaussiano e mediano, as partes escritas da placa.

10. Definir uma máscara elíptica com dimensões 10% maiores que as dimensões iniciais e, a partir desta região, aplicar o borrão mediano de modo a criar uma transição relativamente suave entre a região de aplicação do efeito de apagamento e a região externa.

11. Substituir a placa original pela placa artificialmente apagada, a partir de operações *bitwise* do *OpenCV*.

3.6.4 Trabalhos de data augmentation correlatos

Embora a metodologia geral desenvolvida na Seção 3.6 seja original, a aplicação de *data augmentation* para melhorar o balanceamento do *dataset* não é uma prática nova ou incomum.

Em Zhang et al. (2020), por exemplo, vários métodos de *data augmentation* são explorados de modo a simular situações reais desafiadoras em placas de trânsito e amplificar o desempenho de um modelo R-CNN. Entre os 15 métodos aplicados, inclui-se a adição de ruído (também considerado no Método 3, Seção 3.6.3.3) e de borrão (aplicado em todos os 3 métodos da Seção 3.6.3). Os autores fazem uso dos métodos no tão conhecido *benchmark* GTSRB.

Outros autores se aproveitam de técnicas mais recentes e robustas, como o uso de Redes Adversárias Generativas (GAN) por Soufi e Valdenegro-Toro (2019). Num trabalho onde se aplica um modelo de arquitetura *pix2pix*, treinado tanto com imagens reais quanto sintéticas, possibilita-se a geração de novas imagens de placas de trânsito, aumentando-se a quantidade de instâncias disponíveis no conjunto de dados e sendo aprimorada a tarefa de classificação. Os autores ainda comparam esta técnica a métodos de *data augmentation* tradicionais, como a variação de contraste em placas circulares. Embora esta tradicional técnica seja aplicada no Método 3 (Seção 3.6.3.3), a forma de aplicação e propósito são diferentes. No presente estudo, diferentemente, tal variação de contraste é feita sempre de modo exacerbado, em conjunto com outros efeitos, para reduzir características distinguíveis das placas e simular apagamento/envelhecimento, enquanto em Soufi e Valdenegro-Toro (2019) as variações são feitas com finalidade de criar variabilidade, apenas.

Muitos dos métodos do estado da arte enfatizam em transformações de cores para realizar *data augmentation*. Em Ashiquzzaman, Tushar e Rahman (2017), além de adição de ruído, rotações e translações a imagens, elas também passam por um processo chamado de embranquecimento ZCA. Krizhevsky, Sutskever e Hinton (2012) aprimoram classificação de CNN ao realizarem *data augmentation* baseado na alteração de intensidade de canais RGB em imagens de treinamento. De forma semelhante, no trabalho de Wu et al. (2015), essa transformação é feita gerando inteiros aleatórios entre -20 e 20 e aplicando-os a canais RGB – anteriormente a este processo, há uma geração de valores booleanos que determina quais canais são modificados.

Percebe-se que muitos dos métodos aplicados no estado da arte (como transformações de cores e adição de ruído e borrão) se assemelham aos considerados no presente estudo. Entretanto, nenhum deles os aplica especificamente de forma a simular as condições das placas pertencentes ao BRTSD, conforme proposto e explicado na Seção 3.6.3.

Capítulo Quatro

Resultados

4.1 *Dataset*

O *dataset* gerado durante a pesquisa contém 7805 imagens com 17493 placas anotadas, todas rotuladas com o tipo e condição.

As condições das placas encontradas estão na Tabela 4.1 juntamente com a quantidade de placas encontrada em cada uma. É possível observar que a maioria das placas se encontram em bom estado e o problema que mais afeta a qualidade das placas é a obstrução por vegetação.

Tabela 4.1: Condições das placas do *dataset* desenvolvido

Condição	**Quantidade**
Boa	16506
Coberta por vegetação	496
Apagada	357
Outras condições (Ferrugem, pichadas sujas ou com adesivo)	134
Total	17493

Obteve-se trinta e cinco tipos de placa, distribuídos conforme Figura 4.1. As classes não apresentam distribuição uniforme devido a sinalizações específicas, como proibido estacionar e limite de velocidade de 50km/h, serem encontradas com uma frequência bem maior nas imagens analisadas. Em contrapartida, placas de trânsito como rotatória à frente e semáforos à frente são relativamente escassos. No histograma também são apresentadas as classes 'vegetacao' e 'apagada', referentes a placas não identificáveis nessas condições. Além disso, há uma classe chamada outros tipos, com placas que não tinham uma classe específica no *dataset*. O objetivo dessa classe é evitar a presença de objetos não anotados e permitir que a rede detecte os mais variados tipos de placas.

Vale ressaltar que os resultados apresentados a seguir não necessariamente dizem respeito às características do *dataset* apresentadas na Tabela 4.1 e Figura 4.1, as quais apresentam a última atualização do *dataset*, que está em constante evolução. Os modelos foram treinados em versões anteriores contendo menos anotações, mas essa diferença não é expressiva o suficiente para afetar as discussões apresentadas.

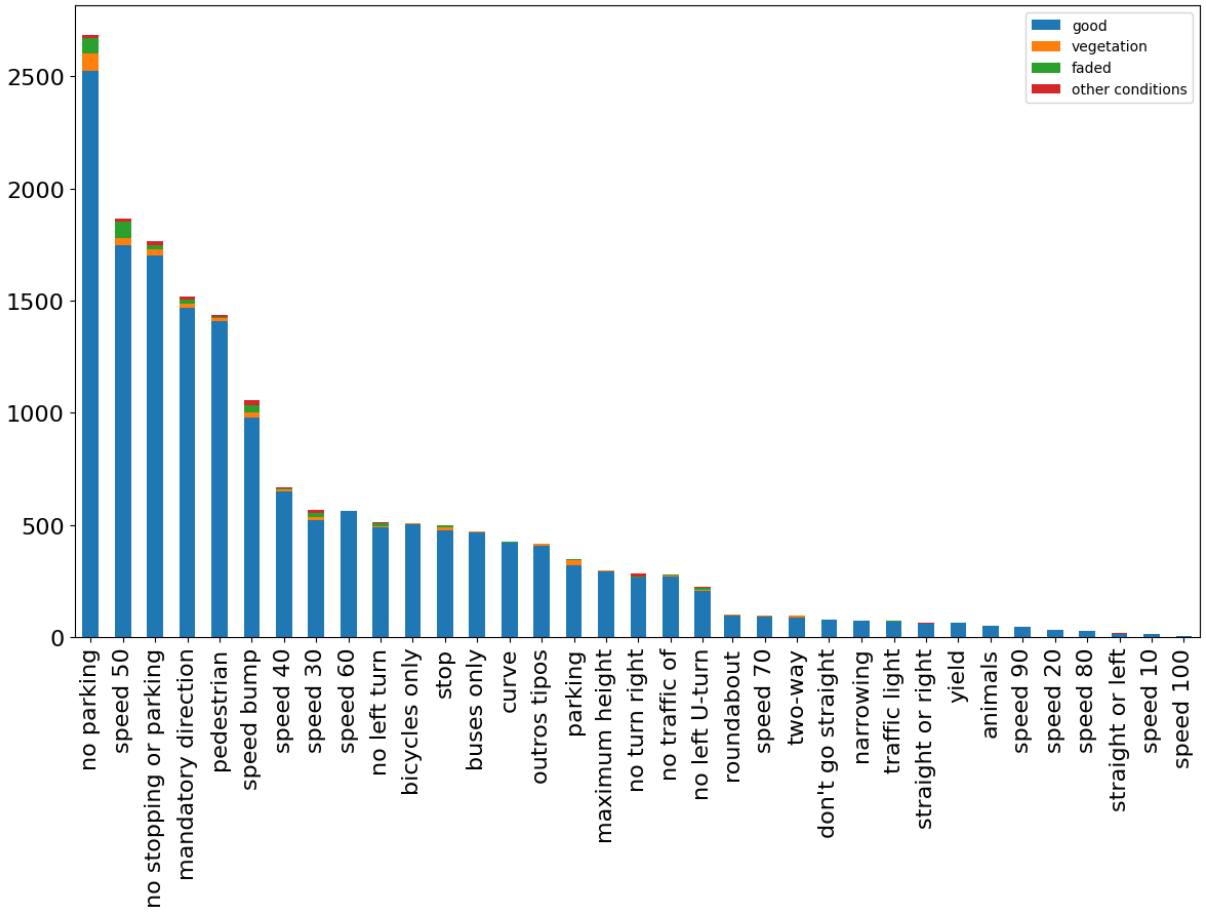


Figura 4.1: Distribuição dos tipos de placas no dataset.
Fonte: Autoria própria.

4.2 Detecção de objetos com yolov5

4.2.1 Placas obstruídas por vegetação

A Tabela 4.2 apresenta as métricas das arquiteturas do yolov5 para detecção de placas cobertas por vegetação. O *support* pode ser diferente entre os modelos porque aplicou-se validação cruzada (K-fold) e escolheu-se os pesos com melhor AP para esta classe dentre os cinco *folds* utilizados.

Observa-se que mesmo o modelo mais simples, yolov5s, consegue detectar e classificar 42 placas corretamente de um total de 90. Provavelmente devido à dificuldade em detectar esse tipo de objeto, a ocorrência de FNs e baixa confiança das detecções (representada no AP) são consideráveis e necessitam de melhorias. O primeiro passo seria aumentar a quantidade de anotações para esta classe que, de acordo com a Tabela 4.1, apresenta 496 anotações. De acordo com Timofte, Zimmermann e Gool (2014), outra possibilidade seria adicionar imagens sem objetos de interesse.

A má qualidade da sinalização de trânsito pode trazer consequências desastrosas e mesmo a detecção de uma única placa em condição de obstrução por vegetação ou apagada pode fazer toda a diferença, salvando vidas e evitando prejuízos econômicos.

O modelo yolov5m seria uma melhor escolha pois apresenta métricas com valores próximos aos máximos e tem uma velocidade de detecção intermediária.

Tabela 4.2: Resultados do YOLOv5 para a classe 'Vegetação'

	precision	**recall**	**f1-score**	**AP**	**support**
yolov5n	0.689	0.467	0.556	0.248	90
yolov5s	0.804	0.500	0.616	0.300	90
yolov5m	0.766	0.584	0.663	0.346	101
yolov5l	0.776	0.578	0.662	0.365	90
yolov5x	0.782	0.604	0.682	0.374	101

A Figura 4.2 mostra algumas das detecções de placas cobertas por vegetação realizadas na validação do modelo yolov5m. É possível perceber que o modelo consegue detectar situações difíceis como as ilustradas nas Figuras 4.2 c) e d), bem como obstruções em menor nível como nas Figuras 4.2 a) e e). A Figura 4.2 f) é representativa dos falsos positivos que ocorreram para esta classe: a vegetação se encontra próxima à placa (atrás neste caso), mas sem obstruí-la de fato.

A Figura 4.3 mostra uma detecção de placa obstruída por vegetação que não havia sido previamente anotada. A imagem tem zoom aplicado e observe que mesmo assim é difícil para um humano identificar e anotar esse objeto. Por isso o processo de validação cruzada nesse processo de construção do *dataset* é importante. As placas podem ter uma área relativamente pequena em comparação com a área total da imagem e algumas podem passar despercebidas pela equipe de

Figura 4.2: Detecções de placas com vegetação.
Fonte: Autoria própria.

anotação, mas não pelo modelo que está sendo desenvolvido. Na etapa de avaliação (Figura 3.1) dos modelos, o *FiftyOne* registra esses casos como falsos positivos e assim a equipe pode fazer uma revisão no sentido de incluir esses novos objetos no *dataset*. Após essa revisão, a avaliação do modelo pode ser refeita para verificar o impacto nas métricas. Com a correção desse caso da Figura 4.3, por exemplo, seria somada uma unidade na contagem de TPs e reduzida uma unidade na contagem de FPs, melhorando as métricas analisadas de uma maneira geral.

Figura 4.3: Falso positivo na classe de vegetação.
Fonte: Autoria própria.

4.2.2 Placas apagadas

Os resultados obtidos para detecção de placas apagadas são apresentados na Tabela 4.3. Da mesma forma como ocorre nos resultados para placas obstruídas por vegetação discutidos acima, tem-se um baixo *recall* nos modelos, porém a precisão é pelo menos 70%.

A discussão aqui é semelhante no que diz respeito à segurança no trânsito: uma única placa ruim detectada pode fazer toda a diferença. Aumentar o número de amostras no *dataset* pode trazer melhorias e o modelo yolov5m é uma boa escolha na relação métrica-velocidade.

Tabela 4.3: Resultados do YOLOv5 para a classe apagada

	precision	**recall**	**f1-score**	**AP**	**support**
yolov5n	0.704	0.317	0.437	0.222	60
yolov5s	0.778	0.368	0.500	0.267	38
yolov5m	0.895	0.447	0.596	0.356	38
yolov5l	0.967	0.446	0.611	0.332	65
yolov5x	0.952	0.526	0.678	0.362	38

A Figura 4.4 mostra algumas das detecções de placas apagadas realizadas na validação do modelo yolov5m. Note que o modelo consegue detectar placas bem apagadas como as ilustradas nas Figuras 4.4 d) e f), bem como apagamentos em menor nível como na Figura 4.4 c). Observe também que o falso positivo da Figura 4.4 b) guarda semelhanças com a situação encontrada na Figura 4.4 f).

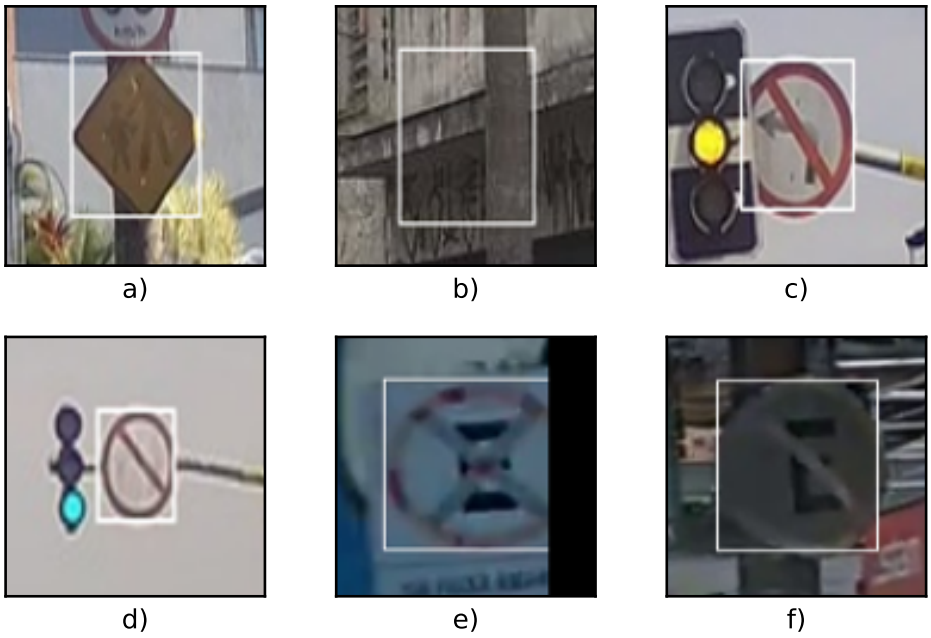


Figura 4.4: Detecções de placas apagadas.
Fonte: Autoria própria.

4.2.3 Todas as classes

A Tabela 4.4 apresenta as médias de *precision*, *recall* e f1-score para cada arquitetura do yolov5 quando se observa todas as classes. Também é apresentado o mAP.

Levando em conta a relação métricas-velocidade, nota-se que boas escolhas seriam o yolov5s ou yolov5m, pois suas métricas se encontram próximas as ao yolov5x, que é o modelo com mais parâmetros e mais lento.

Tabela 4.4: Resultados médios do YOLOv5 para todas as classes

	precision	***recall***	***f1-score***	**mAP**	***support***
yolov5n	0.754	0.684	0.717	0.386	2995
yolov5s	0.799	0.796	0.798	0.520	2959
yolov5m	0.802	0.824	0.813	0.592	3104
yolov5l	0.807	0.825	0.816	0.587	3104
yolov5x	0.811	0.834	0.822	0.602	3104

A Tabela 4.5 apresenta *precision*, *recall* e f1-score por classe para a arquitetura yolov5m. A tabela está ordenada pelo *support*. Observe que não existe uma relação bem definida entre quantidade de amostras treinadas e métrica, embora um *support* pequeno esteja mais suscetível a boas métricas. Esses *supports* pequenos occorrem porque os *folds* do *K-fold* são escolhidos de maneira aleatória, o *dataset* é desbalanceado e as imagens têm diferentes quantidades e tipos de placas, o que pode até mesmo anular o support para uma classe específica. Nesse sentido, melhorias podem ser alcançadas ao balancear o *dataset*.

De fato, as classes que representam as más condições de placas ('vegetação' e 'apagada') estão entre as classes com f1-score mais baixo pois representam um desafio maior em TSR, conforme discutido anteriormente.

É difícil fazer uma comparação justa entre os modelos obtidos neste trabalho e os apresentados em outros trabalhos, uma vez que um novo *dataset* está sendo proposto, com diferentes classes, resolução de imagem, distribuições de classes, e assim por diante. Recentemente, Mannan et al. (2022) apresentaram um estudo semelhante envolvendo oclusão em TSR, mas usando seu próprio conjunto de dados. Os autores obtiveram métricas de *precision* e *recall* de 0,81 e 0,79, respectivamente. Na Tabela 4.5, o modelo yolov5m proposto, por exemplo, tem precisão de 0,92 e revocação de 0,85. No entanto, o conjunto de dados descrito em Mannan et al. (2022) pode ter mais instâncias com situações desafiadoras.

Adicionalmente, Mannan et al. (2022), são apresentados resultados de *accuracy* de 0,8 a 0,83 na classificação de placas de trânsito obstruídas por vegetação. No entanto, a *accuracy* depende da computação TNs, que não é observada no contexto da arquitetura do yolov5.

Analisando a métrica mAP referente à tarefa TSR em geral, em Gu e Si (2022), a aplicação de

Tabela 4.5: Resultados do YOLOv5 para todas as classes

	precision	*recall*	*f1-score*	*support*	*training count*
proibido estacionar	0.754	0.855	0.801	462	1846
proibido parar e estacionar	0.877	0.863	0.870	315	1167
velocidade 50	0.787	0.818	0.802	285	1211
sentido obrigatorio	0.741	0.796	0.768	274	1037
pedestres	0.776	0.817	0.796	229	850
lombada	0.865	0.906	0.885	191	721
velocidade 30	0.770	0.784	0.777	111	378
velocidade 40	0.703	0.743	0.722	105	430
proibido virar a esquerda	0.697	0.745	0.720	102	330
vegetacao	0.610	0.465	0.528	101	356
pare	0.847	0.865	0.856	96	360
exclusivo ciclistas	0.755	0.822	0.787	90	357
exclusivo onibus	0.748	0.933	0.830	89	321
estacionamento	0.716	0.727	0.722	66	244
altura maxima	0.746	0.820	0.781	61	175
proibido circulacao	0.839	0.797	0.817	59	192
velocidade 60	0.698	0.772	0.733	57	287
curva	0.726	0.789	0.756	57	214
outros tipos	0.620	0.554	0.585	56	276
proibido virar a direita	0.569	0.617	0.592	47	176
apagada	0.500	0.277	0.356	47	210
proibido retornar a esquerda	0.725	0.659	0.690	44	143
duplo sentido	0.842	0.762	0.800	21	67
estreitamento	0.714	0.556	0.625	18	44
sentido proibido	0.625	0.588	0.606	17	51
velocidade 70	0.615	0.571	0.593	14	53
semaforo	0.700	0.583	0.636	12	41
rotatoria	0.909	0.833	0.870	12	46
preferencia	0.857	0.750	0.800	8	47
siga em frente ou a direita	1.000	1.000	1.000	6	33
animais	1.000	0.500	0.667	6	30
velocidade 20	0.833	1.000	0.909	5	22
velocidade 90	0.667	0.800	0.727	5	25
velocidade 80	1.000	0.750	0.857	4	18
micro avg	0.764	0.791	0.777	3072	11758

Faster R-CNN em GTSDB atinge um mAP de 0,911, enquanto o modelo yolov5, na Tabela 4.4, atinge no máximo 0,602. No entanto, o conjunto de dados GTSDB não possui placas obstruídas por vegetação ou apagadas, portanto, o respectivo modelo *Faster R-CNN* não é testado nessas situações, resultando em um mAP melhor. Outro modelo *Faster R-CNN*, treinado em ITSDB (ZHOU; ZHAN; FU, 2021) e que considera oclusão e cenários noturnos, apresenta AP de 0,456. Portanto, fica evidente que, quanto mais realista e desafiador for o conjunto de dados, pior tende a ser o desempenho dos modelos treinados. Nesse sentido, o modelo desenvolvido se encontra diante do esperado.

A Figura 4.5 apresenta a matriz de confusão referente à Tabela 4.5. É notável que o modelo detector não confunde as classes das placas de maneira significativa. Entretanto, o modelo não é perfeito e está fadado a eventualmente errar, deixando de detecta r algumas placas. Por exemplo, para a classe proibido estacionar, 46 de 462 (isto é, a soma dos valores na horizontal da linha proibido_estacionar da matriz) placas não são detectadas (menos de 10% do total). Novamente, o aumento da quantidade de amostras do *dataset* pode trazer melhorias para esta questão, contribuindo à medida de desempenho *recall*.

4.3 Classificação com CNN Multilabel

Esta seção apresenta os resultados do treinamento da CNN multilabel, juntamente com sua avaliação e as métricas resultantes, que são posteriormente analisadas.

4.3.1 Resultados de treinamento

Na Figura 4.6, são ilustrados os resultados do treinamento da CNN multilabel ao longo de 100 épocas, tanto para os subconjuntos de treinamento, quanto de validação, juntamente com os pontos correspondentes de valores mínimos. A curva de treinamento demonstra que apenas algumas épocas são necessárias para atingir níveis de convergência para a métrica de Hamming loss. Apesar do limite de épocas estar definido em 100, o conjunto ótimo de pesos foi obtido na 37ª época, onde o Hamming loss ficou em 0.081, como evidenciado pela curva de validação no gráfico.

4.3.2 Resultados da validação

As métricas de validação para as *labels* (rótulos) tipo e condição da CNN multilabel são apresentados na Tabela 4.6, demonstrando a considerável robustez desta abordagem. O modelo treinada obtém valores ideais para todas as métricas em 10 das classes de tipo. Exceto pelo tipo *vegetation* (vegetação), que indica oclusão por vegetação completa, o f1-score mínimo é de 0.849.

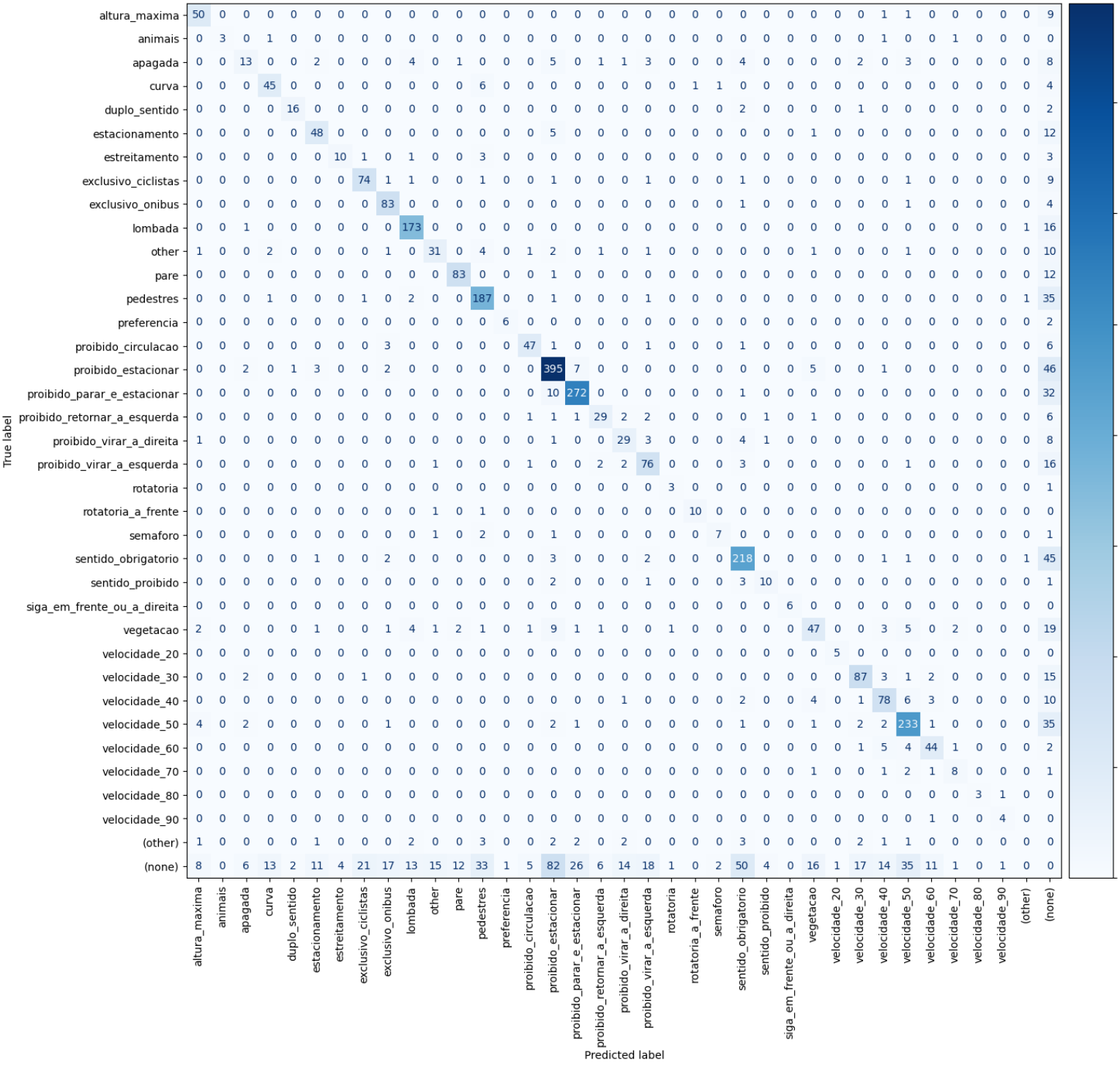


Figura 4.5: Matriz de confusão para todas as classes
Fonte: Autoria própria.

Tabela 4.6: Resultados para a *label* de tipo

	precision	recall	f1-score	support
altura maxima	1.000	1.000	1.000	18.0
animais	1.000	1.000	1.000	19.0
apagada	1.000	0.750	0.857	20.0
curva	0.889	1.000	0.941	16.0
duplo sentido	1.000	0.917	0.957	24.0
estacionamento	1.000	0.794	0.885	34.0
estreitamento	0.947	0.947	0.947	19.0
exclusivo ciclistas	0.889	0.941	0.914	17.0
exclusivo onibus	1.000	1.000	1.000	18.0
lombada	0.950	0.950	0.950	40.0
pare	1.000	0.852	0.920	27.0
pedestres	0.952	0.870	0.909	23.0
preferencia	1.000	1.000	1.000	19.0
proibido circulacao	0.938	0.882	0.909	17.0
proibido estacionar	0.953	0.953	0.953	85.0
proibido parar e estacionar	0.947	0.973	0.960	37.0
proibido retornar a esquerda	1.000	0.944	0.971	18.0
proibido virar a direita	1.000	1.000	1.000	14.0
proibido virar a esquerda	1.000	1.000	1.000	26.0
rotatoria	1.000	1.000	1.000	28.0
semaforo	0.947	0.947	0.947	19.0
sentido obrigatorio	0.933	0.933	0.933	30.0
sentido proibido	1.000	1.000	1.000	19.0
siga em frente ou a direita	0.941	1.000	0.970	16.0
siga em frente ou a esquerda	1.000	1.000	1.000	4.0
vegetacao	0.533	0.857	0.658	28.0
velocidade 20	1.000	1.000	1.000	8.0
velocidade 30	0.969	0.939	0.954	33.0
velocidade 40	1.000	0.955	0.977	22.0
velocidade 50	0.942	0.985	0.963	66.0
velocidade 60	0.905	1.000	0.950	19.0
velocidade 70	1.000	1.000	1.000	17.0
velocidade 80	1.000	0.778	0.875	9.0
velocidade 90	1.000	0.933	0.966	15.0
accuracy	0.943	0.943	0.943	0.0
macro avg	0.960	0.944	0.949	824.0
weighted avg	0.953	0.943	0.945	824.0

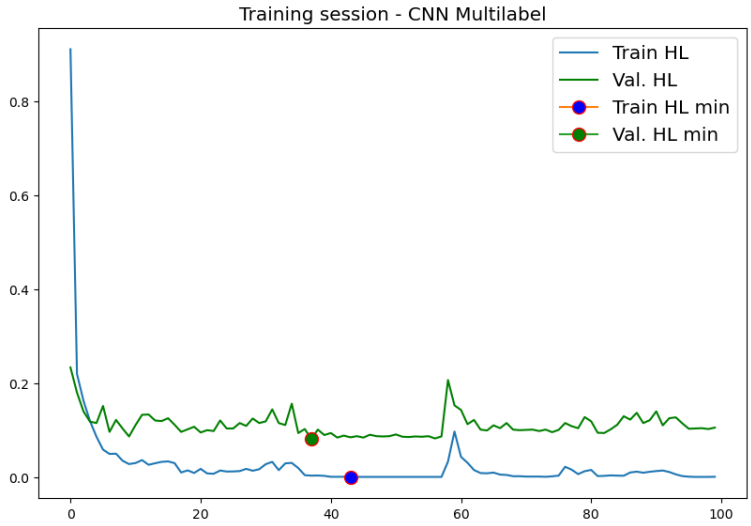


Figura 4.6: Hamming loss ao longo de 100 épocas de treinamento para a CNN multilabel
Fonte: Autoria própria.

A Tabela 4.7 fornece as métricas de validação para a *label* de condição. O f1-score mínimo de 0,849 em todas as condições mostra que o modelo também pode classificar efetivamente a condição de uma placa junto com o tipo (conforme mostrado na Tabela 4.6). As condições *apagada* e *vegetação* tiveram *recall* menor do que a precisão, o que não é necessariamente ruim. Isso pode simplesmente significar que o modelo prioriza a precisão em relação ao reconhecimento de todas as instâncias. Um alto *recall* com baixa precisão indicaria um modelo potencialmente menos confiável nesse contexto, exigindo verificação humana para os fins de notificação de irregularidades.

Embora não explicitamente indicado nas Tabelas 4.6 e 4.7, considerando que a CNN multilabel possui duas camadas totalmente conectadas independentes para a classificação de cada atributo da placa, mesmo quando ela classifica erroneamente o tipo, ela ainda pode rotular corretamente a condição do objeto, e vice-versa. Portanto, desde que a placa seja consideravelmente reconhecível, a arquitetura de modelo proposta muito provavelmente a dará um rótulo apropriado para pelo menos um de seus atributos.

Tabela 4.7: Resultados para a *label* de condição

	precision	recall	f1-score	support
apagada	0.984	0.759	0.857	162.0
boa	0.856	0.994	0.920	486.0
vegetacao	0.978	0.750	0.849	176.0
accuracy	0.896	0.896	0.896	0.0
macro avg	0.939	0.834	0.875	824.0
weighted avg	0.907	0.896	0.892	824.0

Dado que cada amostra do conjunto de dados possui dois atributos associados (tipo e condição), é necessário gerar duas matrizes de confusão distintas. A Figura 4.7 mostra como essa abordagem permitiu que o modelo acertasse a maioria das classes diferentes na classificação de tipo. Muitos dos falsos positivos estão entre as classes de velocidade, já que são muito semelhantes entre si e

diferem apenas nos números escritos no centro da placa de trânsito.

A Figura 4.8 mostra a matriz de confusão resultante para as classes de condição no conjunto de dados. Mesmo que a quantidade de falsos positivos para a classe *good* seja comparável à quantidade de predições corretas para as classes *faded* e *vegetation*, isso se deve ao fato de que as condições não estão tão balanceadas no *dataset*. Entretanto, a proporção de verdadeiros positivos para falsos positivos dentro de cada classe é consideravelmente alta, o que é refletido nas métricas de validação a serem apresentadas mais adiante.

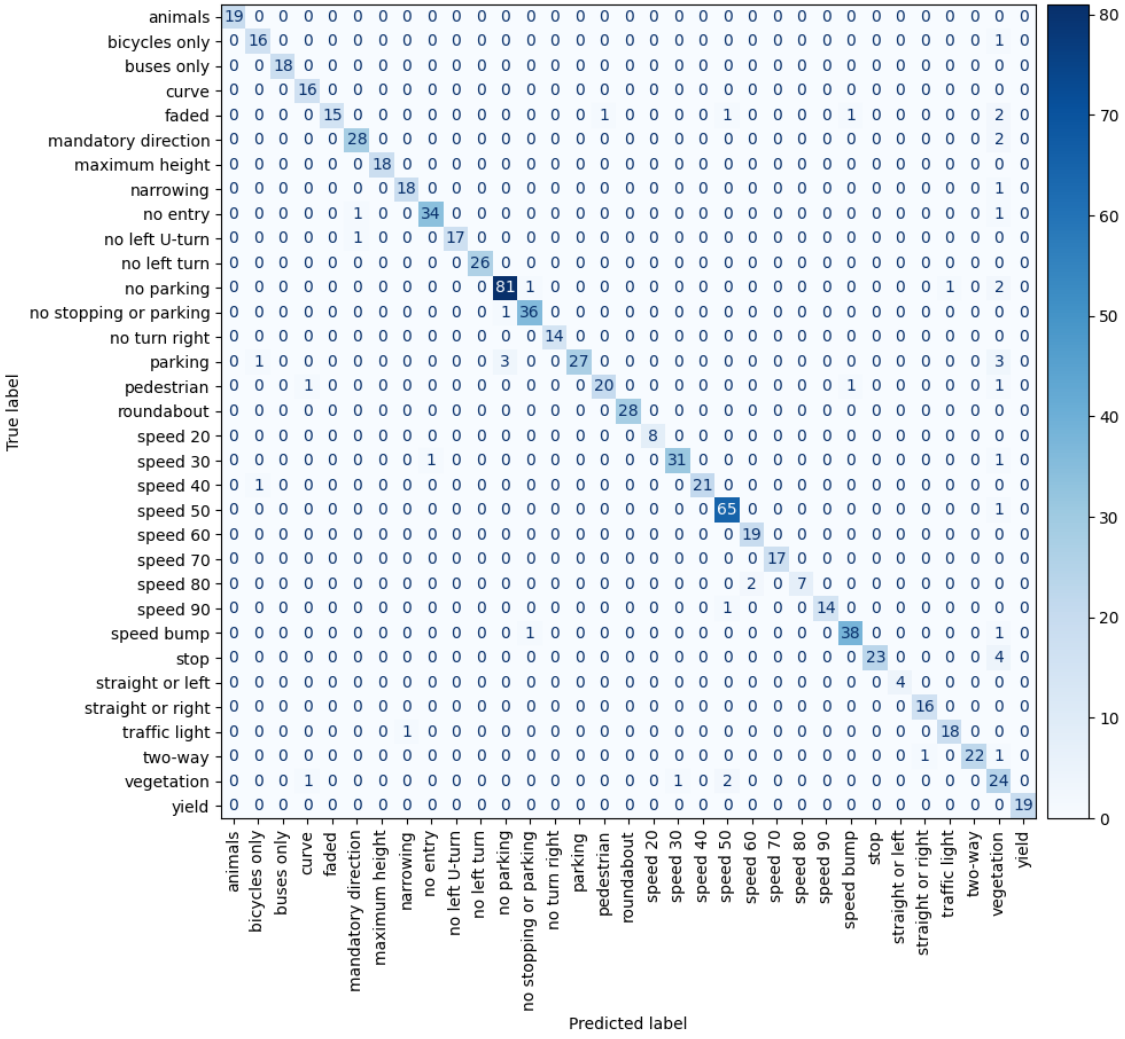


Figura 4.7: Matriz de confusão para os tipos de placa

Fonte: Autoria própria.

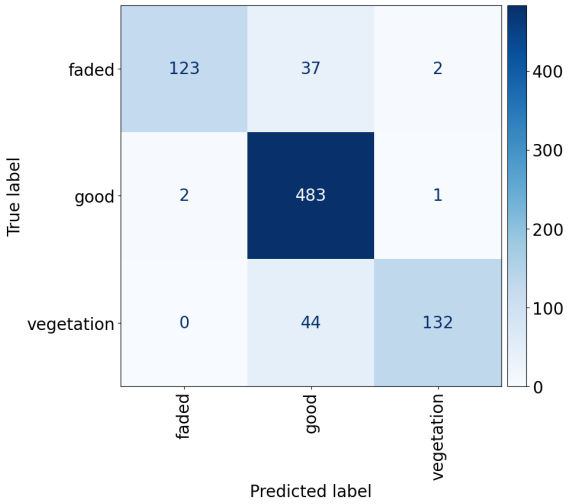


Figura 4.8: Matriz de confusão para as condições de placa

4.4 *Data augmentation* para CNN Multilabel

Nesta seção de resultados, apresentamos os resultados obtidos com diferentes métodos de *data augmentation* desenvolvidos com o objetivo para aumentar o conjunto de dados de treinamento do modelo de classificação de placas de trânsito. Métricas para o *dataset* original são reveladas e, em sequência, utilizadas como referência para os resultados de cada método estudado. É importante ressaltar que estes resultados foram obtidos com uma versão do *dataset* anterior àquela do *dataset* aplicado na Seção 4.3. Sendo assim, as métricas utilizadas como referência durante comparações entre o *dataset* original e os modificados também irão diferir das já exibidas. Este consiste, portanto, num estudo à parte, visando exclusivamente analisar os possíveis impactos da aplicação de *data augmentation* num *dataset* de classificação. Adicionalmente, esta seção considera no *dataset* a inclusão de poucas instâncias de placas pichadas que, embora sejam insuficientes para a tarefa de classificação, são utilizadas em tentativas de criar métodos eficientes de geração artificial de pichações em placas reais. Por último, o grau de evolução do *dataset* que é aplicado na Seção 4.3 impede que as instâncias artificiais obtidas na atual seção nele sejam inclusas.

4.4.1 Métricas para o conjunto de dados original

Foram executados 5 treinamentos sequenciais com o *dataset* original. A Tabela 4.8 exibe a média das métricas obtidas nesta sessão. Esta é referenciada em toda Seção 4.4 como meio de comparação entre os *datasets* ampliados com os métodos descritos e o *dataset* original (a partir de tabelas geradas com o Algoritmo 13 do Apêndice C). Cada linha da tabela refere-se a uma condição específica. A linha para condição '*Vegetação*', por exemplo, exibe cálculos de *recall* do tipo (*tipo_ acerto*), condição (*condição_ acerto*) e ambas as *labels* (*todos_ acerto*) para o grupo de placas ocultas por vegetação no subconjunto de validação. Adicionalmente, a matriz de confusão

Tabela 4.8: Média de *recall* obtida com 5 treinamentos segundo o *dataset* original

	Média recall obtida com 5 treinamentos - original dataset		
Condição	**tipo_acerto**	**condição_acerto**	**todos_acerto**
Todos	0.884	0.825	0.747
Vegetação	0.839	0.782	0.649
Apagamento	0.816	0.675	0.577

na Figura 4.9 permite visualizar a precisão relativa do modelo entre todas as condições de placa estudadas.

Adicionalmente, a Figura 4.10 exibe a variação das métricas *train_mean_acc* (média geral entre *type_match* e *condition_match* para dados de treino) e *val_mean_acc* (média para dados de validação) no decorrer do treinamento do melhor modelo treinado com o *dataset* original. Neste contexto, entre todos os modelos treinados, foi considerado melhor aquele que obteve maior média de *recall* para a condição das placas, sobre a qual os métodos de *data augmentation* são aplicados. Entretanto, o critério de seleção do melhor conjunto de pesos durante cada treinamento continuou sendo o *val_mean_acc*.

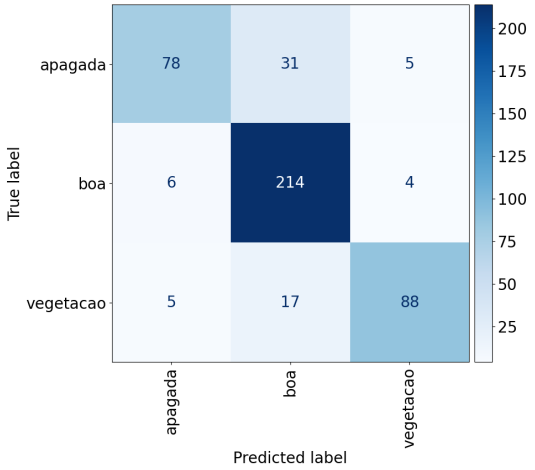


Figura 4.9: Matriz de confusão para *dataset* original: rede a possuir maior *recall* para todas condições (0.855)

Fonte: Autoria própria.

4.4.2 Método 1: Sobreposição de vegetação artificial

Utilizando uma rede avaliadora, foram selecionadas apenas 79 das cerca de 2400 placas artificialmente modificadas geradas pelo Método 1 (conforme Seção 3.6.3.1), segundo critério de confiança maior igual a 80% e ajustes finos. A Figura 4.11 exemplifica uma das placas oclusas artificialmente com vegetação a partir do Método 1. Apesar do pós-processamento aplicado, o tipo e a condição desta placa continuam reconhecíveis, o que é validado pela capacidade da rede avaliadora em detectá-lo com alta confiança.

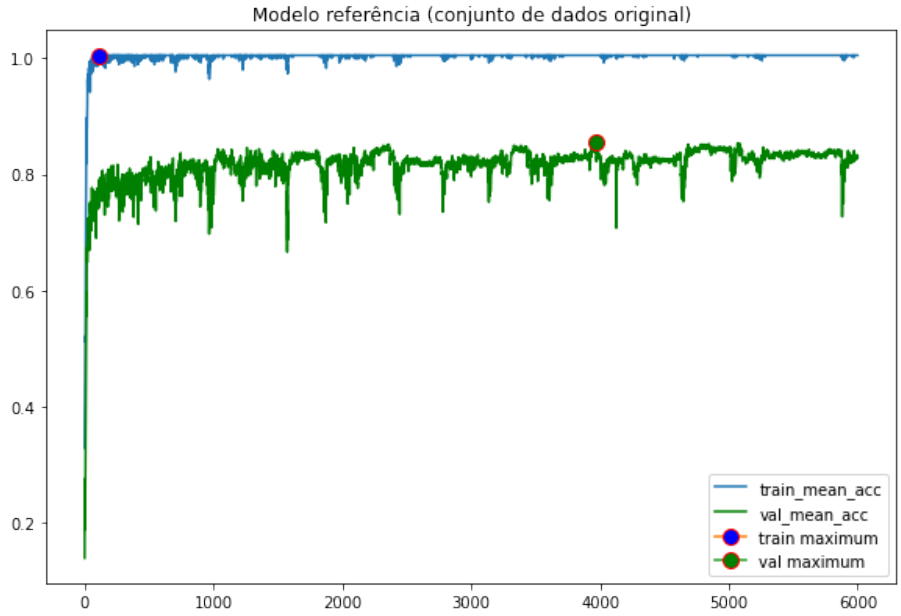


Figura 4.10: Relatório de treinamento para o melhor modelo obtido com o conjunto de dados original numa sessão de 5 treinamentos consecutivos

Fonte: Autoria própria.

A ausência do pós-processamento resulta em efeitos indesejados, como os presentes na instância exibida na Figura 4.12. Neste caso, percebe-se um serrilhamento e conjuntos de *pixels* incompatíveis com o fundo da imagem original, além de ser facilmente notável tratar-se de uma imagem sobreposta em outra, ao invés uma placa oculta por vegetação.

Com a aplicação do Método 1, foi possível gerar novas imagens de placas de trânsito com oclusões de vegetação, ampliando o conjunto de dados para o treinamento de modelos de classificação. A Figura 4.13 exibe a variação das métricas de treinamento e validação através de 6000 épocas. A Tabela 4.9 apresenta a média de *recall* para 5 redes treinadas com o conjunto de dados ampliados pelo Método 1. Adicionalmente, é exibido o valor referência (Ref.) para cada métrica, valores obtidos da Tabela 4.8.

Tabela 4.9: Comparação do resultado da média de *recall* sobre 5 treinamentos entre dataset aumentado via Método 1 e dataset original (Ref.)

	Média recall obtida com 5 treinamentos - Método 1					
	tipo_acerto		**condição_acerto**		**todos_acerto**	
Condição	**Método 1**	**Ref.**	**Método 1**	**Ref.**	**Método 1**	**Ref.**
Todos	0.897	0.884	0.821	0.825	0.748	0.747
Vegetação	0.833	0.839	0.782	0.782	0.649	0.649
Apagamento	0.860	0.816	0.649	0.675	0.568	0.577

Observa-se que o objetivo de aumentar o *recall* na classificação da condição de placas com vegetação não é atingida, visto que o valor se mantém igual em comparação ao *dataset* original: 0.782. Por outro lado, este método afeta o *recall* de placas apagadas, variando positivamente

Figura 4.11: Exemplo de placa artificialmente oculta por vegetação a partir do método descrito na Seção 3.6.3.1

Fonte: Autoria própria.

Figura 4.12: Exemplo de instância artificial sem pós-processamento, o que a faz apresentar características não naturais e indesejáveis para os propósitos do método descrito na Seção 3.6.3.1

Fonte: Autoria própria.

(+5.39%) o *tipo_acerto* delas e causando uma leve queda (-3.85%) no *recall* para a condição. Acredita-se que a inclusão de mais instâncias de placas ocultas por vegetação ajude a distinguir placas com outras condições. De modo geral, a inclusão desta *batch* de placas artificialmente ocultas por vegetação não parece alterar os resultados consideravelmente, e novas variações neste método devem ser testadas no futuro. A matriz de confusão na Figura 4.14 permite visualizar a precisão relativa do modelo entre todas as condições.

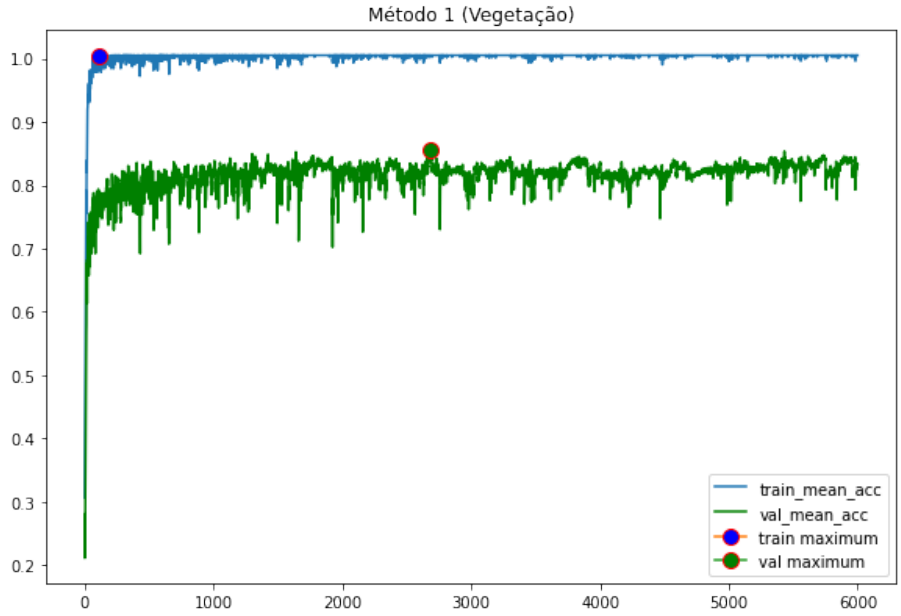


Figura 4.13: Relatório de treinamento para o melhor modelo obtido com o Método 1 numa sessão de 5 treinamentos consecutivos

Fonte: Autoria própria.

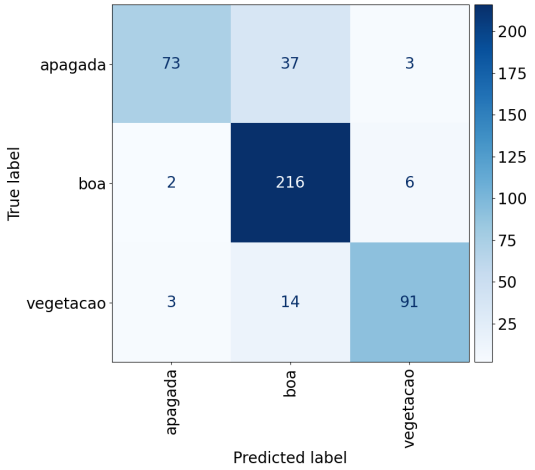


Figura 4.14: Matriz de confusão para *dataset* do Método 1 (Seção 4.4.2): rede o possuir maior *recall* para a condição de oclusão por vegetação

Fonte: Autoria própria.

4.4.3 Método 2: Reuso de instâncias com vegetação

Seguindo a metodologia na Seção 3.6.3.2, foram selecionadas 125 novas placas ocultas por vegetação para o treinamento de novas redes. O critério principal utilizado foi a confiança mínima de 95% da rede avaliadora ao classificar o tipo e a condição das novas instâncias criadas.

Um exemplo é exibido na Figura 4.15. Neste, é perceptível que, entre os efeitos aplicados

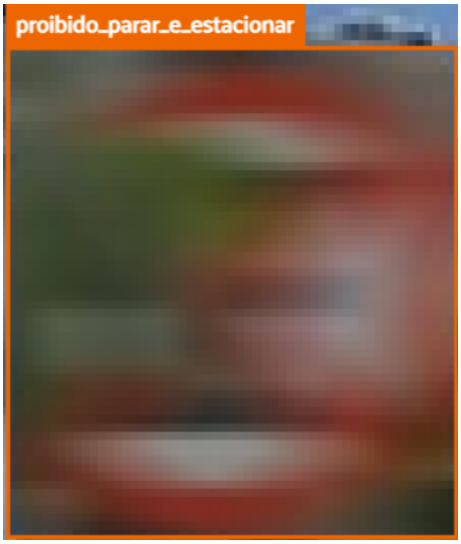


Figura 4.15: Exemplo de placa oculta por vegetação obtida pelo método descrito na Seção 3.6.3.2
Fonte: Autoria própria.

aleatoriamente, foi utilizado o borrão de movimento. Essas variações em relação à placa original cria diversidade no conjunto de dados, tornando a rede mais robusta e preparada para detectar novos casos de oclusão por vegetação, ainda que em circunstâncias desafiadoras. Além disso, nota-se uma variação entre o fundo da placa e o fundo fora das dimensões do *bounding box*. Esta discrepância não afeta o treinamento da rede multi-classificação, visto que apenas o recorte do *bounding box* é aproveitado.

A utilização das amostras resultantes num treinamento de um detector, por outro lado, pode gerar resultados não favoráveis. Figura 4.16 exibe a variação das métricas de treinamento e validação através de 6000 épocas para a rede com maior *recall* para a condição de vegetação.

A Tabela 4.10 exibe a média dos recalls obtidas segundo o Método 2. Percebe-se um aumento em quase todas as métricas, com exceção de *condição_acerto* e *todos_acerto* para a condição de placas apagadas. O objetivo de aumentar o *recall* para placas ocultas por vegetação é atingido, com um significativo aumento de 7.03% na classificação da condição. Englobando-se todas as condições, tem-se variações de +1.24% (*tipo_acerto*), +0.72% (*condição_acerto*) e +1.87% (*todos_acerto*). Apesar de não tão significativas, demonstram que o método causa melhorias gerais no modelo, e não só para as instâncias ocultas por vegetação. Assim, percebe-se que este é um método com potencial de melhorar os resultados de qualquer classe, dado que as mesmas operações podem ser aplicadas para outros casos.

As mesmas observações feitas segundo comparação de tabelas podem ser obtidas, semelhantemente, ao comparar a matriz de confusão resultante na Figura 4.17 com a da Figura 4.9 (original), onde tem-se 10 predições corretas a mais para a condição 'apagada' segundo o Método 2, o que é refletido em uma coloração levemente mais escura respectiva na matriz. Há 10 predições corretas a menos para a classe apagada, mas compensada menos 9 falsos positivos, por outro lado.

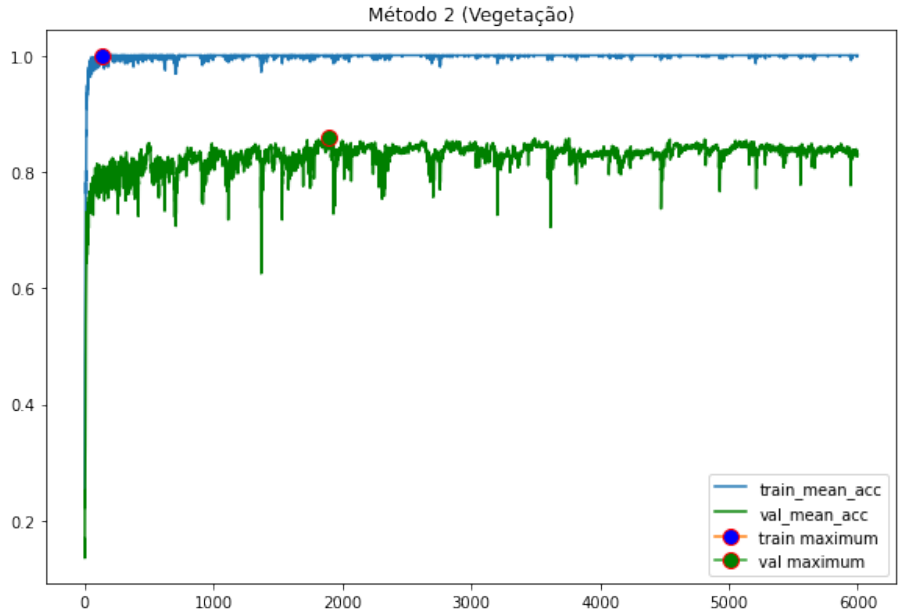


Figura 4.16: Relatório de treinamento para o melhor modelo obtido com o Método 2 numa sessão de 5 treinamentos consecutivos

Fonte: Autoria própria.

Tabela 4.10: Comparação do resultado da média de *recall* sobre 5 treinamentos entre *dataset* aumentado via Método 2 e *dataset* original (Ref.)

	Média recall obtida com 5 treinamentos - Método 2					
	tipo_acerto		**condição_acerto**		**todos_acerto**	
Condição	**Método 2**	**Ref.**	**Método 2**	**Ref.**	**Método 2**	**Ref.**
Todos	0.895	0.884	0.831	0.825	0.761	0.747
Vegetação	0.840	0.839	0.837	0.782	0.709	0.649
Apagamento	0.833	0.816	0.649	0.675	0.567	0.577

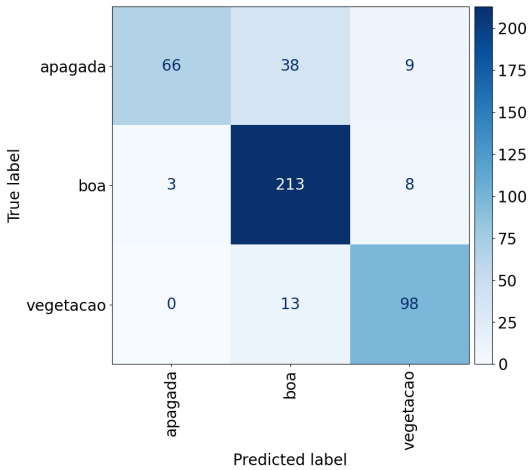


Figura 4.17: Matriz de confusão para *dataset* do Método 2 (Seção 4.4.3): rede a possuir maior *recall* para a condição de oclusão por vegetação

Fonte: Autoria própria.

4.4.4 *Método 3: Apagamento/envelhecimento artificial*

Com a aplicação do método descrito na Seção 3.6.3.3, resultou-se em 114 placas artificialmente apagadas selecionadas para o treino de uma novas redes. O critério principal consistiu em selecionar apenas placas artificiais cuja rede avaliadora tivesse designado confiança acima de 85% para o tipo e condição. Além disso, "*outliers*" (placas com padrões incomuns) foram descartadas para assegurar qualidade das instâncias selecionadas. O gráfico das métricas de treinamento e validação durante a sessão de treino para o melhor modelo treinado é exibido na Figura 4.19.

A Figura 4.18 exibe um exemplo de placa artificialmente apagada segundo o Método 3. É notável uma descoloração artificial e a aplicação borrões na região da placa, o que dificulta a identificação da mesma, mas no limite em que o modelo avaliador ainda possa corretamente classificar o seu tipo. É também perceptível que a máscara elíptica aplicada efetivamente limita a aplicação dos efeitos à região da placa, não afetando o espaço ao redor.

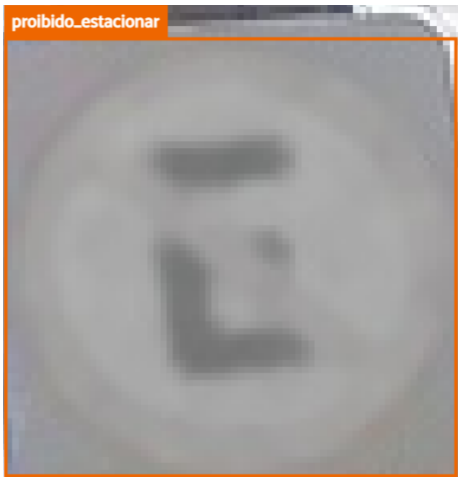


Figura 4.18: Exemplo de placa artificialmente apagada segundo Método descrito na Seção 3.6.3.3
Fonte: Autoria própria.

A Tabela 4.11 faz a comparação de médias de de *recall* entre o *dataset* do Método 3 e o original, permitindo mensurar o impacto da inclusão das amostras de placas artificialmente apagadas nos resultados. Dentre os métodos aplicados, este é o resultado mais expressivo, provavelmente pelo considerável realismo das novas instâncias geradas. Entre as placas apagadas, têm-se variações de $+6.86\%$ ($tipo\_acerto$), $+9.19\%$ ($condição\_acerto$) e $+16.12\%$ ($todos\_acerto$), ganhos expressivos e que deixam claro a efetividade do método. Além disso, todas as outras métricas têm algum ganho percentual com a inclusão das placas apagadas. Novamente, imagina-se que a inclusão destas placas artificiais ajudam na distinção das outras placas. Isto pode ser sustentado pela matriz de confusão na Figura 4.20, onde, comparativamente à matriz original (Figura 4.9), há 7 falsos positivos para a condições 'boa' e mais classificações corretas para a condição 'vegetação'.

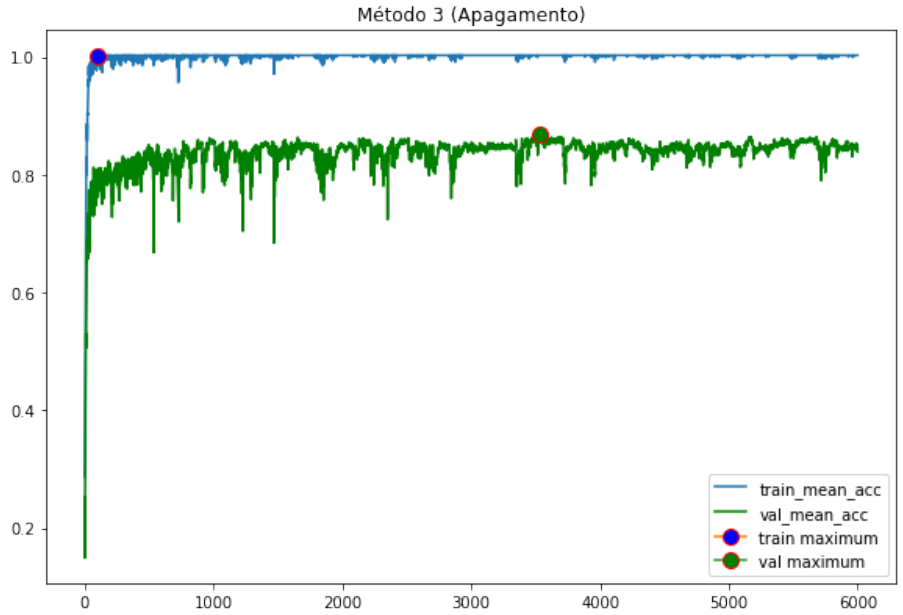


Figura 4.19: Relatório de treinamento para o melhor modelo obtido com o Método 3 numa sessão de 5 treinamentos consecutivos

Fonte: Autoria própria.

Tabela 4.11: Comparação do resultado da média de *recall* sobre 5 treinamentos entre *dataset* aumentado via Método 3 e *dataset* original (Ref.)

	Média recall obtida com 5 treinamentos - Método 3					
	tipo_acerto		**condição_acerto**		**todos_acerto**	
Condição	**Método 3**	**Ref.**	**Método 3**	**Ref.**	**Método 3**	**Ref.**
Todos	0.904	0.884	0.838	0.825	0.776	0.747
Vegetação	0.844	0.839	0.788	0.782	0.672	0.649
Apagamento	0.872	0.816	0.737	0.675	0.670	0.577

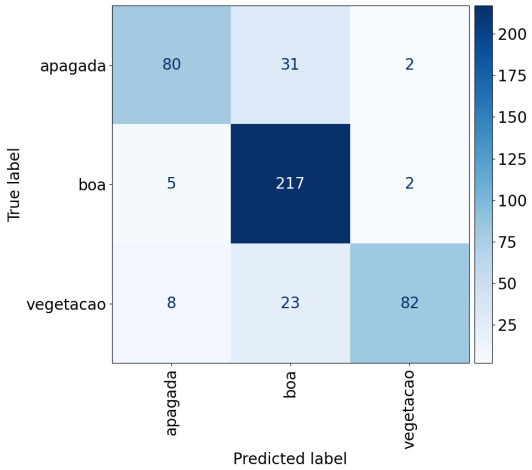


Figura 4.20: Matriz de confusão para *dataset* do Método 3 (Seção 4.4.4): rede a possuir maior *recall* para a condição de apagamento

Fonte: Autoria própria.

4.4.5 *Comparação com trabalhos correlatos e métodos alternativos explorados*

Em Zhang et al. (2020), não há ênfase em analisar o impacto da aplicação de *data augmentation* em classes individuais, como neste trabalho, mas na performance geral do modelo. Realizar uma comparação direta com tal pesquisa não é possível, visto que diferentes *datasets*, para diferentes propósitos, são utilizados. Além disso, são utilizados modelos diferentes: aqui utiliza-se uma CNN multilabel, enquanto em Zhang et al. (2020) é considerada a aplicação de um modelo R-CNN (detecção e classificação). Entretanto, a citação de um outro resultado, de outros autores, revela-se pertinente, uma vez que permite uma avaliação relativa do impacto e da relevância do presente estudo no contexto da área.

A metodologia de *data augmentation* conduzida por Zhang et al. (2020) leva a uma variação do *recall* de 85% para 90.5% (+5.5%) sobre uma mesma arquitetura, atingindo o objetivo almejado. Comparativamente, e conforme exibido na Tabela 4.11 da Seção 4.4.4, o Método 3 (para placas apagadas), que resultou nas maiores variações positivas produzidas entre os métodos de *data augmentation* criados e aplicados neste estudo, teve para todas as classes uma variação de *recall* de +2.9% no *todos_acerto*, +2% no *tipo_acerto* e +1.3% no *condição_acerto*. Conforme já discutido, entretanto, os métodos do presente trabalho enfatizaram na melhoria de métricas de classes específicas. Neste caso, conforme anteriormente discutido na Seção 4.4.4, o *todos_acerto* entre placas apagadas é variado em +16.1%.

No trabalho de Soufi e Valdenegro-Toro (2019), a aplicação de GAN na geração de novas instâncias tem sucesso ao aumentar a acurácia de classificação de 92.1% para 94.0% (+1.9%). Autores comparam este resultado a técnicas de aumento de dados tradicionais, como a baseada

em contraste, que entrega aumento de 3.4% em acurácia de classificação. Isto é, abordagens tradicionais provêm uma maior variação na acurácia do que a metodologia apresentada por Soufi e Valdenegro-Toro (2019). Este desfecho não invalida o método baseado em GAN, entretanto. Ashiquzzaman, Tushar e Rahman (2017), por outro lado, obtêm com procedimentos de aumento de dados baseados em variação de cores e aplicação de ruído variações de +5.4% e +2% em relação a outras duas abordagens usadas como referência.

Na abordagem mista de Krizhevsky, Sutskever e Hinton (2012), que incorpora um método de aumento de dados baseado em variações de intensidade RGB, translações, entre outros, validada no conjunto de dados ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) de 2010, os autores alcançaram uma taxa de erro top-5 de 17%. Isso representou variação de -8,7% em relação a uma abordagem que envolveu a média de dois classificadores treinados em *Fisher Vectors*.

Por fim, na abordagem de Wu et al. (2015), que empregou uma metodologia semelhante à utilizada por Krizhevsky, Sutskever e Hinton (2012), mas com o conjunto de dados ILSVRC-2014, os autores alcançaram uma taxa de erro top-5 de 5,33%, representando uma melhora relativa de 20% em relação ao resultado anterior mais destacado para o mesmo conjunto de dados. Este resultado demonstra a eficácia da metodologia proposta, considerando que a taxa de erro top-5 humana é próxima, sendo ligeiramente inferior: 5.1% (RUSSAKOVSKY et al., 2014).

Observa-se, então, que há potencial de se gerar um melhor balanceamento em *datasets* ao se investir em métodos de *data augmentation*. Apesar do Método 2 (Seção 4.4.3) não ter mostrado variações positivas tão significativas, seu emprego em múltiplas classes pode ter o potencial de gerar melhorias na performance geral do modelo treinado, assim como realizado por Zhang et al. (2020). O Método 3 (Seção 4.4.4) resultou em métricas que evidenciam um considerável desempenho na ampliação da quantidade de instâncias de placas parcialmente apagadas/envelhecidas, tendo o maior impacto positivo entre os 3 métodos de *data augmentation* introduzidos neste trabalho.

Apesar de não discutida, a condição de placa pichada também foi estudada no *dataset*. Com uma quantidade limitante de 14 instâncias no treinamento e 9 na validação, a classificação obtida pelo modelo não foi satisfatória, o que justifica a procura por métodos de *data augmentation*. Um método de sobreposição para as placas pichadas, semelhante ao Método 1 (Seção 3.6.3.1), foi explorado, mas sem as melhorias almejadas. Entretanto, ainda que variações consideravelmente positivas tivessem sido obtidas nas medidas de desempenho, o pequeno subconjunto de validação para pichação não seria suficiente para afirmar que o modelo treinado generalizaria o suficiente. Esta limitação dificulta, ainda, a avaliação da efetividade do método descrito, que pode ter falhado apenas por não conseguir gerar placas artificiais que representem a possivelmente muito específica variabilidade das poucas placas pichadas reais disponíveis.

Por outro lado, além da limitação da quantidade de placas pichadas reais, há também possibilidade do problema de classificação de placas pichadas ser apenas mais complexo que o de clas-

sificação das condições consideradas na Seção 4.3. Variações no método de geração de pichação artificial foram levadas em conta, como no tamanho da pichação, diferentes pós-processamentos e quantidades de instâncias incluídas, entre outros, mas sem os esperados resultados. Também foram explorados outros caminhos, como a aplicação manual de pichação via *softwares* profissionais de edição de imagens, os quais permitem um melhor ajuste fino e, portanto, maior semelhança entre as instâncias geradas e as reais. Este último caso, embora parecesse promissor, também não gerou o impacto desejado. Assim sendo, é possível que a metodologia geral introduzida, no momento, não seja suficiente para lidar com a classificação de pichação em placas de trânsito reais. Estudos desta condição requerem, portanto, uma maior atenção no futuro.

Capítulo Cinco

Considerações Finais

5.1 Conclusões

Nesta pesquisa foi desenvolvido um conjunto de dados (*dataset*) com placas de trânsito e desenvolvida/analisada a aplicação de modelos de visão computacional baseados em redes neurais convolucionais (CNN) na realização dessa tarefa. Os diferenciais do *dataset* desenvolvido são a utilização de placas de trânsito nacionais e a presença de placas em situações desafiadoras como obstrução por vegetação e apagamento. A maioria dos *datasets* públicos contém placas de países europeus, que têm padrões e classes diferentes em relação às placas brasileiras, o que dificulta a classificação dos objetos pelos modelos de visão computacional. Além disso, esses *datasets* públicos contemplam basicamente placas em boas condições, o que pode levar aplicações como veículos autônomos a falhar quando encontrar situações como as mencionadas acima.

A inclusão de placas com vegetação e apagadas no *dataset* já resulta em uma melhoria dos modelos em classificar o tipo de placa nessas situações. Porém, o principal resultado da pesquisa é que com o *dataset* e modelos desenvolvidos é possível localizar e classificar tanto o tipo quanto a condição das placas, abrindo portas para melhoria da manutenção desses itens de segurança no trânsito. O propósito é facilitar a resolução do problema na raiz e não somente atuar na melhoria dos modelos computacionais, porque existem situações em que as placas estão totalmente obstruídas ou apagadas de tal forma que nem humanos nem computadores podem identificar o tipo da placa. Com uma melhor manutenção das placas espera-se evitar acidentes e multas, além de aumentar a eficiência de sistemas veiculares, principalmente os autônomos, que necessitam identificar com alta precisão as placas de trânsito.

Conforme apresentado no Capítulo 4, os modelos de detecção baseados em YOLO apresentaram um resultado promissor durante os processos de validação, mas, para uma aplicação mais realista, é necessária uma ampliação do *dataset*, incluindo as classes que não foram encontradas e aumentando o número de objetos das classes que possuem poucas instâncias como velocidade 100, rotatória, preferência, entre outras conforme Figura 4.1.

A arquitetura de CNN *multilabel* utilizada, ao se aproveitar do extrator de características ResNet50 e incorporando camadas totalmente conectadas independentes para atributos de tipo e condição, demonstrou a possibilidade da classificação simultânea. Essa abordagem inovadora pode ser usada de suporte tanto por motoristas humanos quanto veículos autônomos, ao mesmo tempo em que auxilia os serviços de manutenção a abordar prontamente questões relacionadas a qualidade de placas de trânsito, aprimorando assim a segurança no ambiente de tráfego.

O Hamming loss de validação do modelo CNN *multilabel* de 8,1% confirma a adequação da dis-

tribuição de classes do BRTSD para realizar a tarefa de classificação dupla nesta abordagem. Embora nosso estudo mostre um progresso significativo, é importante lembrar que o BRTSD não abrange todos os tipos de classes de sinalização de trânsito descritos nas Normas Brasileiras de Sinalização de Trânsito (BTSS). Além disso, este modelo foi testado em outro conjunto de dados brasileiro publicamente disponível, onde alcançou métricas de precisão, *recall* e pontuação F1 médias micro, macro e ponderadas, todas superiores a 90%. Ao contribuir tanto para assistência à condução quanto para a detecção de irregularidades em placas de trânsito, o modelo CNN multilabel desenvolvido se destaca em relação aos sistemas de assistência ao motorista convencionais.

Além disso, foi realizado um estudo de *data augmentation* à parte, direcionado à CNN *multilabel*, que resultou em 3 métodos focados em gerar variabilidade e aumento no número de instâncias nas condições de placas consideradas em nosso *dataset*. O Método 1 (Seção 3.6.3.1, condição de oclusão por vegetação), aplicado segundo uma estratégia comum de sobreposição de imagens artificiais, não gerou os resultados esperados, talvez devido às específicas imagens artificiais consideradas, possivelmente não adequadas ao contexto das imagens do nosso *dataset*. Com o Método 2 (Seção 3.6.3.2, condição de oclusão por vegetação), a aplicação de pós-processamento em placas reais resultou em melhoras nas métricas de *recall*, o que indica que esta estratégia pode também ser utilizada em outros casos onde há carência em número de instâncias. O Método 3 (Seção 3.6.3.3, condição de apagamento/envelhecimento) foi o que mais se destacou, gerando melhoras expressivas nas métricas de *recall* e mostrando grande potencial de melhora na classificação de placas apagadas e/ou envelhecidas.

Por fim, os métodos de *data augmentation* apresentados tiveram suas particularidades e enfatizaram pontos específicos de condições específicas (vegetação ou apagamento/envelhecimento). A metodologia geral, incluindo a forma de seleção de instâncias apropriadas e de avaliação se fez necessária e foi introduzida neste estudo. Entretanto, algumas das operações aplicadas nos métodos não são novas, como a adição de ruído e borrão à amostras disponíveis. Conclui-se, ainda, que os métodos podem e devem ser aprimorados, considerando o claro potencial de melhorias na tarefa de classificação.

5.2 Atividades Futuras de Pesquisa

Em atividades futuras é prevista a ampliação do *dataset* com agrupamento das placas, pois uma das limitações relacionadas aos resultados apresentados é que existem tipos de placas presentes no Manual Brasileiro de Sinalização de Trânsito que não foram encontradas e outras foram encontradas em poucas quantidades (ver 4.1). Essa ampliação também deve atender o aumento de instância de placas em más condições, possivelmente incluindo novas classes como placas com ferrugem, quebradas, dobradas etc.

Quanto ao estudo de algoritmos de *data augmentation*, pretende-se explorar variações nos mé-

todos criados, testando novos tipos de pós-processamentos, além de, em métodos como os da Seção 3.6.3.1, considerar outras imagens a serem sobrepostas às placas. Além disso, por haverem muitas variáveis a serem analisadas, fatores como a quantidade ótima de instâncias artificiais a serem incluídas no *dataset* não puderam ser considerados. Assim, trabalhos futuros buscarão explorar as lacunas deixadas de modo que se possa chegar a maiores conclusões sobre o tópico.

É prevista, também, uma ênfase na adição de placas pichadas reais ao *dataset* criado. Conforme discutido, a quantidade de instâncias dessas placas, até o momento da escrita deste trabalho, foi um fator limitante na aplicação dos modelos propostos e na avaliação das métricas. De tal forma, não é possível garantir que o modelo teria o mesmo *recall* ao ser colocado à prova no mundo real, dado que ele foi exposto a um conjunto muito pequeno de placas pichadas. Portanto, um aumento da quantidade de placas com esta condição implicará num maior balanceamento das classes do *dataset*, permitirá uma análise mais profunda sobre este específico caso e tenderá a amplificar a robustez geral do modelo.

Outras possíveis aplicações veiculares que podem se beneficiar da metodologia desenvolvida e ferramentas aplicadas são:

- Detecção de buracos nas ruas e rodovias;
- Detecção e classificação da sinalização horizontal, que também necessita de manutenção;
- Detecção e classificação de peças com defeitos na linha de montagem das montadoras ou fornecedores, auxiliando na qualidade das peças.

Por último, é necessário aplicar os modelos nos sistemas veiculares e analisar a eficiência dos mesmos.

Apêndice A

Testes com datasets públicos

A seguir serão apresentados alguns testes realizados com os datasets alemães GTSRB (STALLKAMP et al., 2012) e GTSDB (HOUBEN et al., 2013). O intuito destes testes é demonstrar que não é adequado utilizar somente esses datasets para aplicações de TSR com placas brasileiras e, por isso, foi necessário construir o nosso próprio dataset. Além disso, são apresentados resultados de validação para o dataset Stanford Cars (KRAUSE et al., 2013), demonstrando a aplicabilidade dos Algoritmos 9 a 11 em lidar com datasets de qualquer quantidade de atributos.

A.1 Modelo treinado no GTSRB_training e testado no GTSRB_test

No repositório do dataset GTSRB estão disponíveis dois *splits*, um para treinamento (GTSRB_training) e outro para teste (GTSRB_test). Os resultados da Tabela A.1 apresentam as métricas de classificação para este teste. Os resultados indicam que o modelo consegue classificar bem os tipos de placas alemães, com uma acurácia geral de aproximadamente 94%.

A.2 Modelo treinado no GTSRB_training e testado no BraTSD

Nesta seção são apresentados os resultados para o mesmo modelo anterior, porém testado em um dataset com placas brasileiras chamado de BraTSD. Nesse caso, existem classes que estão presentes no GTSRB e não no BraTSD. A Tabela A.2 mostra as métricas para as classes em comum nos dois datasets. Note que os melhores resultados referem-se à classe pare, porque as placas guardam maiores semelhanças em ambos os países. No geral, as classificações não são suficientemente boas, o que indica a necessidade de usar um dataset com placas nacionais, como o criado durante o projeto.

A.3 Modelo treinado no dataset proposto e testado no BraTSD

Nesse caso, o modelo treinado com o nosso dataset é testado no BraTSD. A Tabela A.3 apresenta as métricas para algumas classes em comum. Note que o resultado é bem melhor se comparado com a Tabela A.2, mesmo o dataset proposto sendo menor em comparação com o GTSRB_training. Isso mostra que a diferença entre cores, formatos e símbolos observadas nas placas de trânsito de diferentes regiões influencia diretamente na aplicação de métodos de classificação de imagens baseados em CNN. Por isso, foi necessária a criação do nosso dataset.

Tabela A.1: Métricas para modelo treinado no GTSRB_training e testado no GTSRB_test

	precision	recall	f1-score	support
animals	0.928	0.952	0.940	270
bend	0.787	0.656	0.715	90
bend left	0.950	0.950	0.950	60
bend right	0.724	0.933	0.816	90
construction	0.992	0.973	0.982	480
cycles crossing	0.926	0.967	0.946	90
danger	0.981	0.905	0.941	390
give way	0.973	0.992	0.982	720
go left	0.787	0.983	0.874	120
go left or straight	0.632	1.000	0.774	60
go right	1.000	0.729	0.843	210
go right or straight	0.904	0.942	0.922	120
go straight	0.972	0.995	0.984	390
keep left	0.292	0.989	0.451	90
keep right	0.958	0.767	0.852	690
no entry	1.000	0.886	0.940	360
no overtaking	0.980	0.998	0.989	480
no overtaking (trucks)	1.000	0.979	0.989	660
no traffic both ways	0.954	0.990	0.972	210
no trucks	0.961	0.987	0.974	150
pedestrian crossing	0.508	0.500	0.504	60
priority at next intersection	0.928	0.981	0.954	420
priority road	0.995	0.959	0.977	690
restriction ends	0.759	1.000	0.863	60
restriction ends (overtaking (trucks)	0.953	0.911	0.932	90
restriction ends (overtaking)	0.961	0.817	0.883	60
restriction ends 80	0.984	0.833	0.903	150
road narrows	0.917	0.978	0.946	90
roundabout	0.954	0.689	0.800	90
school crossing	0.942	0.973	0.957	150
slippery road	0.836	0.887	0.861	150
snow	0.928	0.853	0.889	150
speed limit 100	0.991	0.958	0.974	450
speed limit 120	0.949	0.953	0.951	450
speed limit 20	0.921	0.967	0.943	60
speed limit 30	0.891	0.989	0.937	720
speed limit 50	0.976	0.988	0.982	750
speed limit 60	0.980	0.969	0.974	450
speed limit 70	0.997	0.958	0.977	660
speed limit 80	0.973	0.913	0.942	630
stop	1.000	1.000	1.000	270
traffic signal	0.962	0.844	0.899	180
uneven road	0.981	0.883	0.930	120
accuracy	0.938			

Tabela A.2: Métricas para modelo treinado no GTSRB training e testado no BraTSD

	precision	recall	f1-score	support
pare	0.673	0.435	0.529	170
rotatoria	0.000	0.000	0.000	80
sentido obrigatorio	0.051	0.199	0.081	176
siga em frente ou a direita	0.194	0.087	0.120	150
siga em frente ou a esquerda	0.133	0.027	0.044	150
velocidade 50	0.319	0.429	0.365	84
velocidade 60	0.000	0.000	0.000	202
micro avg	0.152	0.160	0.156	1012

Tabela A.3: Métricas para modelo treinado no dataset proposto e testado no BraTSD

	precision	recall	f1-score	support
lombada	0.994	0.988	0.991	160.0
pare	1.000	1.000	1.000	170.0
proibido estacionar	0.995	0.968	0.981	218.0
proibido parar e estacionar	0.991	1.000	0.995	108.0
proibido retornar a esquerda	1.000	1.000	1.000	132.0
proibido virar a direita	0.983	0.358	0.525	162.0
proibido virar a esquerda	0.968	0.747	0.843	162.0
rotatoria	0.000	0.000	0.000	80.0
rotatoria a frente	0.588	0.989	0.737	88.0
sentido obrigatorio	0.374	1.000	0.545	176.0
siga em frente ou a direita	0.986	0.913	0.948	150.0
velocidade 40	1.000	1.000	1.000	80.0
velocidade 50	0.965	0.976	0.970	84.0
velocidade 60	0.990	0.985	0.988	202.0
micro avg	0.823	0.872	0.847	1972.0

A.4 Dataset Stanford Cars

O *dataset* Stanford Cars apresenta cerca de 8144 imagens de carros para treinamento e 8041 de validação. Cada imagem está rotulada com quatro atributos dos carros: modelo, marca, tipo e ano. A Tabela A.4 mostra, para cada atributo, a acurácia do modelo obtido sobre o conjunto de validação durante o treinamento da CNN *multilabel*. Note que a acurácia varia entre 0.836 e 0.909, o que indica uma boa capacidade do algoritmo em lidar com os quatro atributos ao mesmo tempo.

Tabela A.4: Resultados para classificação dos atributos do dataset Stanford Cars

Atributo	Acurácia
Tipo	0.909
Modelo	0.836
Marca	0.891
Ano	0.898

Apêndice B

Tutorial *FiftyOne*

A seguir será apresentado um breve tutorial sobre a biblioteca FiftyOne, que reúne recursos para ajudar as pessoas que trabalham em projetos de visão computacional.

B.1 Introdução

Atualmente os problemas de visão computacional (CV) estão associados à utilização de deep learning (DL). Uma das principais questões envolvidas no processo de obtenção de bons resultados é a utilização de um workflow que permita acelerar os experimentos e análises dos dados (imagens) e modelos. Entretanto, mesmo pensando de forma restrita a DL, o campo de CV apresenta um grande leque de possibilidades que precisam ser testadas.

Vamos supor, por exemplo, que uma aplicação de detecção de objetos utilizando DL esteja sendo criada do zero. O primeiro passo seria adquirir muitas imagens com placas de trânsito e anotar esses objetos. Esse processo de anotação consiste em utilizar um software assistente para desenhar bounding boxes em torno dos objetos de interesse e atribuir uma classe a cada bounding box. Como resultado, esse software retorna arquivos que contém a localização e a classe de cada objeto para serem utilizados nos treinamentos dos modelos DL.

Para a obtenção de bons modelos, é muito importante que o *dataset* tenha as seguintes características (JOCHER, 2020):

1. Uma grande quantidade e variedade de objetos anotados para evitar overfitting;
2. Sem erros nas classes atribuídas;
3. Todos os objetos de interesse anotados nas imagens;
4. *Bounding boxes* bem ajustados às bordas dos objetos.

Levando-se em conta que o processo de anotação é monótono e demorado devido à grande quantidade de imagens necessárias para a obtenção de um bom modelo, os itens 2 a 4 acima estão sujeitos a não serem atendidos adequadamente pela resposta humana, resultando em uma baixa qualidade das anotações e, consequentemente, do modelo produzido com o treinamento a partir desse *dataset*. Sem a utilização de uma ferramenta adequada, a revisão das anotações pode ser igualmente demorada e ineficiente. Além disso, é difícil para o olho humano fazer uma avaliação da singularidade de cada objeto anotado/imagem em relação ao restante do *dataset* para remover do *dataset* imagens que possam contribuir para a ocorrência de overfitting.

Depois de garantir a qualidade do *dataset*, é necessário utilizá-lo para treinar o modelo. Porém, a equipe se depara com as seguintes questões:

1. Qual framework utilizar? PyTorch, TensorFlow ou outros?
2. Qual arquitetura? YOLO (que por sua vez tem sete versões), Faster R-CNN, SSD, ou outras?
3. Qual o formato de anotação será utilizado? YOLO, COCO, Pascal VOC, csv, ou outros?
4. Como preparar subconjuntos do meu *dataset* completo para realizar testes?

O ideal é que todas as possibilidades listadas acima possam ser testadas e comparadas de maneira eficaz e rápida.

FiftyOne é uma biblioteca python que reúne recursos para auxiliar os desenvolvedores a garantir aspectos de qualidade de datasets e gerenciar todas as possibilidades para experimentação e avaliação de modelos. A seguir serão apresentados alguns desses recursos.

B.2 A classe *Dataset*

O principal elemento da biblioteca *FiftyOne* é a classe *Dataset*. Através dela é possível carregar, modificar, visualizar e avaliar datasets a partir de diferentes formatos de anotações.

O *FiftyOne* possui métodos integrados que fazem o download e carregam uma coleção (zoo) de datasets públicos, como o COCO e o Open Images. É possível também importar datasets locais em diversos formatos de anotação.

O código abaixo mostra como importar um conjunto de imagens e anotações do *Open Images* para a classe *Dataset*.

```
1  import fiftyone as fo
2  import fiftyone.zoo as foz
3  from fiftyone import ViewField as F
4  import fiftyone.brain as fob
5
6  try:
7      dataset = fo.load_dataset('open-images-v6-train')
8  except:
9      dataset = foz.load_zoo_dataset(
10                 "open-images-v6",
11                 split="train",
12                 label_types=["detections"],
```

```
            classes=["Traffic sign"],
        )

dataset.persistent = True
```

O objeto *dataset* criado em python reúne todas as informações referentes ao *dataset* anotado, como o caminho das imagens, as coordenadas e a classe de todas os objetos em cada imagem. *FiftyOne* permite o acesso a essas informações através da linguagem python e também a partir de uma interface gráfica. Vamos primeiro dar uma olhada nesta última, depois analisaremos algumas possibilidades com a linguagem python.

B.3 Interface gráfica do FiftyOne

Para abrir uma sessão da interface gráfica use a linha de código abaixo. O parâmetro fornecido é o objeto *dataset* criado anteriormente.

```
session = fo.launch_app(dataset)
```

A interface é apresentada na imagem B.1 e é criada na célula de um notebook jupyter ou pode ser acessada em um navegador pelo link http://localhost:5151/. Observe que a interface tem um layout em grid para facilitar a visualização das imagens. O tamanho das células do grid pode ser ajustado no canto superior direito. Clicando em uma dessas células é possível exapandir a visualização de uma imagem e visualizar detalhes específicos da mesma.

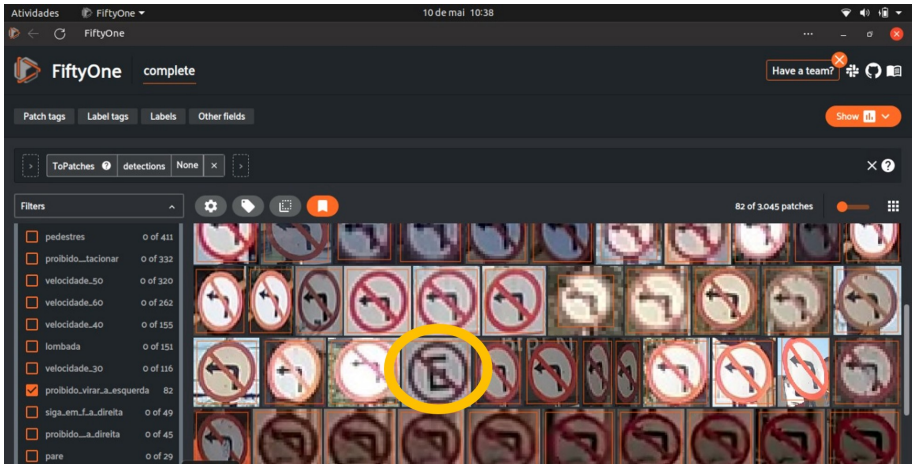


Figura B.1: Interface gráfica do *FiftyOne*

Na interface é possível visualizar os *bounding boxes* e a classe referentes às anotações importadas no objeto *dataset*. Na barra da lateral esquerda estão presentes alguns *fields* do *FiftyOne* com checkboxes que permitem a filtragem por classe, id das imagens, caminho do arquivo, metadata

(largura ou altura da imagem, por exemplo) ou tags fornecidas pelo usuário para cada imagem ou anotação. Além disso, o usuário pode criar outros *fields* via código.

Clicando em *labels* na parte superior da janela, a interface mostra um histograma com a distribuição de objetos por classe. É possível visualizar histogramas referentes a outros *fields* também.

Uma das opções mais interessantes na interface gráfica do *FiftyOne* é o botão 'patches' localizado na posição superior esquerda do grid. Clicando nesse botão aplica-se uma função de *crop* nas imagens do *dataset*, extraindo apenas as região anotadas e apresentando-as no grid para o usuário, como na Figura B.1. Com isso é possível ter uma visualização melhorada dos objetos anotados e encontrar erros de anotação com menos esforço do que ao usar um software de anotação.

Assim, a interface do *FiftyOne* fornece uma experiência visual muito útil para o usuário, permitindo que a revisão de um *dataset* seja mais rápida e precisa.

B.4 Explorando o objeto dataset com a linguagem python

B.4.1 Samples e Fields

Para o *FiftyOne*, *samples* são a unidade basica da classe *Dataset*. Os samples armazenam informações relacionadas a cada imagem ou video do objeto *dataset* e são compostos por atributos chamados *fields*. Durante a importação de um *dataset* o *FiftyOne* organiza todos os *samples* atribuindo alguns *fields* padronizados descritos a seguir:

- *id*: um identificador para representar cada *sample*,
- *media_type*: o tipo de mídia (imagem ou vídeo),
- *meta_data*: metadados das imagens (altura, largura),
- *filepath*: caminho da imagem ou vídeo associado ao *sample*,
- *filename*: nome do arquivo de imagem ou vídeo,
- *detections*: informações como *label* e coordenadas referentes aos *bounding boxes* do *dataset* (também há suporte para outros tipos de rótulos como classificação e segmentação).

É possível visualizar os dados do primeiro *sample* do *dataset* utilizando o método first(), conforme abaixo. Após executar, observe os *fields* mencionados acima.

```
1 dataset.first()
```

É possível acessar cada *field* do *sample*. Por exemplo, o conteúdo do *field* filepath do *sample* first() é obtido pelo código abaixo.

```
dataset.first().filepath
```

Os samples de um *dataset* podem ser iterados para obtenção e modificação dos valores dos *fields*. O código abaixo obtém uma lista com os caminhos de todas as imagens do *dataset*.

```
paths = []
for sample in dataset:
    paths.append(sample.filepath)
paths
```

Um sumário do objeto *dataset* contendo algumas informações úteis pode ser visualizado com a linha de código abaixo.

```
print(dataset)
```

Outras informações interessantes podem ser acessadas a partir das linhas de código abaixo.

```
#Número total de anotações no field detections
dataset.count("detections.detections.label")

# Dicionário com a quantidade de objetos por classe
dataset.count_values("detections.detections.label")

# Lista com o nome das classes no field detections
list(dataset.distinct("detections.detections.label"))
```

B.4.2 Views

As *views* podem ser pensadas como *subsets* de um objeto da classe *Dataset*. Quando aplicamos filtros na interface gráfica, estamos na verdade criando *views*.

Com esse recurso é possível explorar análises com variações do *dataset* completo.

O código abaixo mostra como criar *views* filtrando objetos e selecionando samples aleatoriamente a partir do objeto *dataset* criado anteriormente.

```
from fiftyone import ViewField as F

#cria uma view somente com os objetos da classe Person no field detections
view = dataset.filter_labels('detections',F('label')=='Person')

#o método take(x) retorna uma view com x samples aleatórias do dataset.
view2 = dataset.take(100)

#envia a view para a interface
session.view = view
```

B.5 Avaliação

É possível adicionar predições de modelos de visão computacional a um *Dataset* do *FiftyOne* e avaliá-las tanto de forma visual, através da interface gráfica, como computacionalmente através das métricas *precision*, *recall*, f1-score e mAP; das curvas *precision* x *recall* e matrizes de confusão. Para a avaliação computacional é necessário um *dataset* com os valores verdadeiros que serão comparados com as predições realizadas.

O código abaixo mostra como utilizar o detector yolov5 pre-treinado no *dataset* COCO para realizar predições da classe Person em um *view* do *dataset* carregado nos nossos exemplos anteriores. As predições são incorporadas em um *field* customizado do *dataset* e por fim avaliadas com os métodos do FiftyOne.

```
#Importação do modelo yolov5

import torch
import torchvision
weights = 'yolov5m'
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = torch.hub.load('ultralytics/yolov5', weights) #pré treinado com o dataset COCO
model.to(device)
model.eval()

print("Model ready")

from PIL import Image

# Adiciona predições nos samples
with fo.ProgressBar() as pb:
    for sample in pb(view):

        image = Image.open(sample.filepath)
        results = model(image)
        w, h = image.size
        a = results.pandas().xyxyn[0]

```

```
        detections = []
        for idx in a.index:
            x1, y1, x2, y2 = list(a.iloc[idx,0:4])
            label = a.iloc[idx,6]
            score = a.iloc[idx,4]
            rel_box = [x1, y1, (x2 - x1), (y2 - y1)]

            # Adiciona itens na lista de predições de cada sample
            detections.append(
                fo.Detection(
                    label=label,
                    bounding_box=rel_box,
                    confidence=score
                )
            )

        # Adiciona a lista de detecções ao field yolov5m
        sample[weights] = fo.Detections(detections=detections)
        sample.save()

# As detecções do yolo retornam a classe 'person' do dataset COCO,
# mas no Open Images temos a classe 'Person' com P maiúsculo.
# A avaliação do FiftyOne é case-sensitive, logo precisamos modificar o nome da classe em um dos
↪  fields.

for sample in view:
    for detection in sample.detections.detections:
        detection.label = 'person'
    sample.save()

# Atualiza o filtro do view
view = dataset.filter_labels('detections',F('label')=='person')

# Mostra precision, recall e f1-score para a classe que estamos analisando, 'person'.
results.print_report(classes=['person'])

# Mostra o mAP
results.mAP(classes=['person'])

# Exibe o gráfico precision x recall
plot = results.plot_pr_curves(classes=["person"])
plot.show()

# Exibe a matriz de confusão

fig = results.plot_confusion_matrix(classes=['person'])
fig.show()
session.plots.attach(fig)
```

B.6 Módulo Brain

Esse módulo do *FiftyOne* fornece ferramentas de análise de imagens baseadas em machine learning. Os códigos deste módulo são fechados, mas permitidos para uso comercial. Vamos dar uma olhada nos principais recursos desse módulo!

B.6.1 Clusterização de imagens e objetos

Segundo a documentação do *FiftyOne*, o método compute_visualization() permite gerar uma "representação de baixa dimensão"para as imagens e/ou para os objetos anotados nessas imagens. A ideia básica é utilizar modelos de machine learning para analisar uma grande quantidade de imagens/objetos e agrupá-los de acordo com a similaridade entre eles. Como resultado, essa análise pode ser apresentada graficamente para o usuário, podendo revelar alguns padrões que ajudem na tomada de decisões de um projeto de visão computacional.

O código abaixo mostra como o *FiftyOne* agrupa os objetos das classes *Person* e *Traffic sign* do nosso *dataset* já carregado anteriormente. Lembre-se que o termo *patch* refere-se à objetos anotados *bounding boxes*.

```
# Criando um view com as classes 'person' e 'Traffic sign'
view = dataset.filter_labels('detections',F('label').is_in(['person','Traffic sign']))

model = foz.load_zoo_model("mobilenet-v2-imagenet-torch") # Carrega um modelo do FiftyOne zoo

patches_field = "detections" # Field de objetos anotados que será analisado

embeddings = view.compute_patch_embeddings(model, patches_field) # Calcula os embeddings

results = fob.compute_visualization(view, patches_field=patches_field,
↪  embeddings=embeddings,brain_key='openimagessigns2')
```

No código abaixo geramos um gráfico com os resultados coletados acima, onde verifica-se o agrupamento dos objetos de cada classe. O gráfico é iterativo, de maneira que você pode selecionar alguns pontos e a interface gráfica atualiza o grid com essa seleção.

```
# Generate scatterplot
bbox_area = F("bounding_box")[2]  F("bounding_box")[3]
plot = results.visualize(
    labels=F("detections.detections.label"),
    sizes=F("detections.detections[]").apply(bbox_area),
)
plot.show(height=600)
session.plots.attach(plot)
```

Observe nas células de coordenadas (4-6,7-8) que existem pontos vermelhos e azuis que representam cada classe. Se você selecionar esses pontos, verá na interface gráfica que ocorre uma certa confusão do modelo aplicado na construção do *dataset* porque o mesmo classificou desenhos de pessoas nas placas de trânsito como sendo da classe 'person'. Com esse recurso, fica mais fácil analisar e corrigir o *dataset*, por exemplo.

B.6.2 Similaridade visual

O método compute_similarity() do *FiftyOne Brain* é um recurso para indexar imagens/objetos por similaridade visual. Os resultados desse método podem ser aplicados na operação sort_by_similarity() para ordenar as imagens/objetos de um *dataset* de acordo com a similaridade em relação a uma imagem/objeto de referência. Além dessa ordenação, as seguintes operações podem ser executadas a partir dos resultados do compute_similarity():

- find_duplicates(): utilizada para verificar se há imagens/objetos aproximadamente identicos no *dataset*.
- find_unique(): encontra um subset de imagens/objetos que são aproximadamente únicos no *dataset*.

B.6.3 Singularidade de imagens

O método compute_uniqueness() fornece uma medida de quão única uma imagem é em relação a todas as outras do *dataset*. Esse método não necessita de modelos pré-treinados para ser executado e é muito importante porque os modelos treinados com imagens singulares são mais eficientes.

O método fornece como saída um *field* chamado uniqueness para cada imagem. Esse *field* é preenchido com valores entre 0 e 1, onde a imagem com valor igual a 1 é a mais singular dentro do *dataset*.

B.6.4 Dureza da amostra

O método compute_hardness() é útil para identificar quais tipos de amostras são mais difíceis para o aprendizado do modelo. Como resultado tem-se um *field* chamado hardness que será preenchido com um valor numérico para cada *sample*, indicando a dificuldade do modelo em lidar com cada amostra.

B.7 Conclusão

Esse breve tutorial apresenta apenas alguns recursos do *FiftyOne* e foi possível perceber como essa biblioteca é de grande importância para o gerenciamento de datasets e modelos de visão computacional a fim de acelerar as análises, testes e avaliações dos projetos. Há muitos outros recursos do *FiftyOne* que podem ser explorados e a documentação disponível é bastante intuitiva. Qualquer dúvida ou sugestão a respeito deste tutorial ou sobre o *FiftyOne* favor contactar os autores.

Apêndice C

Algoritmos

Neste apêndice serão listados os algoritmos mencionados no relatório.

Algoritmo 1: Código para marcar placas não identificáveis cortadas pelas bordas e pequenas.

```
import fiftyone as fo
from fiftyone import ViewField as F
dataset = fo.load_dataset('dataset-name-here')

dataset.compute_metadata()

left = F("bounding_box")[0] < 0.001
right = F("bounding_box")[0] + F("bounding_box")[2] > 0.999
top =  F("bounding_box")[1] < 0.001
bottom = F("bounding_box")[1] + F("bounding_box")[3] > 0.999
dataset.untag_samples('edge')
dataset.untag_labels('edge')

dataset.filter_labels('detections',left | right | top |
↪  bottom).tag_labels('edge',label_fields='detections')

dataset.compute_metadata()
dataset.untag_labels('small')
bbox_area = round(
    F("$metadata.width") * F("bounding_box")[2] *
    F("$metadata.height") * F("bounding_box")[3]
)
small_boxes = bbox_area < 11 ** 2

dataset.filter_labels('detections',small_boxes).tag_labels('small',label_fields='detections')
```

Algoritmo 2: Código para aplicação do Método 1 (Seção 3.6.3.1) de *data augmentation*

```
import cv2
import numpy as np

def apply_medianblur(input_img, blur_param = 3):
    # Aplicando Median Blur
    output_img = cv2.medianBlur(input_img, blur_param)

    return output_img

def apply_gaussianblur(input_img, blur_param = 3):
    # Aplicando Gaussian Blur
    output_img = cv2.GaussianBlur(input_img, (blur_param, blur_param), 0)
```

```
13
14      return output_img
15
16  def add_veget(img, box):
17
18      # Choose vegetation image at random
19      num = np.random.randint(1, 16)
20      img_veg = cv2.imread('Images/veg'+str(num)+'.jpg')
21
22      # Sign position on image
23      p1 = (box[0], box[1])  # start point  (xmin, ymin)
24      p2 = (box[2], box[3])  # end point (xmax, ymax)
25      dim = (p2[1]-p1[1], p2[0]-p1[0])  # sign dimensions
26      dim2 = (p2[0]-p1[0], p2[1]-p1[1])  # correction for resize error
27      # Cropping an image
28      cropped_image = img[p1[1]:p2[1], p1[0]:p2[0]]  # sign region
29
30      # Resize graffiti to match sign's size
31      resized = cv2.resize(img_veg, dim2, interpolation=cv2.INTER_AREA)
32  #    resized = cv2.resize(img_veg, dim2, interpolation = cv2.INTER_CUBIC) # Resize graffiti
    ↪  to sign size
33
34      for y in range(1, dim[1]):
35          for x in range(1, dim[0]):
36              if (resized[x][y][0] < 150):
37                  cropped_image[x][y] = resized[x][y]
38
39      postprocessing = [np.random.choice([apply_medianblur, apply_gaussianblur]),
        ↪  np.random.choice([apply_medianblur, apply_gaussianblur])]
40      cropped_image = postprocessing[0](cropped_image)
41      cropped_image = postprocessing[1](cropped_image)
42
43      img[p1[1]:p2[1], p1[0]:p2[0]] = cropped_image
44      return img
```

Algoritmo 3: Código para aplicação do Método 2 (Seção 3.6.3.2) de *data augmentation*

```
1   import fiftyone as fo
2   import numpy as np
3   import cv2
4   from tqdm import tqdm
5
6   def apply_medianblur(input_img, blur_param = 3):
7       # Aplicando Median Blur
8       output_img = cv2.medianBlur(input_img, blur_param)
9
10      return output_img
11
12  def apply_motionblur(input_img, kernel_size = np.random.choice([7, 9, 13, 15, 17])):
13      # Definindo tamanho do kernel aleatoriamente
14      # kernel_size = np.random.choice([15,19])
```

```
15
16      # Definindo kernel com tamanho especificado
17      motion_blur_kernel = np.zeros((kernel_size, kernel_size))
18      motion_blur_kernel[int((kernel_size-1)/2), :] = np.ones(kernel_size)
19      motion_blur_kernel /= kernel_size
20
21      # Aplicando convolução
22      output_img = cv2.filter2D(input_img, -1, motion_blur_kernel)
23
24      return output_img
25
26  def apply_gaussianblur(input_img, blur_param = 3):
27      # Aplicando Gaussian Blur
28      output_img = cv2.GaussianBlur(input_img, (blur_param, blur_param), 0)
29
30      return output_img
31
32  # Percorrer as amostras originais
33  for i, original_sample in enumerate(tqdm(view.take(1000))):
34      if len(original_sample.detections.detections) == 1:
35          save_dir = '../../ignore/Datasets/filter_tests/new_method2/'
36
37          corresp_label = original_sample.detections.detections[0].label
38          vegetation_patches = dataset.filter_labels("detections", (F("condition") ==
            ↪  "vegetacao") & (big_boxes)
            ↪  &(F('label')==corresp_label)).to_patches("detections").match_tags(['filter',
            ↪  'veget_filter'], False)
39          if vegetation_patches.count() == 0:
40              continue
41
42          path = os.path.join(save_dir, original_sample.filename.replace('.jpg',
            ↪  '_filter.jpg'))
43
44          # Obter a detecção original da amostra
45          xmin, ymin, w, h = original_sample.detections.detections[0].bounding_box
46
47          img = cv2.imread(original_sample.filepath)
48          img2 = img.copy()
49
50          # Obter a largura e altura da detecção original
51          original_W = int(original_sample.metadata.width)
52          original_H = int(original_sample.metadata.height)
53
54          orig_bbox = [round(xmin*original_W), round(ymin*original_H),
            ↪  round((xmin+w)*original_W), round((ymin+h)*original_H)]
55
56          # Selecionar um patch de oclusão por vegetação aleatório
57          vegetation_patch = vegetation_patches.take(1, seed = i + 1).first()
58          veget_W = vegetation_patch.metadata.width
59          veget_H = vegetation_patch.metadata.height
60
61          # Carregar imagem respectiva ao patch de oclusão por vegetação usando OpenCV
```

```
        vegetation_image = cv2.imread(vegetation_patch.filepath)

        # Obter as coordenadas do patch de oclusão por vegetação
        xmin, ymin, w, h = vegetation_patch.detections.bounding_box
        veget_bbox = [round(xmin*veget_W), round(ymin*veget_H), round((xmin+w)*veget_W),
        ↪  round((ymin+h)*veget_H)]

        # Recortar a região do patch de oclusão por vegetação
        vegetation_crop = vegetation_image[veget_bbox[1]:veget_bbox[3],
        ↪  veget_bbox[0]:veget_bbox[2]]

        # Redimensionar o recorte da vegetação para corresponder à detecção substituída
        resized_vegetation_crop = cv2.resize(vegetation_crop, (orig_bbox[2]-orig_bbox[0],
        ↪  orig_bbox[3]-orig_bbox[1]))

        # Selecionar um efeito aleatório para definir pós-processamento
        postprocessing = np.random.choice([apply_medianblur, apply_motionblur,
        ↪  apply_gaussianblur])

        # Aplicar pós-processamento
        resized_vegetation_crop = postprocessing(resized_vegetation_crop)

        postprocessing = np.random.choice([apply_medianblur, apply_motionblur,
        ↪  apply_gaussianblur])
        resized_vegetation_crop = postprocessing(resized_vegetation_crop)

        detections = []

        # Criar nova amostra
        new_sample = fo.Sample(filepath=path)
        detections.append(fo.Detection(
                                label = vegetation_patch.detections.label,
                                bounding_box =
                                ↪  original_sample.detections.detections[0].bounding_box,
                                condition = vegetation_patch.detections.condition))

        img2[orig_bbox[1]:orig_bbox[3], orig_bbox[0]:orig_bbox[2]] = resized_vegetation_crop

        new_sample['detections'] = fo.Detections(detections=detections)
        dataset.add_sample(new_sample)
        dataset.select(new_sample).tag_samples('veget_filter')

        cv2.imwrite(path, img2)
```

Algoritmo 4: Código para aplicação do Método 3 (Seção 3.6.3.3) de *data augmentation*

```
import cv2

def apag_placa(image, bounding_box, increase_factor = 1.1, decrease_factor = 0.68):
    # Definição de máscara genérica
    mask = np.zeros_like(image, dtype=np.uint8)
```

```
    x1, y1, x2, y2 = bounding_box

    # Cálculo de dimensões de bounding box
    width = x2 - x1
    height = y2 - y1

    # Cálculo do centro do bounding box
    cx = int((x1 + x2) / 2)
    cy = int((y1 + y2) / 2)

    # Cálculo do raio da elipse em x e y
    radius_x = int(width / 2)
    radius_y = int(height / 2)

    # Definindo máscara elíptica
    cv2.ellipse(mask, (cx, cy), (radius_x, radius_y), 0, 0, 360, (255, 255, 255), -1)

    faded_roi = cv2.bitwise_and(image.copy(), mask)

    # Geração de ruído em toda a imagem
    noise = np.random.randint(0, 60, faded_roi.shape[:2], dtype=np.uint8)
    noise = cv2.merge([noise] * image.shape[2])  # Replicação de ruído nos três canais (red,
    ↪  green e blue)

    # Adicionar ruído em toda imagem
    faded_roi = cv2.add(faded_roi, noise)

    # Conversão para HSV
    faded_roi = cv2.cvtColor(faded_roi.astype(np.uint8), cv2.COLOR_RGB2HSV)

    # Ajustes aleatórios em saturação e brilho
    faded_roi[:, :, 1] = faded_roi[:, :, 1] * np.random.uniform(0.05, 0.1)
    faded_roi[:, :, 2] = faded_roi[:, :, 2] * np.random.uniform(1, 1.1)

    faded_roi = cv2.cvtColor(faded_roi, cv2.COLOR_HSV2RGB)

    # Aplicar contraste
    alpha = np.random.uniform(0.4, 0.7)
    beta = np.random.uniform(20, 45)
    faded_roi = cv2.convertScaleAbs(faded_roi, alpha=alpha, beta=beta)

    # Aplicando borrão dentro
    blur_radius = 7  # Adjust the blur radius as needed
    blurred_roi = cv2.GaussianBlur(faded_roi, (blur_radius, blur_radius), 0)

    # Substituindo região de efeitos dentro da elipse na imagem original
    modified_image = image.copy()
    modified_roi = cv2.bitwise_and(modified_image[y1:y2, x1:x2], cv2.bitwise_not(mask[y1:y2,
    ↪  x1:x2])) + cv2.bitwise_and(blurred_roi[y1:y2, x1:x2], mask[y1:y2, x1:x2])
    modified_image[y1:y2, x1:x2] = modified_roi

    new_roi = modified_image.copy()
```

```
56      new_mask = np.zeros_like(image, dtype=np.uint8)
57      new_radius_x = int(radius_x*increase_factor)
58      new_radius_y = int(radius_y*increase_factor)
59      cv2.ellipse(new_mask, (cx, cy), (int(new_radius_x/2)*2, int(new_radius_y/2)*2), 0, 0,
        ↪  360, (255, 255, 255), -1)
60
61      # Borrão que engloba região externa à placa para criar transição entre imagem original e
        ↪  região de elipse (com efeitos)
62      blurred_roi2 = cv2.medianBlur(new_roi, 5)
63
64      new_x1 = cx - round(new_radius_x)
65      new_x2 = cx + round(new_radius_x)
66      new_y1 = cy - round(new_radius_y)
67      new_y2 = cy + round(new_radius_y)
68      if new_x1 < 0:
69          new_x1 = 0
70      if new_x2 < 0:
71          new_x2 = 0
72      if new_y1 < 0:
73          new_y1 = 0
74      if new_y2 < 0:
75          new_y2 = 0
76
77      modified_roi2 = cv2.bitwise_and(modified_image[new_y1:new_y2, new_x1:new_x2],
        ↪  cv2.bitwise_not(new_mask[new_y1:new_y2, new_x1:new_x2])) +
        ↪  cv2.bitwise_and(blurred_roi2[new_y1:new_y2, new_x1:new_x2], new_mask[new_y1:new_y2,
        ↪  new_x1:new_x2])
78      modified_image[new_y1:new_y2, new_x1:new_x2] = modified_roi2
79
80      new_roi = modified_image.copy()
81      new_mask = np.zeros_like(image, dtype=np.uint8)
82      new_radius_x = int(radius_x*decrease_factor)
83      new_radius_y = int(radius_y*decrease_factor)
84      cv2.ellipse(new_mask, (cx, cy), (new_radius_x, new_radius_y), 0, 0, 360, (255, 255,
        ↪  255), -1)
85
86      blur_radius2 = 5
87      blurred_roi2 = new_roi
88      # blurred_roi2 = cv2.GaussianBlur(new_roi, (blur_radius2, blur_radius2), 0) # Borrão
        ↪  opcional
89
90      new_x1 = cx - round(new_radius_x)
91      new_x2 = cx + round(new_radius_x)
92      new_y1 = cy - round(new_radius_y)
93      new_y2 = cy + round(new_radius_y)
94      if new_x1 < 0:
95          new_x1 = 0
96      if new_x2 < 0:
97          new_x2 = 0
98      if new_y1 > image.shape[0]:
99          new_y1 = image.shape[0]
100     if new_y2 > image.shape[0]:
```

```
        new_y2 = image.shape[0]

    modified_roi2 = cv2.bitwise_and(modified_image[new_y1:new_y2, new_x1:new_x2],
    ↪  cv2.bitwise_not(new_mask[new_y1:new_y2, new_x1:new_x2])) +
    ↪  cv2.bitwise_and(blurred_roi2[new_y1:new_y2, new_x1:new_x2], new_mask[new_y1:new_y2,
    ↪  new_x1:new_x2])
    modified_image[new_y1:new_y2, new_x1:new_x2] = modified_roi2

    return modified_image
```

Algoritmo 5: Aquisição de imagens para *dataset*.

```
import os
import sys
sys.path.append(os.path.abspath('./'))
sys.path.append('./CNN')
import requests
import json
from tqdm import tqdm
import fiftyone.brain as fob
import fiftyone.zoo as foz
import fiftyone as fo
import torch
from PIL import Image
from datetime import datetime, timezone, timedelta

weight = '../ignore/artifacts/10_05_2023/YOLO_NF_S_300_10-05-23_131438/weights/best.pt'
yolo = torch.hub.load('ultralytics/yolov5', 'custom', weight)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
yolo.to(device)
yolo.eval()

CNN = torch.load('./training/runs/06_06_2023/09_26_21/best.pth')
CNN.eval()
CNN.to(device)

print("Model ready", weight.split('/')[-1])

data_root = 'C:\\Users\\VMADUREI\\Desktop\\GIT\\ignore\\Datasets\\new_annotations/'

tzinfo = timezone(timedelta(hours=-3.0))
project_name = datetime.now(tzinfo).strftime("%m-%d-%Y")

dataset = fo.load_dataset([i for i in fo.list_datasets() if '01-06-23' in i][-1])
print('Based on', dataset.name)
values = dataset.count_values("detections.detections.label")

done_file = data_root.replace('ignore','traffic-signals-detection') + 'done.txt'

with open(done_file) as file:
    done = [line.rstrip() for line in file]
```

```
40
41  response =
    ↪  requests.get('https://api.openstreetcam.org/2.0/sequence/?userId=13979&itemsPerPage=1480').text
42  response_info = json.loads(response)
43
44  tracks = []
45  lista = response_info['result']['data']
46  lista.reverse()
47
48  for d in lista:
49
50          if datetime.strptime(d['dateAdded'], '%Y-%m-%d %H:%M:%S') > datetime(2020,10,1) and
            ↪  int(d['dateAdded'][11:13]) < 15 and int(d['countActivePhotos']) < 20000 and
            ↪  d['id'] not in done and 'Paulo' in d['address']:
51              tracks.append(d['id'])
52
53  total = len(tracks)
54
55  for j, cod in enumerate(tracks):
56      if fo.dataset_exists(cod):
57          dataset = fo.load_dataset(cod)
58          if dataset.count() > 0:
59              # Verifica último download para datasets existentes
60              last = max([int(x['filepath'].split('/')[-1].split('_')[-1].split('.jpg')[0])
61                          for x in dataset.to_dict()['samples']])
62              save = dataset.first()['filepath'].split(cod + '_')[0][:-1]
63          else:
64              last = 0
65              save = os.path.join(data_root,project_name,cod)
66      else:
67          dataset = fo.Dataset(cod)
68          last = 0
69          save = os.path.join(data_root,project_name,cod)
70
71      dataset.persistent = True
72
73      if not os.path.exists(save):
74          os.makedirs(save)
75
76      os.chdir(save)
77
78      response = requests.get('https://api.openstreetcam.org/2.0/sequence/'+cod).text
79      response_info = json.loads(response) # string to dict
80      tam = response_info['result']['data']["countActivePhotos"]
81      response =
        ↪  requests.get('https://api.openstreetcam.org/2.0/sequence/'+cod+'/photos?itemsPerPage='+tam).text
82      response_info = json.loads(response) # json to dict
83      try:
84          re = response_info['result']['data']
85      except:
86          continue
87
```

```
    print('Downloading track ',cod,', ',j,'/',total,'\n')

    for i, d in enumerate(tqdm(re)):

        if last and i <= last:
            continue

        filepath = cod + '_' + str(i) + '.jpg'

        link = d['fileurlLTh']

        try:
            image = Image.open(requests.get(link, stream=True).raw)
        except:
            continue
        results = yolo(image)
        a = results.pandas().xyxyn[0]
        b = results.pandas().xyxy[0]

        if len(a) > 0:

            detections = []
            add_sample = True
            for idx in a.index:
                x1, y1, x2, y2 = list(a.iloc[idx,0:4])
                label = a.iloc[idx,6]
                score = a.iloc[idx,4]
                rel_box = [x1, y1, (x2 - x1), (y2 - y1)]

                crop = image.crop(list(b.iloc[idx,0:4]))
                prediction = CNN.prediction(crop,device)
                label = prediction['type']['label']
                if (len(a) == 1 and values[label] > 1000) or
                ↪  rel_box[2]*image.size[0]*rel_box[3]*image.size[1] < 12**2 or
                ↪  prediction['condition']['label'] == 'boa':
                    add_sample = False
                    break

                print(prediction)

                detections.append(
                    fo.Detection(
                        label=prediction['type']['label'],
                        bounding_box=rel_box,
                        confidence=prediction['type']['score'],
                        condition = prediction['condition']['label']
                    )
                )

            if add_sample:
                image.save(filepath)
                new_sample = fo.Sample(filepath=filepath)
```

```
                new_sample['detections'] = fo.Detections(detections=detections)
                dataset.add_sample(new_sample)

    if dataset.has_sample_field('detections'):
        dataset.tag_samples(cod)

        # SIMILARITY ANALISYS
        model = foz.load_zoo_model("mobilenet-v2-imagenet-torch")
        patches_field = "detections"
        embeddings = dataset.compute_patch_embeddings(model, patches_field)
        fob.compute_similarity(
            dataset, patches_field=patches_field,
            ↪ brain_key=patches_field,embeddings=embeddings)

    else:
        dataset.delete()
        os.rmdir(save)

    with open(done_file,'a') as file:
        file.write(cod + '\n')

print('\n Finished!')
```

Algoritmo 6: Código para geração da Figura 3.8.

```
import os
from PIL import Image

def merge_images(original_image, traffic_sign_images, sign_coordinates, spacing = 3):
    # Abrir imagem original
    original = Image.open(original_image)

    x_min, y_min, x_max, y_max = sign_coordinates

    # Cortar imagem original para obter região de interesse (ROI, region of interest)
    original_roi = original.crop((x_min, y_min, x_max, y_max))

    # Calcular espessura e altura da placa
    sign_width = x_max - x_min
    sign_height = y_max - y_min

    # Calcular tamanho final da imagem que será criada via PIL
    total_width = original_roi.width + len(traffic_sign_images) * (sign_width + spacing)
    max_height = max(original_roi.height, sign_height)

    # Criar imagem em branco (espaço branco) com a largura desejada
    white_space = Image.new('RGB', (spacing, max_height), color=(255, 255, 255))

    # Criar imagem final com o tamanho total calculado
    new_image = Image.new('RGB', (total_width, max_height), color=(255, 255, 255))

```

```
27      # Colando placa original na extremidade esquerda (x=0, y=0)
28      new_image.paste(original_roi, (0, 0))
29
30      # Colar espaços brancos e placas de trânsito lado a lado
31      current_x = original_roi.width
32      for sign_image in traffic_sign_images:
33          sign_image = Image.open(sign_image)
34          sign = sign_image.crop((x_min, y_min, x_max, y_max))
35
36          # Espaço branco antes de cada placa
37          new_image.paste(white_space, (current_x, 0))
38          current_x += spacing
39
40          # Colar recorte de placa
41          new_image.paste(sign, (current_x, 0))
42
43          # Atualizar a posição x para a próxima placa
44          current_x += sign_width
45
46      return new_image
47
48
49  if __name__ == "__main__":
50
51      if os.path.exists("./figura_relatorio/output_image.jpg"):
52          os.remove("./figura_relatorio/output_image.jpg")
53
54      original_image_path = '../Datasets/_images/main/3218050_3819.jpg'
55      sign_images = os.listdir("./figura_relatorio/")
56      traffic_sign_images_paths = ["./figura_relatorio/"+filename for filename in sign_images]
57
58      image_width, image_height = 1280, 720
59      xmin, ymin, w, h = [0.970704, 0.349096, 0.029296, 0.069178]
60
61      sign_coordinates = [round(xmin*image_width), round(ymin*image_height),
        ↪  round((xmin+w)*image_width), round((ymin+h)*image_height)]
62
63      factor_x = 0.05
64      factor_y = 0.25
65
66      sign_coordinates[0] = int(sign_coordinates[0]) - int(sign_coordinates[0]*factor_x)
67      sign_coordinates[1] = int(sign_coordinates[1]) - int(sign_coordinates[1]*factor_y)
68      sign_coordinates[3] = int(sign_coordinates[3]) + int(sign_coordinates[3]*factor_y)
69
70      output_image = merge_images(original_image_path, traffic_sign_images_paths,
        ↪  sign_coordinates)
71
72      output_image.save("./figura_relatorio/output_image.jpg")
73
```

Algoritmo 7: Treinamento de yolov5 com validação cruzada.

```
#!/usr/bin/env python3

import os
import shutil
import yaml
from tqdm import tqdm
from datetime import datetime, timezone, timedelta
import glob

tzinfo = timezone(timedelta(hours=-3.0))
date = datetime.now(tzinfo).strftime("%d_%m_%Y")
PROJECT='artifacts/no_filter/' + date
EPOCHS=600
BATCH=128
WORKERS=16
WEIGHTS = 'yolov5m.pt'

data_root = 'Datasets/YOLO_double_04-07-23'
images_path = data_root + '/images/train'
labels_path = data_root + '/labels/train'
YAML = data_root + '/dataset.yaml'

files = glob.glob(data_root + '/labels/*/*.txt') #anotações
other_files = glob.glob(os.path.dirname(data_root) + '/**/*.jpg',recursive=True) #imagens
error_files = []
print('checking files')
for filepath in tqdm(files):
    filename = os.path.basename(filepath)
    try:
        file = [x for x in other_files if filename[:-4] in x and data_root.split('/')[-1]
        ↪  not in x][0]
    except Exception as e:
        print(e)
        error_files.append(filename)
        continue
    split = os.path.dirname(filepath.replace('labels','images'))
    if not os.path.exists(split):
        os.makedirs(split)
    new_path = os.path.join(split,filename.replace('txt','jpg'))
    if not os.path.exists(new_path):
        shutil.copy(file,new_path)

assert error_files == []

folds = [x for x in os.listdir(data_root + '/images') if 'fold' in x ]
folds.sort()

for fold in folds:

    print('FOLD:', fold)

```

```
    try:
        shutil.rmtree(images_path)
        shutil.rmtree(labels_path)
    except:
        print('No dirs')

    #cria diretórios de treinamento
    os.makedirs(images_path)
    os.makedirs(labels_path)

    #Ajustando o arquivo yaml
    with open(YAML) as f:
        doc = yaml.safe_load(f)

        doc['path'] = '/s/vmadurei/traffic-signals-detection/' + data_root
        doc['train'] = './images/train'
        doc['val'] = './images/' + fold

    with open(data_root + '/dataset.yaml', 'w') as f:
        yaml.dump(doc,f)

    for root, dirs, files in os.walk(data_root,topdown=False):
        if 'fold' in root and not fold in root:
            print('Copying files from', root)
            for name in tqdm(files):
                f = os.path.join(root, name)
                if name.endswith('.txt'):
                    shutil.copy(f, labels_path)
                elif name.endswith('.jpg') or name.endswith('.png'):
                    shutil.copy(f, images_path)

    images = os.listdir(images_path)
    annotations = os.listdir(labels_path)
    images.sort()
    annotations.sort()

    assert [x[:-4] for x in images] == [x[:-4] for x in annotations]

    models = ['n','s','m','l','x']
    for letter in models:
        date = datetime.now(tzinfo).strftime("%H%M%S")
        weights = f'yolov5{letter}.pt'
        NAME = f'multi_cls_{EPOCHS}ep_{date}_{fold}_{weights[:-3]}'
        print(NAME)
        #train
        os.system(f'python -m torch.distributed.launch --nproc_per_node 8
        ↪  YOLO/yolov5/train.py \
        --batch-size {BATCH} \
        --epochs {EPOCHS} \
        --data {YAML} \
        --weights {weights} \
        --workers {WORKERS} \
```

```
        --name {NAME} \
        --project {PROJECT} \
        --cache')
```

Algoritmo 8: Funções para validação cruzada e exportação de dataset em formato yolov5.

```
def kfold(view,k=5,exception=['test']):
    #list and remove actual tags from dataset, except test tag
    folds = [x for x in view.distinct("tags") if 'fold' in x]
    print('Folds existentes: ', folds, '/ removendo folds antigas')
    view.untag_samples(folds)


    #create new tags based on k
    exception = exception.copy()
    base_len = int(view.match_tags(exception,False).count()/k)
    k_len = [base_len for x in range(k)]
    k_len[-1] = k_len[-1] + (view.match_tags(exception,False).count() -
    ↪  int(view.match_tags(exception,False).count()/k)*k)


    if sum(k_len) == view.match_tags(exception,False).count():
        print('Ok splitting')


        for x in range(k):
            view.match_tags(exception,False).take(k_len[x]).tag_samples('fold'+str(x))
            exception.append('fold'+str(x))


def yolov5_export(view,export_dir,label_field='detections',export_media=True,split =
↪  ['train','val','test']):
    classes = list(view.count_values("detections.detections.label").keys())
    for x in split:
        view.match_tags(x).export(
            export_dir=export_dir,
            dataset_type=fo.types.YOLOv5Dataset,
            label_field=label_field,
            split=x,
            classes=classes,
            export_media=export_media
        )
```

Algoritmo 9: Código para definição da arquitetura da CNN multilabel.

```
import torch
import torchvision
import torch.nn as nn
import numpy as np
from PIL import Image
```

```
6
7  class TSMultiLabel(nn.Module):
8      def __init__(self, labels, transforms, model = 'resnet50', weights='DEFAULT', **kwargs):
9          super().__init__()
10
11         self.labels = labels
12         #model pode ser alguma variação da ResNet, testado resnet50 e resnet101
13         exec(f'self.resnet = torchvision.models.{model}(weights="{weights}")')
14         self.model_wo_fc = nn.Sequential(*(list(self.resnet.children())[:-1]))
15
16         # if weights != None:
17         #     for param in self.model_wo_fc.parameters():
18         #         param.requires_grad = False
19
20         in_features = self.resnet.fc.in_features
21
22         self.flatten = nn.Flatten()
23
24         self.dropout = nn.Dropout(0.2)
25
26         fc = {}
27         for label in self.labels:
28             out_features =len(self.labels[label])
29             fc[label] = nn.Linear(in_features,out_features)
30         self.fc = nn.ModuleDict(fc)
31
32         self.transforms = transforms
33
34     def forward(self,X):
35         X= self.model_wo_fc(X)
36         X = self.flatten(X)
37         X = self.dropout(X)
38
39         output = {}
40
41         for label in self.labels:
42             output[label] = self.fc[label](X)
43
44         return output
45
46
47     def prediction(self, image: Image,device):
48         self.eval()
49         image = self.transforms(image).unsqueeze(0)
50         image.to(device)
51         outputs = self(image)
52         func_act = torch.nn.Softmax(dim=1)
53         output = {}
54         for label in self.labels:
55             output[label]= {}
56             output[label]['label'] =
               ↪  self.labels[label][np.argmax(outputs[label].cpu().detach().numpy())]
```

```
            output [label]['score'] =
            ↪  np.max(func_act(torch.Tensor(outputs[label].cpu().detach())).numpy())

        return output
```

Algoritmo 10: Execução do treinamento para CNN *multilabel*.

```
import os
import torch
import torchvision.transforms as T
from torch.utils.data import Dataset, DataLoader
from datasets import *
from model import TSMultiLabel
import numpy as np
import pandas as pd
from sklearn.metrics import classification_report

if torch.cuda.device_count() < 2:
    import fiftyone as fo
    #from _utils.cnn_torch_fo import *

import torch.nn.functional as TF
from tqdm import tqdm
from datetime import datetime, timezone, timedelta
import glob
import json
from csv import writer

from torch.utils.tensorboard import SummaryWriter

import torch.multiprocessing as mp
from torch.utils.data.distributed import DistributedSampler
from torch.nn.parallel import DistributedDataParallel as DDP
from torch.distributed import init_process_group, destroy_process_group, barrier, reduce

def ddp_setup():
    init_process_group(backend="nccl")
    torch.cuda.set_device(int(os.environ["LOCAL_RANK"]))

class Trainer:
    def __init__(
        self,
        model: torch.nn.Module,
        train_dataloader: DataLoader,
        val_dataloader: DataLoader,
        optimizer: torch.optim.Optimizer,
        ddp_mode: bool,
        args
    ) -> None:

        self.args = args
```

```
45          self.ddp_mode = ddp_mode
46
47          if ddp_mode:
48              self.device = int(os.environ["LOCAL_RANK"])
49          else:
50              self.device = torch.device('cuda') if torch.cuda.is_available() else
                ↪  torch.device('cpu')
51              print(f'{torch.cuda.device_count()} GPUs available. Using {self.device}')
52
53          self.model = model.to(self.device)
54
55          if ddp_mode:
56              self.model = DDP(self.model,
                ↪  device_ids=[self.device],find_unused_parameters=True)
57
58          self.is_save_dev = not self.ddp_mode or (self.ddp_mode and self.device == 0)
59
60          self.train_dataloader = train_dataloader
61          self.val_dataloader = val_dataloader
62          self.optimizer = optimizer
63          self.attributes = list(val_dataloader.dataset[0]['labels'].keys())
64          self.len_attributes = len(self.attributes)
65          self.save_dir = args.name
66
67          self.results = {
68              'epoch': 0,
69  #             'train_type_acc': 0,
70  #             'train_condition_acc': 0,
71  #             'train_mean_acc': 0,
72  #             'train_mean_weighted_f1': 0,
73              'val_mean_acc': 0,
74              'val_mean_weighted_f1': 0,
75              'train_loss': 0,
76              'val_loss': 0,
77              'train_hamming_loss': 0,
78              'val_hamming_loss': 0,
79              'dataset': args.dataset
80          }
81
82          self.bests = {
83              'val_mean_acc': 0,
84              'val_mean_weighted_f1': 0,
85              'val_hamming_loss': 10000,
86              'val_loss': 100000,
87          }
88
89          for x in self.attributes:
90              self.results['val_' + x + '_acc'] = 0
91              self.bests['val_' + x + '_acc'] = 0
92
93
94          if self.is_save_dev:
```

```
95              # Tensorboard initialization
96              dataiter = iter(train_dataloader)
97              data = next(dataiter)
98              self.writer = SummaryWriter(log_dir=self.save_dir)
99              #self.writer.add_graph(model,data['image'],use_strict_trace=False)
100
101             if not os.path.exists(self.save_dir):
102                 os.makedirs(self.save_dir)
103
104             with open (self.save_dir + '/results.csv','x',newline='') as file:
105                 writer_object = writer(file)
106                 writer_object.writerow(list(self.results.keys()))
107
108     def _run_batch(self, source, targets):
109
110         if self.model.training:
111             # zero the parameter gradients
112             self.optimizer.zero_grad()
113
114         with torch.set_grad_enabled(self.model.training):
115             outputs = self.model(source)
116             loss = 0
117             #print(outputs)
118             #weights = [3,22]
119
120             for key in outputs:
121
122                 loss_ = TF.cross_entropy(outputs[key],
                    ↪  targets[key].to(self.device))#*weights[i]
123                 loss += loss_
124                 self.hamming_loss = self.hamming_loss + torch.sum(torch.max(outputs[key],
                    ↪  1)[1] != targets[key].to(self.device))
125                 self.match_[key][0] += targets[key].to(self.device).tolist()
126                 self.match_[key][1] += torch.max(outputs[key], 1)[1].tolist()
127
128             #loss = loss_function(outputs, labels.float().view(*outputs.shape))
129
130             if self.model.training:
131                 # backward
132                 loss.backward()
133                 self.optimizer.step()
134
135         # current loss
136         self.current_loss = self.current_loss + loss
137
138     def _run_epoch(self, epoch):
139
140         data_loader = self.train_dataloader
141         if not self.model.training:
142             data_loader = self.val_dataloader
143
144         if self.ddp_mode and self.model.training:
```

```
145             data_loader.sampler.set_epoch(epoch)
146
147         pbar_disable = True
148         if self.is_save_dev:
149             print('Train' if self.model.training else 'Val')
150             pbar_disable = False
151
152         length = len(data_loader.dataset)
153
154         prog_bar = tqdm(data_loader, total=len(data_loader),disable=pbar_disable )
155         self.match_ = {key: [[],[]] for key in self.attributes}
156         self.current_loss = torch.zeros(1,requires_grad=True).to(self.device)
157         self.hamming_loss = torch.zeros(1,requires_grad=True).to(self.device)
158
159         for data in prog_bar:
160             source = data['image'].to(self.device)
161             targets = data['labels']
162             self._run_batch(source, targets)
163
164         self.hamming_loss = (self.hamming_loss / (length*self.len_attributes))
165
166         loss = (self.current_loss / length)
167
168         if self.model.training:
169             operation = 'train'
170             if self.ddp_mode:
171                 reduce(self.hamming_loss,dst=0) #isso vai somar a loss em todas as gpus
172                 reduce(loss,dst=0)
173         else:
174             operation = 'val'
175
176         if self.is_save_dev:
177
178             self.results[f'{operation}_hamming_loss'] = self.hamming_loss.item()
179             self.results[f'{operation}_loss'] = loss.item()
180
181             if not self.model.training:
182                 self.update_results()
183
184     def update_results(self):
185
186         report = {}
187         labels = self.args.labels
188         zero_precision = False
189         for x in self.match_:
190             report[x] = classification_report(self.match_[x][0], self.match_[x][1],
                ↪  target_names=labels[x],output_dict=True)
191
192             # print(report[x])
193             df = pd.DataFrame(report[x])[:-1]
194             for y in df:
195                 if (df[y]==0).all():
```

```
                    print(f'Classe {y} sem classificações')
                    zero_precision = True

            print(classification_report(self.match_[x][0], self.match_[x][1],
            ↪  target_names=labels[x]))

        operation = 'train' if self.model.training else 'val'
        cum1, cum2 = 0, 0
        for key in report:
            self.results[f'{operation}_{key}_acc'] = report[key]['accuracy']
            cum1 += report[key]['accuracy']
            cum2 += report[key]['weighted avg']['f1-score']

        self.results[f'{operation}_mean_acc'] = cum1/self.len_attributes
        self.results[f'{operation}_mean_weighted_f1'] = cum2/len(report)

        print('='*30 + ' Results ' + '='*30)
        if not zero_precision:
            for metric in self.bests:
                check = self.bests[metric] > self.results[metric] if 'loss' in metric else
                ↪  self.bests[metric] < self.results[metric]
                if check:
                    self.bests[metric] = self.results[metric]
                    if self.ddp_mode:
                        torch.save(self.model.module, self.save_dir + f'/{metric}.pth')
                    else:
                        torch.save(self.model, self.save_dir + f'/{metric}.pth')
                    print(f"{metric} checkpoint saved at {self.save_dir}")

        for metric in self.results:
            if metric in self.bests.keys():
                print(f'{metric}: {self.results[metric]} / best: {self.bests[metric]}')
            else:
                print(f'{metric}: {self.results[metric]}')

        print('='*68 + '\n')

        with open (self.save_dir + '/results.csv', 'a',newline='') as file:
            writer_object = writer(file)
            writer_object.writerow(list(self.results.values()))

    def train(self, max_epochs: int):
        for epoch in range(max_epochs):
            self.results['epoch'] = epoch
            if self.is_save_dev:
                print(f'Epoch: {epoch}/{max_epochs-1}')

            #training
            self.model.train()
            self._run_epoch(epoch)

            if self.is_save_dev:
```

```
246                 #add to tensorboard
247                 for result in [x for x in self.results if 'train' in x]:
248                     self.writer.add_scalar(f'train/{result}',self.results[result],epoch)
249 
250                 #validation
251                 self.model.eval()
252                 self._run_epoch(epoch)
253 
254                 #add to tensorboard
255                 for result in [x for x in self.results if 'val' in x]:
256                     self.writer.add_scalar(f'val/{result}',self.results[result],epoch)
257         if self.is_save_dev:
258             torch.save(self.model, self.save_dir + '/last.pth')
259         return self.results
260 
261     def validation_report(self,samples=None):
262         report = {}
263 
264         for label in self.model.labels:
265             report['gt_' + label] = []
266             report['pred_' + label] = []
267             report[label + '_match'] = []
268 
269         report['all_match'] = []
270 
271         device = torch.device('cpu')
272 
273         self.model.to(device)
274 
275         if samples == None:
276             samples = self.val_dataloader.dataset
277 
278         for sample in samples:
279             img = T.ToPILImage()(sample['image'])
280             pred = self.model.prediction(img,device)
281 
282             gt = {}
283             all_match = True
284             for label in pred:
285                 gt = self.model.labels[label][sample['labels'][label]]
286                 pred_ = pred[label]['label']
287                 match_ = pred_ == gt
288 
289                 if not match_:
290                     all_match = False
291 
292                 report['gt_' + label].append(gt)
293                 report['pred_' + label].append(pred_)
294                 report[label + '_match'].append(match_)
295 
296             report['all_match'].append(all_match)
297 
```

```
298         df = pd.DataFrame(report)
299 
300         # def calc(df_):
301 
302         #     for key in df_.columns:
303         #         if not 'match' in key:
304         #             continue
305         #         TP = df_[key].value_counts()[True]
306         #         support = df_[key].count()
307         #         print(f'{key} - TPs: {TP}, support: {support}, accuracy: {TP/support}')
308         #     print('\n')
309 
310         # conditions_list = list(self.model.conditions.values())
311 
312         # for condition in [conditions_list,*conditions_list]:
313         #     print(f'Conditions: {condition}')
314         #     calc(df[df['gt_condition'].isin(condition if type(condition)==list else
        ↪  [condition])])
315 
316         return df
317 
318 def load_train_objs(args):
319 
320     #Dataset:
321     #If not a fiftyone dataset
322     if os.path.isdir(args.dataset):
323 
324         args.transforms.append(T.Resize([args.size,args.size]))
325         args.transforms = T.Compose(args.transforms)
326 
327         train_images = glob.glob(args.dataset + '/class_train/**/*')
328         val_images = glob.glob(args.dataset + '/class_val/**/*')
329 
330         with open(args.dataset + '/class_train/labels.json') as json_file:
331             train_annotations = json.load(json_file)
332 
333         with open(args.dataset + '/class_val/labels.json') as json_file:
334             val_annotations = json.load(json_file)
335 
336         labels = {}
337         labels[args.first_label] = train_annotations['classes']
338 
339         for x in train_annotations['labels']:
340             attributes = list(train_annotations['labels'][x]['attributes'].keys())
341             break
342 
343         for label in attributes:
344             labels[label] = []
345             for x in train_annotations['labels']:
346                 classe = train_annotations['labels'][x]['attributes'][label]
347                 if classe not in labels[label]:
348                     labels[label].append(classe)
```

```
349
350         train_set = folder_dataset(train_images,train_annotations,
            ↪  labels,args.size,transforms=args.transforms)
351
352         val_set = folder_dataset(val_images,val_annotations, labels, args.size)
353
354     else:
355
356         fo_dataset = fo.load_dataset(args.dataset)
357         fo_dataset.compute_metadata()
358
359         view =
            ↪  fo_dataset.select_labels(tags=['class_train','class_val'],fields='detections') #
            ↪  objetos selecionados no
360         classes = sorted(list(view.distinct('detections.detections.label')))
361         conditions = sorted(list(view.distinct('detections.detections.condition')))
362
363         train_set =
            ↪  fiftyone_dataset(view.select_labels(tags='class_train').to_patches('detections'),
364                                     classes,conditions,transforms=args.transforms)
365
366         val_set =
            ↪  fiftyone_dataset(view.select_labels(tags='class_val').to_patches('detections'),
367                                     classes,conditions,transforms=None)
368
369     #Model
370     if args.weights != '':
371         print(f'Loading pre-trained model {args.weights}')
372         model = torch.load(args.weights,map_location=torch.device('cpu'))
373     else:
374         print(labels)
375         model = TSMultiLabel(labels,transforms=val_set.transforms,model=args.model)
            ↪  #classificador() sequential torchvision.models.resnet18  # load your model
376
377     return train_set, val_set, model
378
379 def prepare_dataloader(dataset: Dataset, batch_size: int,ddp_mode: bool):
380     return DataLoader(
381         dataset,
382         batch_size=batch_size,
383         pin_memory=True,
384         shuffle= False if ddp_mode else True,
385         num_workers = 8,
386         sampler = DistributedSampler(dataset) if ddp_mode else None
387     )
388
389 def main(args,load_only=False):
390
391     tzinfo = timezone(timedelta(hours=-3.0))
392     date = datetime.now(tzinfo).strftime("%d_%m_%Y")
393     hour = datetime.now(tzinfo).strftime("%H_%M_%S")
394
```

```
395     if args.name != './runs':
396         args.name = (f'./runs/{date}/{args.name}/{hour}').replace('//','/')
397     else:
398         args.name = f'./runs/{date}/{hour}/'
399 
400     ddp_mode = os.getenv('LOCAL_RANK') != None
401 
402     if ddp_mode:
403         ddp_setup()
404 
405     train_set, val_set, model = load_train_objs(args)
406     args.labels = model.labels
407 
408     train_dataloader = prepare_dataloader(train_set, args.batch_size,ddp_mode)
409     val_dataloader = prepare_dataloader(val_set, args.batch_size,False)
410 
411     optimizer = torch.optim.Adam(model.parameters(),lr=args.lr)
412 
413     trainer = Trainer(model, train_dataloader, val_dataloader, optimizer, ddp_mode,args)
414 
415     if not load_only:
416         trainer.train(args.epochs)
417 
418         if ddp_mode:
419             destroy_process_group()
420 
421     return trainer
422 
423 def arg_parser(known=False):
424     import argparse
425     parser = argparse.ArgumentParser()
426     parser.add_argument('--dataset', type=str, default='datasets/CNNmulti_21_09_23',
    ↪  help='Name of fiftyone dataset or path of the root folder')
427     parser.add_argument('--epochs', type=int, default=100, help='Total epochs to train the
    ↪  model')
428     parser.add_argument('--batch-size', default=32, type=int, help='Input batch size on each
    ↪  device (default: 32)')
429     parser.add_argument('--name', type=str, default='./runs', help='save folder')
430     parser.add_argument('--lr', type=float, default=0.001, help='Total epochs to train the
    ↪  model')
431     parser.add_argument('--weights', type=str, default='', help='weights')
432     parser.add_argument('--size', type=int, default=224, help='Resize image if value != 0')
433     parser.add_argument('--first-label', type=str, default='kind', help='name of the first
    ↪  label attribute')
434     parser.add_argument('--model', type=str, default='resnet50', help='cnn backbone model')
435     parser.add_argument('--transforms', type=list, default=[
436                             T.ToTensor(),
437                             T.Normalize((0.5,0.5,0.5), (0.5,0.5,0.5))
438                         ], help='transforms')
439 
440     return parser.parse_known_args()[0] if known else parser.parse_args()
441 
```

```
442
443 def load_trainer(**kwargs):
444
445     args = arg_parser(True)
446
447     for k, v in kwargs.items():
448         setattr(args, k, v)
449     print(args)
450     trainer = main(args,load_only=True)
451
452     return trainer
453
454 if __name__ == "__main__":
455
456     args = arg_parser()
457
458     main(args)
```

Algoritmo 11: Código para definição da classe de *dataset* para CNN *multilabel*.

```
1  import torch
2  from torch.utils.data import Dataset
3  from PIL import Image
4  import os
5  import torchvision.transforms as T
6
7  class fiftyone_dataset(Dataset):
8
9      def __init__(self, view,classes,conditions,transforms,size):
10         #super(MyDataset, self).__init__()
11
12         self.view = view #must be a fiftyone patchesview
13
14         self.transforms = get_transform(transforms,size=64)
15
16         self.img_list = self.view.values("filepath")
17
18         self.classes = classes
19         self.conditions = conditions
20         self.ids = self.view.distinct('id')
21
22         self.map_label = {c: i for i, c in enumerate(self.classes)}
23         self.map_condition = {c: i for i, c in enumerate(self.conditions)}
24
25     def __getitem__(self, idx):
26
27         sample = self.view[self.ids[idx]]
28
29         W, H = sample.metadata.width, sample.metadata.height
30         xmin, ymin, w, h = sample.detections.bounding_box
31
```

```
        box_ = [xmin*W, ymin*H, (xmin+w)*W, (ymin+h)*H]

        img = Image.open(sample.filepath).crop(box_)

        label = self.map_label[sample.detections.label] if sample.detections.label in
        ↪  self.map_label.keys() else 'generic'

        condition = self.map_condition[sample.detections.condition]

        if self.transforms is not None:
            img = self.transforms(img)

        sample = {'image':img, 'labels': {'type':label, 'condition': condition}}

        return sample

    def __len__(self):
        return len(self.img_list)

class folder_dataset(Dataset):
    #This class works for fiftyone datasets exported as
    ↪  https://docs.voxel51.com/user_guide/export_datasets.html#fiftyoneimageclassificationdataset
    def __init__(self, img_list, annotations, labels,size,transforms=None):
        #super(folder_dataset, self).__init__()
        self.img_list = img_list
        self.labels = labels
        if transforms != None:
            self.transforms = transforms
        else:
            self.transforms = T.Compose([T.ToTensor(), T.Resize([size,size])])

        assert any('train' in file for file in img_list) or any('val' in file for file in
        ↪  img_list)

        self.annotations = annotations

    def __len__(self):
        return len(self.img_list)

    def __getitem__(self, idx):
        path = self.img_list[idx]
        filename = os.path.basename(path)[:-4]
        img = Image.open(path).convert("RGB")

        if self.transforms is not None:
            img = self.transforms(img)

        sample = {'image':img,'labels': {}}
        sample['labels'][list(self.labels.keys())[0]] =
        ↪  self.annotations['labels'][filename]['label']

        for label in list(self.labels.keys())[1:]:
```

```
            sample['labels'][label] =
            ↪  self.labels[label].index(self.annotations['labels'][filename]['attributes'][label])

        return sample

#aparentemente, collate fn serve para lidar com situações em que há varios bounding_boxes em
↪  uma imagem, por exemplo numa relação de 1:4
#sendo que o dataloader por padrão espera 1:1
def collate_fn(batch):
    image = [sample['image'] for sample in batch]
    target = [sample['target'] for sample in batch]

    return image, target
```

Algoritmo 12: Código para cálculo de medidas estatísticas para abordagem da Seção 3.6.2.

```
.

root = './runs/architecture_test2/'
datasets = os.listdir(root)


for dataset in datasets:
    if 'v2_' in dataset:
        reports = glob.glob(root + dataset + '/accuracies_report*.csv')
        reports.sort()
        if (len(reports) == 5):
            averaged_training_report = 0
            squared_diffs = 0
            max_cond = 0

            for report in reports:
                df = pd.read_csv(report, index_col=0)

                if 'new_veget' in dataset:
                    cond = 'vegetacao'
                    if max_cond < df['condition_match'].loc[cond]:
                        max_cond = df['condition_match'].loc[cond]
                        max_df = df
                        max_address = report

                elif 'old_veget' in dataset:
                    cond = 'vegetacao'
                    if max_cond < df['condition_match'].loc[cond]:
                        max_cond = df['condition_match'].loc[cond]
                        max_df = df
                        max_address = report

                elif 'graffiti' in dataset:
                    cond = 'pichacao'
                    if max_cond < df['condition_match'].loc[cond]:
                        max_cond = df['condition_match'].loc[cond]
```

```
                        max_df = df
                        max_address = report

                elif 'apagada' in dataset:
                    cond = 'apagada'
                    if max_cond < df['condition_match'].loc[cond]:
                        max_cond = df['condition_match'].loc[cond]
                        max_df = df
                        max_address = report

                elif 'orig' in dataset:
                    cond = 'all'
                    if max_cond < df['condition_match'].loc[cond]:
                        max_cond = df['condition_match'].loc[cond]
                        max_df = df
                        max_address = report

                # Somando valores dos DataFrames para média
                averaged_training_report += df

            # Dividindo para obter média dos valores dos DataFrames
            averaged_training_report /= len(reports)

            for report in reports:
                # Lendo novamente cada DataFrame
                df = pd.read_csv(report, index_col=0)

                # Calculando as diferenças entre o DataFrame e a média
                diff = df - averaged_training_report

                # Calculando os quadrados das diferenças
                squared_diffs += (diff ** 2)

            # Calculando o desvio padrão dos valores
            std_training_report = np.sqrt(squared_diffs / (len(reports)-1))

            # Calculando o coeficiente de variação percentual para cada valor
            cv_percentage = (std_training_report / averaged_training_report) * 100

            # Salvando o DataFrame da média em um arquivo CSV
            averaged_training_report.to_csv(root + dataset +
            ↪  '/averaged_training_report.csv')

            # Salvando o DataFrame do desvio padrão em um arquivo CSV
            std_training_report.to_csv(root + dataset + '/std_training_report.csv')

            # Salvando o DataFrame do coeficiente de variação percentual em um arquivo CSV
            cv_percentage.to_csv(root + dataset + '/cv_percentage_report.csv')

            max_df.to_csv(f'{root}{dataset}_max_{cond}.csv')


```

```
            print(f'O endereço do report para o dataset {dataset} com melhor acurácia para
            ↪  {cond}: \n{max_address}\nDataframe respectivo:')
            display(max_df)
            if not 'orig' in dataset:
                print(f'Comparativamente, a maior acurácia para a condição {cond} no dataset
                ↪  original foi {map[cond]}')
            print('\n\n')
```

Algoritmo 13: Código para geração de tabelas comparativas da Seção 4.4.

```
.

root = './runs/retraining_filters2/'
datasets = os.listdir(root)

for dataset in datasets:
    if 'new_veget' in dataset:
        df_name = "Method 2"
        cond = 'new_veget'

    elif 'old_veget' in dataset:
        df_name = "Method 1"
        cond = 'old_veget'

    elif 'graffiti' in dataset:
        df_name = "Method 4"
        cond = 'graffiti'

    elif 'apagada' in dataset:
        df_name = "Method 3"
        cond = 'fade'

    else:
        continue

    df = pd.read_csv('./runs/retraining_filters2/' + dataset +
    ↪  '/averaged_training_report.csv')


    conditions = ['All', 'Vegetation', 'Faded/Aged', 'Graffiti']
    df.insert(0, 'Condition', conditions)


    ref_df =
    ↪  pd.read_csv('./runs/retraining_filters2/v2_01-06-23_orig/averaged_training_report.csv')
    ref_df.insert(0, 'Condition', conditions)



    original_name = "Ref."  # Assign your desired name for Original here


```

```
    table_caption = "Comparação do resultado da média de acurácia sobre 5 treinamentos entre dataset aumentado via método " + df_name.replace('Method ', '') + ' e dataset original (Ref.)'
    table_label = "tab:" + cond


    latex_table = """
    \\begin{table}[]
    \\centering
    \\caption{%s}
    \\label{%s}
    \\begin{tabular}{@{}ccccccc@{}}
    \\toprule
                        & \\multicolumn{6}{c}{\\textbf{Mean accuracy over 5 trainings - %s}} \\\\ \\cmidrule(l){2-7}
                        & \\multicolumn{2}{c}{\\textbf{type\_match}} & \\multicolumn{2}{c}{\\textbf{condition\_match}} & \\multicolumn{2}{c}{\\textbf{all\_match}} \\\\ \\midrule
    \\textbf{Condition} & \\textbf{%s}      & \\textbf{%s}     & \\textbf{%s}        & \\textbf{%s}        & \\textbf{%s}     & \\textbf{%s}     \\\\ \\midrule
    """ % (table_caption, table_label, df_name, df_name, original_name, df_name, original_name, df_name, original_name)


    merged_df = df.merge(ref_df, on='Condition', suffixes=('_df', '_original'))


    num_rows = merged_df.shape[0]


    for i, (_, row) in enumerate(merged_df.iterrows(), 1):
        latex_table += "\\textbf{%s} & %.3f & %.3f & %.3f & %.3f & %.3f & %.3f \\\\\n" % (
            row['Condition'],
            row['type_match_df'], row['type_match_original'],
            row['condition_match_df'], row['condition_match_original'],
            row['all_match_df'], row['all_match_original']
        )


    latex_table += "\\bottomrule\n\\end{tabular}\n\\end{table}"

    print('--------------------------------')
    print(f'Tabela para dataset {dataset}')
    print(latex_table)
```

Referências Bibliográficas

ASHIQUZZAMAN, A.; TUSHAR, A. K.; RAHMAN, M. A. Applying data augmentation to handwritten arabic numeral recognition using deep learning neural networks. 8 2017. Disponível em: <http://arxiv.org/abs/1708.05969>. 3.6.4, 4.4.5

BAIRD, H. S. *Document Image Defect Models*. Springer Berlin Heidelberg, 1992. 546-556 p. ISBN 978-3-642-77281-8. Disponível em: <https://doi.org/10.1007/978-3-642-77281-8_26http://link.springer.com/10.1007/978-3-642-77281-8_26>. 3.1.6

BALALI, V.; GOLPARVAR-FARD, M. *Video-based detection and classification of US traffic signs and mile markers using color candidate extraction and feature-based recognition*. 2014. 858-866 p. 1

BJERRUM, E. J. Smiles enumeration as data augmentation for neural network modeling of molecules. 3 2017. Disponível em: <http://arxiv.org/abs/1703.07076>. 3.1.6

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. 4 2020. ISSN 2331-8422. Disponível em: <http://arxiv.org/abs/2004.10934>. 1

BRINK, H.; RICHARDS, J. W.; FETHEROLF, M.; CRONIN, B. *Real-World Machine Learning*. [S.l.: s.n.], 2017. ISBN 9781617291920. 3.6

CAO, J.; SONG, C.; PENG, S.; XIAO, F.; SONG, S. Improved traffic sign detection and recognition algorithm for intelligent vehicles. *Sensors (Switzerland)*, v. 19, 2019. ISSN 14248220. 1

CARDOSO, W. *Danificadas, placas de rua deixam motoristas e pedestres na mão em SP*. 2023. Disponível em: <https://www.metropoles.com/sao-paulo/placas-de-rua-estao-destruidas-enquanto-solucao-engatinha-em-sp>. 1

CORPORATION, C. *Computer Vision Annotation Tool (CVAT)*. 2022. Disponível em: <https://github.com/opencv/cvat>. 2.6, 2.7.3

DALBORGO, V. et al. Traffic sign recognition with deep learning: Vegetation occlusion detection in brazilian environments. *Sensors*, v. 23, 2023. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/23/13/5919>. 1, 3.2, 3.3

Departamento Nacional de Infraestrutura de Transportes (DNIT). 2023. <https://www.gov.br/dnit/pt-br/assuntos/noticias/diretoria-de-infraestrutura-rodoviaria-do-dnit-encerra-2021-com-diversas-entregas-pelo-pais>. Acessado em 23 de outubro de 2023. 1

GALDRAN, A. et al. Data-driven color augmentation techniques for deep skin image analysis. 3 2017. Disponível em: <http://arxiv.org/abs/1703.03702>. 3.1.6

GALVANI, M. History and future of driver assistance. *IEEE Instrumentation Measurement Magazine*, IEEE, v. 22, p. 11–16, 2 2019. ISSN 1094-6969. Disponível em: <https://ieeexplore.ieee.org/document/8633345/>. 1

GILROY, S.; JONES, E.; GLAVIN, M. Overcoming occlusion in the automotive environment - a review. *IEEE Transactions on Intelligent Transportation Systems*, Institute of Electrical and Electronics Engineers Inc., v. 22, p. 23–35, 1 2021. ISSN 15580016. 1

GIRSHICK, R. Fast r-cnn. 4 2015. Disponível em: <http://arxiv.org/abs/1504.08083>. 2.4

GIRSHICK, R.; DONAHUE, J.; DARRELL, T.; MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation. 11 2013. Disponível em: <http://arxiv.org/abs/1311.2524>. 2.4

GU, Y.; SI, B. A novel lightweight real-time traffic sign detection integration framework based on yolov4. *Entropy*, v. 24, 2022. ISSN 10994300. 1, 4.2.3

GéRON, A. *Mãos à obra: aprendizado de máquina com Scikit-Learn TensorFlow*. Alta Books, 2019. ISBN 9788550803814. Disponível em: <https://books.google.com.br/books?id=7Y7izQEACAAJ>. 2.1

HALEVY, A. Y.; NORVIG, P.; PEREIRA, F. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, v. 24, p. 8–12, 2009. Disponível em: <http://dblp.uni-trier.de/db/journals/expert/expert24.html#HalevyNP09>. 3.1.6

HAYKIN, S. S. *Neural networks and learning machines*. Third. [S.l.]: Pearson Education, 2009. 2.2, 2.2, 2.2, 2.3

HIRT, P. R.; HOLTKAMP, J.; HOEGNER, L.; XU, Y.; STILLA, U. Occlusion detection of traffic signs by voxel-based ray tracing using highly detailed models and mls point clouds of vegetation. *International Journal of Applied Earth Observation and Geoinformation*, Elsevier B.V., v. 114, p. 103017, 2022. ISSN 1872826X. Disponível em: <https://doi.org/10.1016/j.jag.2022.103017>. 3.1.1

HOU, Y. L.; HAO, X.; CHEN, H. A cognitively motivated method for classification of occluded traffic signs. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Institute of Electrical and Electronics Engineers Inc., v. 47, p. 255–262, 2 2017. ISSN 21682232. 1.4

HOUBEN, S.; STALLKAMP, J.; SALMEN, J.; SCHLIPSING, M.; IGEL, C. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. *Proceedings of the International Joint Conference on Neural Networks*, IEEE, 2013. 1, A

JOCHER, G. *YOLOv5 by Ultralytics*. 2020. Disponível em: <https://github.com/ultralytics/yolov5>. 2.1, 2.7.3, 3.2, 3.3, B.1

Jornal de Brasília. *Gastos com manutenção de placas de trânsito são de R*28*milporms*.2017.*Disponvelem* : <>. 1

JOSEPH, V. R. Optimal ratio for data splitting. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, Wiley, v. 15, p. 531–538, 8 2022. ISSN 1932-1864. Disponível em: <https://doi.org/10.1002%2Fsam.11583https://onlinelibrary.wiley.com/doi/10.1002/sam.11583>. 3.1.5

JUNGES, R. Z.; PAULA, M. de; AGUIAR, M. de. Brazilian traffic signs detection and recognition in videos using clahe, hog feature extraction and svm cascade classifier with temporal coherence. In: MARTíNEZ-VILLASEñOR, L.; BATYRSHIN, I.; MARíN-HERNáNDEZ, A. (Ed.). [S.l.]: Springer International Publishing, 2019. p. 589–600. ISBN 978-3-030-33749-0. 1.1

KHAN, S.; RAHMANI, H.; SHAH, S. A. A.; BENNAMOUN, M. *A Guide to Convolutional Neural Networks for Computer Vision*. Springer International Publishing, 2018. ISBN 978-3-031-00693-7 978-3-031-01821-3. Disponível em: <https://link.springer.com/10.1007/978-3-031-01821-3>. 2.1, 2.2, 2.2, 2.3

KIM, J. U.; KWON, J.; KIM, H. G.; LEE, H.; RO, Y. M. Object bounding box-critic networks for occlusion-robust object detection in road scene. In: . [S.l.]: IEEE Computer Society, 2018. p. 1313–1317. ISBN 9781479970612. ISSN 15224880. 1

KRAUSE, J.; STARK, M.; DENG, J.; FEI-FEI, L. 3d object representations for fine-grained categorization. In: *2013 IEEE International Conference on Computer Vision Workshops*. [S.l.: s.n.], 2013. p. 554–561. A

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. *ImageNet Classification with Deep Convolutional Neural Networks*. 2012. Disponível em: <http://code.google.com/p/cuda-convnet/>. 3.6.4, 4.4.5

LI, H.; SUN, F.; LIU, L.; WANG, L. A novel traffic sign detection method via color segmentation and robust shape matching. *Neurocomputing*, Elsevier, v. 169, p. 77–88, 12 2015. ISSN 18728286. 1

LIU, W. et al. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 9905 LNCS, p. 21–37, 2016. ISSN 16113349. 1

MAGNUSSEN, A. F.; LE, N.; HU, L.; WONG, W. E. A survey of the inadequacies in traffic sign recognition systems for autonomous vehicles. *International Journal of Performability Engineering*, v. 16, p. 1588, 2020. ISSN 0973-1318. Disponível em: <http://www.ijpe-online.com/EN/10.23940/ijpe.20.10.p10.15881597>. 1

MANNAN, A.; JAVED, K.; REHMAN, A. U.; BABRI, H. A.; NOON, S. K. Cognition based recognition of partially occluded traffic signs. *Scientia Iranica*, Sharif University of Technology, v. 29, p. 0–0, 2 2022. ISSN 2345-3605. Disponível em: <http://scientiairanica.sharif.edu/article_22679.html>. 1, 1.4, 4.2.3

MATHIAS, M.; TIMOFTE, R.; BENENSON, R.; GOOL, L. V. Traffic sign recognition x2014; how far are we from the solution? In: . IEEE, 2013. p. 1–8. ISBN 978-1-4673-6129-3. Disponível em: <http://ieeexplore.ieee.org/document/6707049/>. 1

MATHWORKS, T. *Introducing Deep Learning with MATLAB*. The MathWorks, Inc., 2021.

9 p. Disponível em: <https://www.mathworks.com/content/dam/mathworks/ebook/gated/80879v00_Deep_Learning_ebook.pdf?s_tid=wtest_ctent_HTML>. 2.4

MIKOŁAJCZYK, A.; GROCHOWSKI, M. Data augmentation for improving deep learning in image classification problem. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. p. 117–122. ISBN 9781538661437. 3.1.6

MOORE, B. E.; CORSO, J. J. *FiftyOne*. 2020. 2.7.1, 3.1.3

MOSTAFA, T.; CHOWDHURY, S. J.; RHAMAN, M. K.; ALAM, M. G. R. Occluded object detection for autonomous vehicles employing yolov5, yolox and faster r-cnn. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2022. p. 405–410. ISBN 9781665463164. 1

PADILLA, R.; NETTO, S. L.; SILVA, E. A. B. da. A survey on performance metrics for object-detection algorithms. In: . [S.l.: s.n.], 2020. p. 237–242. 3.5.2

PAL, A.; SELVAKUMAR, M.; SANKARASUBBU, M. Magnet: Multi-label text classification using attention-based graph neural network. *ICAART 2020 - Proceedings of the 12th International Conference on Agents and Artificial Intelligence*, v. 2, p. 494–505, 2020. 3.5.1

PASZKE, A. et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Curran Associates, Inc., 2019. 8024-8035 p. Disponível em: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>. 2.7.4

PEDREGOSA, F. et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. 3.5.1

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. v. 27, p. 306–308, 6 2015. Disponível em: <http://arxiv.org/abs/1506.02640>. 2.4, 2.5, 2.6, 2.5

REHMAN, Y.; RIAZ, I.; FAN, X.; SHIN, H. D-patches: effective traffic sign detection with occlusion handling. *IET Computer Vision*, Institution of Engineering and Technology, v. 11, p. 368–377, 8 2017. ISSN 1751-9640. Disponível em: <https://onlinelibrary.wiley.com/doi/10.1049/iet-cvi.2016.0303>. 1

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks. 2016. 2.4, 2.5

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. Faster r-cnn: Towards real-time object detection with

region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 39, p. 1137–1149, 2017. ISSN 01628828. 1

RUSSAKOVSKY, O. et al. Imagenet large scale visual recognition challenge. 9 2014. Disponível em: <http://arxiv.org/abs/1409.0575>. 4.4.5

SAADNA, Y.; BEHLOUL, A. An overview of traffic sign detection and classification methods. *International Journal of Multimedia Information Retrieval*, Springer London, v. 6, p. 193–210, 9 2017. ISSN 2192-6611. Disponível em: <http://link.springer.com/10.1007/s13735-017-0129-8>. 1

SILVA, I. N.; SPATTI, D. H.; FLAUZINO, R. *Redes neurais artificiais para engenharia e ciências aplicadas*. [S.l.]: Artliber Editora, 2010. 2.2, 2.2

SILVA, R. et al. *Construction of Brazilian Regulatory Traffic Sign Recognition Dataset*. [s.n.], 2021. 163-172 p. Disponível em: <https://link.springer.com/10.1007/978-3-030-93420-0_16>. 1, 1.1

SOUFI, N.; VALDENEGRO-TORO, M. Data augmentation with symbolic-to-real image translation gans for traffic sign recognition. 7 2019. Disponível em: <http://arxiv.org/abs/1907.12902>. 3.6.4, 4.4.5

STALLKAMP, J.; SCHLIPSING, M.; SALMEN, J.; IGEL, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, v. 32, p. 323–332, 8 2012. ISSN 08936080. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0893608012000457>. 1, A

STANIEK, M.; CZECH, P. Self-correcting neural network in road pavement diagnostics. *Automation in Construction*, Elsevier B.V., v. 96, p. 75–87, 12 2018. ISSN 09265805. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0926580518300220>. 1

STEVENS, E.; ANTIGA, L.; VIEHMANN, T. *Deep Learning with PyTorch*. 1. ed. [S.l.]: Manning Publications, 2020. ISBN 1617295264,9781617295263. 2.1, 2.7.4

TIMOFTE, R.; ZIMMERMANN, K.; GOOL, L. V. Multi-view traffic sign detection, recognition, and 3d localisation. *Machine Vision and Applications*, v. 25, p. 633–647, 2014. ISSN 14321769. 1, 4.2.1

TKACHENKO, M.; MALYUK, M.; HOLMANYUK, A.; LIUBIMOV, N. *Label Studio: Data labeling software*. 2023. Open source software available from https://github.com/heartexlabs/label-

studio. Disponível em: <https://github.com/heartexlabs/label-studio>. 2.6

TRIKI, N.; KARRAY, M.; KSANTINI, M. A real-time traffic sign recognition method using a new attention-based deep convolutional neural network for smart vehicles. *Applied Sciences*, v. 13, p. 4793, 4 2023. ISSN 2076-3417. Disponível em: <https://www.mdpi.com/2076-3417/13/8/4793>. 1

VASCONCELOS, C. N.; VASCONCELOS, B. N. Increasing deep learning melanoma classification by classical and expert knowledge based image transforms. *ArXiv*, abs/1702.07025, 2017. Disponível em: <https://api.semanticscholar.org/CorpusID:15545473>. 3.1.6

WADA, K. *Labelme: Image Polygonal Annotation with Python*. 2023. Disponível em: <https://github.com/wkentaro/labelme>. 2.6

WALI, S. B. et al. Vision-based traffic sign detection and recognition systems: Current trends and challenges. *Sensors*, MDPI AG, v. 19, p. 2093, 5 2019. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/19/9/2093>. 1

WANG, B.; KLABJAN, D. Regularization for unsupervised deep neural nets. 8 2016. Disponível em: <http://arxiv.org/abs/1608.04426>. 3.1.6

WONG, S. C.; GATT, A.; STAMATESCU, V.; MCDONNELL, M. D. Understanding data augmentation for classification: when to warp? 9 2016. Disponível em: <http://arxiv.org/abs/1609.08764>. 3.1.6

WU, R.; YAN, S.; SHAN, Y.; DANG, Q.; SUN, G. Deep image: Scaling up image recognition. 1 2015. Disponível em: <http://arxiv.org/abs/1501.02876>. 3.6.4, 4.4.5

WU, X. et al. A survey of human-in-the-loop for machine learning. *Future Generation Computer Systems*, Elsevier {BV}, v. 135, p. 364–381, 10 2022. Disponível em: <https://doi.org/10.1016%2Fj.future.2022.05.014>. 3.1

XU, Y. et al. Improved relation classification by deep recurrent neural networks with data augmentation. 1 2016. Disponível em: <http://arxiv.org/abs/1601.03651>. 3.1.6

YING, X. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, Institute of Physics Publishing, v. 1168, p. 022022, 2 2019. ISSN 1742-6588. Disponível em: <https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022>. 3.1.6

YOUSSEF, A.; ALBANI, D.; NARDI, D.; BLOISI, D. D. *Advanced Concepts for Intelligent*

Vision Systems. Springer International Publishing, 2016. v. 10016. ISBN 978-3-319-48679-6. Disponível em: <http://link.springer.com/10.1007/978-3-319-48680-2>. 1

ZHANG, J.; XIE, Z.; SUN, J.; ZOU, X.; WANG, J. A cascaded r-cnn with multiscale attention and imbalanced samples for traffic sign detection. *IEEE Access*, Institute of Electrical and Electronics Engineers Inc., v. 8, p. 29742–29754, 2020. ISSN 21693536. 3.6.4, 4.4.5

ZHOU, K.; ZHAN, Y.; FU, D. Learning region-based attention network for traffic sign recognition. *Sensors (Switzerland)*, v. 21, p. 1–21, 2021. ISSN 14248220. 1, 4.2.3

ZHU, Y.; YAN, W. Q. Traffic sign recognition based on deep learning. *Multimedia Tools and Applications*, Multimedia Tools and Applications, v. 81, p. 17779–17791, 2022. ISSN 15737721. 1

ZHU, Z. et al. Traffic-sign detection and classification in the wild. In: . [S.l.]: IEEE Computer Society, 2016. v. 2016-Decem, p. 2110–2118. ISBN 9781467388504. ISSN 10636919. 1