Centro de Tecnologia e Urbanismo

Departamento de Engenharia Elétrica

Marcelo Theis Geraldi

Detecção e rastreamento de peixes por meio de redes neurais convolucionais para aplicação em dispositivos embarcados

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

Londrina, PR
2021

Marcelo Theis Geraldi

Detecção e rastreamento de peixes por meio de redes neurais convolucionais para aplicação em dispositivos embarcados

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

Área de concentração: Sistemas Eletrônicos
Especialidade: Instrumentação Eletrônica

Orientadora:
Prof. Dra. Maria Bernadete de Morais França

Londrina, PR
2021

Ficha Catalográfica

Theis Geraldi, Marcelo

Detecção e rastreamento de peixes por meio de redes neurais convolucionais para aplicação em dispositivos embarcados. Londrina, PR, 2021. 100 p.

Dissertação (Mestrado) – Universidade Estadual de Londrina, PR. Departamento de Engenharia Elétrica .

1. Inteligência Artificial. 2. Aprendizagem de Máquina. 3. Visão Computacional I. Universidade Estadual de Londrina. Departamento de Engenharia Elétrica. Departamento de Engenharia Elétrica . II. Título.

Marcelo Theis Geraldi

Detecção e rastreamento de peixes por meio de redes neurais convolucionais para aplicação em dispositivos embarcados

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

Área de concentração: Sistemas Eletrônicos
Especialidade: Instrumentação Eletrônica

Comissão Examinadora

Prof. Dra. Maria Bernadete de Morais França
Depto. de Engenharia Elétrica
Universidade Estadual de Londrina
Orientadora

Prof. Dr. José Alexandre de França
Depto. de Engenharia Elétrica
Universidade Estadual de Londrina

Prof. Dr. Sylvio Barbon Junior
Depto. de Computação
Universidade Estadual de Londrina

Prof. Dr. Rodrigo Moreira Bacurau
Depto. de Mecânica Computacional
Universidade Estadual de Campinas

27 de abril de 2021

Agradecimentos

Agradeço primeiramente a Deus que me deu forças, abençoou e me guiou, nos bons e nos maus momentos, sempre me iluminando, trazendo graças e fazendo eu persistir, para que meus sonhos pudessem ser alcançados.

Agradeço a toda minha família: meus queridos pais Vera e Adenis; meu estimado irmão e parceiro Ricardo; e minha tão amada e companheira Angélica. Estes são em minha vida meu pilar principal, sempre me auxiliando, apoiando e estando comigo em todos os momentos, fazendo acreditar que tudo é possível.

Agradeço também aos professores e especialmente a minha orientadora Prof. Dra. Maria Bernadete de Morais França, que sempre se manteve otimista e alegre, confiando em minha capacidade e me concedendo a oportunidade de desenvolver este trabalho. Também agradeço a instituição pelo fornecimento de toda a sua estrutura, principalmente aos laboratórios de Automação e Instrumentação (LA2I) e Ecofisiologia Animal (LEFA).

Aos colegas de laboratório Andressa, Giuliano, Gustavo, João, Rafael e Sérgio meus singelos e sinceros agradecimentos pois muito me ajudaram nesta jornada, estando sempre disponíveis para auxiliar nas mais diversas questões e tornando o caminho mais descontraído, divertido e prazeroso.

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

Marcelo Theis Geraldi. Detecção e rastreamento de peixes por meio de redes neurais convolucionais para aplicação em dispositivos embarcados. 2021. 100 p. Dissertação do programa de Mestrado em Engenharia Elétrica - Universidade Estadual de Londrina, Londrina.

Resumo

O ambiente aquático e a saúde humana são frequentemente afetados por substâncias presentes nos efluentes agrícolas, industriais e urbanos. O comportamento dos peixes pode ser afetado por várias dessas substâncias, sendo utilizado, portanto, como um indicador da presença de xenobióticos na água. Identificação e monitoramento são dificuldades existentes nos processos de detecção de substâncias e medição da qualidade da água. Técnicas aplicadas para monitorar o comportamento dos peixes podem ser um meio para auxiliar nessa tarefa. O objetivo deste trabalho é desenvolver um *framework* voltado a dispositivos embarcados para a detecção e rastreamento de peixes utilizando técnicas de aprendizagem de máquina, visando auxiliar pesquisadores da área de etologia e ecotoxicologia. Para tanto, utilizou-se o conjunto de imagens e anotações da classe "peixe (*fish*)" presente no *dataset* Open Images Dataset V4. O *framework* desenvolvido foi aplicado em um computador pessoal (*laptop*) e no dispositivo embarcado NVIDIA Jetson Nano, sendo realizados testes e comparações entre dispositivos. Devido as limitações de hardware presentes em um dispositivo embarcado, adotou-se para o processo de detecção as redes neurais convolucionais YOLOv3, YOLOv3-tiny e MobileNetV2 e, para o processo de rastreamento, os algoritmos baseados em filtros correlacionais dlib, CSRT, KCF e MOSSE. Testes entre os detectores e rastreadores foram realizados, atingindo resultados de 73,32% de *Average Precision* (AP) para a MobileNetV2, 31,28% de *Multiple Object Tracker Accuracy* (MOTA) para o CSRT e 75,23% de *Multiple Object Tracker Precision* (MOTP) para o KCF. A MobileNetV2 apresentou melhores resultados dentre as redes. Apesar do CSRT e KCF indicarem melhores valores de precisão e acurácia, o dlib apresentou resultados competitivos (30,60% de MOTA e 73,88% de MOTP) com desempenho e consumo energético superiores. Após diversos testes, considerou-se a MobileNetV2 e o dlib como as melhores opções de detecção e rastreamento respectivamente, apresentando boas trocas entre precisão, acurácia, desempenho e consumo energético. O *framework* desenvolvido realizou a detecção e rastreamento dos peixes, porém, falhas foram notadas em ambos os processos, apontando que aperfeiçoamentos ainda são necessários para alcançar melhores resultados.

Palavras-chaves: *1. Qualidade da Água; 2. Análises Comportamentais de Peixes; 3. Aprendizagem Profunda; 4. Rede Neural Convolucional; 5. Detecção e Rastreamento de Objetos.*

Marcelo Theis Geraldi. Fish detection and tracking using convolutional neural networks for application in embedded devices. 2021. 100 p. Master's thesis in Electrical Engineering - State University of Londrina, Londrina.

Abstract

The aquatic environment and human health are often harmed due to the presence of chemical waste incorrectly dumped in water by agricultural, industrial and urban effluents. Fish behavior can be affected by several substances and it can be used as an indicator for the presence of xenobiotics in water. Identification and monitoring are difficulties in the process of detecting substances and measuring water quality. Techniques applied to monitor fish behavior can be a way to assist the water problems. The objective of this work is to develop a framework for the detection and tracking of fish using machine learning algorithms for embedded devices, supporting researchers in behavioral analysis and ecotoxicology. For the development of this work, we used the set of images and annotations of the class "fish" present in dataset Open Images Dataset V4. The *framework* was applied on a personal computer (laptop) and on the NVIDIA Jetson Nano embedded device. Tests and comparisons between devices were performed. Due to the hardware limitations present in an embedded device, the convolutional neural networks YOLOv3, YOLOv3-tiny and MobileNetV2 were adopted for the detection process and for the tracking process, the algorithms based on correlational filters dlib, CSRT, KCF and MOSSE is used. Tests between the detectors and trackers were performed, achieving 73.32% of Average Precision (AP) for MobileNetV2, 31.28% of Multiple Object Tracker Accuracy (MOTA) for CSRT, and 75.23% of Multiple Object Tracker Precision (MOTP) for the KCF. MobileNetV2 showed better results among the networks. Although CSRT and KCF indicated better precision and accuracy values, dlib showed competitive results (30.60% for MOTA and 73.88% for MOTP) with superior performance and energy consumption. After several tests, MobileNetV2 and dlib were considered as the best options for detection and tracking respectively, showing good tradeoffs between precision, accuracy, performance and energy consumption. The framework performed fish detection and tracking, however, flaws were noted in both processes, indicating that improvements are still needed to achieve better results.

Keywords: *1. Water Quality; 2. Fishes Behavioral Analysis; 3. Deep Learning; 4. Convolutional Neural Network; 5. Object Detection and Tracking.*

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ANA Agência Nacional de Águas

ANN *Artificial Neural Network*

API *Application Programming Interface*

ARS *Agricultural Research Service*

CF *Correlation Filter*

CLEAR *CLassification of Events, Activities and Relationships*

CNN *Convolutional Neural Network*

CONAMA Conselho Nacional do Meio Ambiente

CPU *Central Processing Unit*

CSI *Camera Serial Interface*

CSV *Comma Separated Values*

CUDA *Compute Unified Device Architecture*

DL *Deep Learning*

EPA *Environmental Protection Agency*

FM *FragMentations*

FN *False Negative*

FP *False Positive*

FPS *Frames Per Second*

GT *Ground Truth*

GPU *Graphics Processing Unit*

HOG *Histogram Of Oriented Gradients*

IA Inteligência Artificial

ID Identificação

IDA *IDentity Ascend*

IDF1 *IDentity F1*

IDM *IDentity Migrate*

IDP *IDentity Precision*

IDR *IDentity Recall*

IDS *IDentity Switch*

IDT *IDentity Transfer*

IET Índice de Estado Trófico

IPCA Índice de Parâmetros para proteção das Comunidades Aquáticas

IPMCA Índice de Parâmetros Mínimos para proteção das Comunidades Aquáticas

IQA Índice de Qualidade da Água

IVA Índice de proteção da Vida Aquática

KNN *K Nearest Neighbors*

ML *Mostly Loss*

MLE *Machine LEarning*

MLP *Multilayer Perceptron*

MOT *Multiple Object Tracking*

MOTA *Multiple Object Tracking Accuracy*

MOTP *Multiple Object Tracking Precision*

MT *Mostly Tracked*

NN *Neural Network*

NWIS *National Water Information System*

NWQMC *National Water Quality Monitoring Council*

OID *Open Images Dataset*

PAPI *Performance Application Programming Interface*

PID *Process IDentifier*

pH potencial Hidrogeniônico

PRCN *PReCisioN*

PT *Partially Tracked*

RAM *Random Access Memory*

RCLL *ReCaLL*

SNIRH Sistema Nacional de Informações sobre Recursos Hídricos

SSD *Single Shot Detector*

STEWARDS *Sustaining The Earth's Watersheds - Agricultural Research Database System*

STORET *STOrage and RETrieval*

SVM *Support Vector Machine*

TBD *Tracking-By-Detection*

TPU *Tensor Processing Unit*

USB *Universal Serial Bus*

USDA *United States Department of Agriculture*

USGS *United States Geological Survey*

WQI *Water Quality Index*

WQP *Water Quality Portal*

1 Introdução

A água cobre aproximadamente 71% da superfície da terra, no qual 97% é água salgada e 3% água doce. Dentre esses 3%, apenas 0,5% estão disponíveis para consumo e o restante está confinado em geleiras, camadas polares, no solo, entre outros (RECLAMATION, 2019). Além de somente pequena parte da água estar disponível para consumo, uma parcela importante dessas águas torna-se imprópria devido à poluição dos corpos d'água por substâncias tóxicas, ocasionada pelos altos níveis de crescimento demográfico e pelas diversas atividades econômicas, prejudicando à saúde humana e os ecossistemas aquáticos.

O Brasil é detentor de 12% da oferta de água doce acessível do planeta, sendo 4% desta considerada de ótima qualidade (AGÊNCIA NACIONAL DE ÁGUAS, 2015). Estudos como os de Dellamatrice e Monteiro (2014), Costa et al. (2019) e relatórios da Agência Nacional de Águas (2019) apontam que a qualidade da água, mensurada principalmente por meio de índices como o Índice de Qualidade da Água (IQA) e o Índice de proteção da Vida Aquática (IVA), é crítica em diversas localidades do país, demonstrando que ações devem ser tomadas a fim de sanar este problema.

Todavia, o problema da qualidade da água não se limita ao Brasil. Estudos como os de Schwarzenbach et al. (2010), Bain et al. (2014), Alves, Teresa e Nabout (2014) e P.U. et al. (2017) apontam que outros locais do planeta, principalmente os países emergentes (devido a pobreza e condições de vida precárias, por exemplo), sofrem do mesmo problema, sendo também gerado por fatores antropogênicos. Legislações como a Lei Nº 9.433 (Lei das Águas) (MINISTÉRIO DO MEIO AMBIENTE, 1997) e a do Conselho Nacional do Meio Ambiente (CONAMA) Nº 357/2005 (CONSELHO NACIONAL DO MEIO AMBIENTE, 2005) são aplicadas visando regular o problema, porém, suas obrigatoriedades não garantem com total eficácia na resolução do problema.

Para tanto, dados os avanços tecnológicos dos últimos anos, sistemas foram desenvolvidos buscando solucionar as questões relacionadas a qualidade da água. No Brasil, o Sistema Nacional de Informações sobre Recursos Hídricos (SNIRH)

(SNIRH, 1997) gerido pela Agência Nacional de Águas (ANA) e disponível gratuitamente para toda a sociedade, coleta e armazena informações referentes aos recursos hídricos (temperatura, turbidez, pH, são alguns dos exemplos) por meio da aplicação de tecnologias como: sensores ligados a uma plataforma coletora de dados, telemetria, satélites, entre outros.

Nos Estados Unidos, estratégias e tecnologias similares ao do SNIRH são utilizadas para a coleta de dados. O serviço cooperativo *Water Quality Portal* (WQP) (NWQMC; USGS; EPA, 1997) patrocinado pela *United States Geological Survey* (USGS), *Environmental Protection Agency* (EPA) e o *National Water Quality Monitoring Council* (NWQMC), fornece de maneira pública e gratuita dados relacionados a qualidade da água a partir de outros sistemas: USGS *National Water Information System* (NWIS); o EPA *STOrage and RETrieval* (STORET) *Data Warehouse*; e o *United States Department of Agriculture* (USDA) *Agricultural Research Service* (ARS) *Sustaining The Earth's Watersheds - Agricultural Research Database System* (STEWARDS).

Portanto, fica evidente que investimentos em ações tecnológicas pelos países vêm sendo realizados e aplicados para monitorar continuamente a qualidade da água e, assim, aplicar soluções para melhorá-la. Atualmente, com o advento da inteligência artificial, mais precisamente da aprendizagem de máquina (permitindo que máquinas aprendam padrões a partir de dados) e visão computacional (habilitando as máquinas para extrair dados de imagens ou dados multidimensionais, fazendo com que elas "enxerguem"), um novo e poderoso leque de possibilidades foi aberto, permitindo que os recursos hídricos fossem monitorados e controlados de outras formas.

Além da clássica aplicação de sensores, as análises comportamentais de organismos aquáticos se mostraram como uma nova ferramenta alternativa para avaliação da qualidade da água. Para isto, peixes como *Danio rerio* ("*zebrafish*" ou paulistinha) estão sendo utilizados frequentemente devido às suas diversas respostas comportamentais (esquiva, aprendizado, estresse, dentre outras) às substâncias tóxicas presentes na água, sendo considerado, assim, um modelo experimental robusto e complexo (KUKLINA; KOUBA; KOZÁK, 2013; RESENDE; SIEBEL; BONAN, 2015). Contudo, similar aos demais processos, a análise comportamental de peixes é passível de automatização de seus procedimentos.

Tradicionalmente a realização de análises comportamentais dos peixes consiste na observação visual direta ou de vídeos gravados de seus movimentos, a partir da qual o pesquisador extraía manualmente dados como tempo (imóvel

ou em movimentação), velocidade de movimento, deslocamento, entre outros. Porém, realizar estas tarefas, é uma tarefa desgastante e árdua. Portanto, recursos tecnológicos como câmeras e softwares foram desenvolvidos e aplicados no decorrer dos anos, visando aprimorar estes experimentos (agregando maior precisão e permitindo obter um maior volume de dados e fazer análises mais complexas).

Buscando maiores níveis de precisão, praticidade, facilidade e automatização dos processos, pesquisas envolvendo aprendizagem de máquina e visão computacional nesta área estão sendo desenvolvidas. Trabalhos como os de Xu e Cheng (2017), Meng, Hirayama e Oyanagi (2018) e Xu e Matzner (2018) mostram que redes neurais profundas (que "simulam" e são inspiradas no funcionamento do cérebro) possuem um enorme potencial, próximo ao de um ser humano, sendo consideradas uma ferramenta poderosa.

Em Xu e Cheng (2017) uma rede neural convolucional foi proposta, treinada e utilizada no rastreamento de múltiplos peixes da espécie *zebrafish*. Foi proposto também um método de rastreamento a partir da região da cabeça do peixe, devido à sua estabilidade comparado aos movimentos do restante do corpo. Um pequeno *dataset* foi criado com imagens coletadas de cabeça de peixes *zebrafish* e foi realizado o processo de aumento de dados (do inglês, *data augmentation*) para ampliar a quantidade de imagens e melhorar os resultados do treinamento da rede. O sistema desenvolvido teve uma resposta satisfatória no quesito de identificação única de peixes e manuseamento de oclusões, que são considerados os principais problemas em sistemas de rastreamento. Além disso, o sistema apresentou resultados precisos nos diversos vídeos de teste utilizados. O algoritmo e treinamento da rede foram desenvolvidos e aplicados no software MATLAB.

Meng, Hirayama e Oyanagi (2018) utilizaram um drone subaquático com uma câmera panorâmica de 360° acoplada à um Raspberry Pi e redes neurais convolucionais para realizar a detecção de peixes, visando investigar espécies e proteger o ambiente natural. Para o processo de detecção, as seguintes redes foram utilizadas: LeNet (LECUN et al., 1998), AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) e GoogLeNet (SZEGEDY et al., 2015). O *dataset* foi gerado a partir de 100 imagens de 4 espécies de peixes e, utilizando o processo de *data augmentation*, este número de imagens ampliou para 86.400. O Raspberry Pi foi utilizado para efetuar o processamento das imagens obtidas da câmera panorâmica e realizar o *streaming* destas em tempo real. O processo de treinamento e detecção foi realizado em um computador pessoal e alcançou uma acurácia de detecção de 87%.

O trabalho de Xu e Matzner (2018) aplicou a rede neural convolucional YOLOv3 (REDMON; FARHADI, 2018) para detecção dos peixes, objetivando um monitoramento automatizado e analisar os impactos das estruturas de obtenção de energia advindas das marés (energia maremotriz) na vida aquática dos peixes e outros animais marinhos. O *dataset* para o treinamento foi obtido a partir dos *frames* de três vídeos, resultando em 54.516 imagens. O algoritmo de treinamento e detecção foi desenvolvido utilizando a linguagem Python. Entretanto, a acurácia não foi considerada desejável e, de acordo com os autores, a baixa qualidade das imagens (*frames* dos vídeos) e os fatores presentes no ambiente subaquático (baixa luz, turbidez, alto fluxo de água, dentre outros), influenciaram negativamente os resultados.

Há uma escassez de trabalhos encontrados na literatura referentes a análise comportamental de peixes aplicando aprendizagem de máquina e visão computacional, explorando a possibilidade da utilização de redes neurais convolucionais e algoritmos de rastreamento em uma única solução e, também, que sejam voltados para a aplicação em dispositivos embarcados.

Neste sentido, o objetivo geral deste trabalho é propor um *framework* utilizando aprendizagem de máquina e visão computacional para a detecção e rastreamento de peixes, a fim de auxiliar pesquisadores da área nos processos intrínsecos às análises comportamentais. Conforme o objetivo geral definido, os seguintes objetivos específicos são elencados:

- Desenvolver um *dataset* de vídeos manualmente anotados para avaliar o processo de rastreamento de peixes;
- Avaliar a viabilidade da aplicação em um sistema embarcado e em computador pessoal, provendo *benchmarks* para os processos de detecção e rastreamento de peixes;
- Analisar e investigar a capacidade de redes neurais convolucionais profundas e filtros correlacionais para a tarefa de detecção e rastreamento de peixes respectivamente.

Neste trabalho são exploradas as dificuldades intrínsecas da automatização do processo de análise comportamental de peixes envolvendo aprendizagem de máquina e visão computacional. As principais contribuições providas por este trabalho consistem em:

- Desenvolvimento de um *framework open-source* de detecção e rastreamento de peixes para aplicação em um dispositivo embarcado;
- Produção de *benchmark* da aplicação de redes neurais convolucionais e filtros correlacionais no processo de detecção e rastreamento de peixes;
- Desenvolvimento de um *dataset open-source* anotado manualmente para avaliação de processos de rastreamento de peixes;

Deste modo, no Capítulo 2, são apresentados os conceitos referentes a água, envolvendo suas métricas e técnicas avaliativas qualitativas. Além disso, nesse mesmo capítulo é apresentada uma introdução a visão computacional, redes neurais e filtros correlacionais. No Capítulo 3 é descrita a metodologia aplicada para o desenvolvimento do trabalho, abrangendo os aparatos, técnicas e algoritmos utilizados. No Capítulo 4 são apresentados os resultados obtidos. E, por fim, no Capítulo 5, são feitas as considerações finais, apontando possíveis trabalhos futuros.

2 Fundamentação teórica

2.1 Água e técnicas avaliativas qualitativas

Substância natural, esgotável e fundamental para todos os seres vivos, a água, deve ser preservada para garantir sua qualidade e, consequentemente, a sobrevivência dos organismos. Há alguns anos, as substâncias tóxicas são a grande ameaça ao ecossistema aquático, provocando a contaminação generalizada dos corpos d'água. A deterioração deste recurso está ligada principalmente às atividades humanas (agrícolas, industriais, entre outras) (ALVES; TERESA; NABOUT, 2014). Agrotóxicos, metais (chumbo, zinco, cobre, entre outros) e demais substâncias acarretam em adulterações das propriedades da água, impactando em sua qualidade, inapropriando-a ao consumo e, assim, ameaçando a saúde humana e os seres aquáticos (DELLAMATRICE; MONTEIRO, 2014; OZAKI et al., 2019).

Para mensurar a qualidade da água Brown et al. (1970) propuseram o WQI (*Water Quality Index*), um índice que abrange 9 variáveis mais relevantes para definir a qualidade da água. São levados em consideração o oxigênio dissolvido, coliformes termotolerantes, o nível de pH (potencial hidrogeniônico), demanda bioquímica de oxigênio, temperatura da água, nitrogênio total, fósforo total, turbidez e resíduo total. Este índice tem como objetivo a padronização do que é considerada uma água de qualidade. Curvas de qualidade foram traçadas para cada variável e são correlacionadas suas concentrações com uma nota, além disso, cada variável possui um peso relativo (w_i). Para o cálculo do WQI, utiliza-se a equação a seguir:

$$WQI = \prod_{i=1}^{n} {q_i}^{w_i}, \tag{2.1}$$

sendo:

- WQI: *Water Quality Index*, índice resultante da qualidade da água calculada. Um número entre 0 e 100;

- n: número de parâmetros que entram no cálculo do WQI;
- q_i: qualidade do i-ésimo parâmetro. Um número entre 0 e 100, advindo do respectivo gráfico de qualidade em função de sua concentração ou medida (resultado da análise da água);
- w_i: peso que corresponde ao i-ésimo parâmetro fixado em função da sua importância para a conformação global da qualidade. Um número que se mantém na faixa entre 0 e 1.

Por meio do resultado extraído após aplicar a fórmula nos dados obtidos da análise da água, utiliza-se a Tabela 2.1 de referência para classificar a qualidade da água.

Tabela 2.1: Classificação em nome e cor da qualidade da água resultante do cálculo do índice WQI.

Classificação	WQI	Cores
Excelente	$90 \leq \text{WQI} \leq 100$	Azul
Boa	$70 \leq \text{WQI} < 90$	Verde
Média	$50 \leq \text{WQI} < 70$	Amarelo
Ruim	$25 \leq \text{WQI} < 50$	Laranja
Muito ruim	$0 \leq \text{WQI} < 25$	Vermelho

Fonte: Adaptado de Instituto Ambiental do Paraná (2017)

O WQI foi adotado posteriormente pelos diversos órgãos em vários países do mundo, realizando apenas pequenas adaptações para a realidade local, devido às diferenças existentes nos variados ecossistemas. No Brasil adaptou-se o WQI para IQA (Índice de Qualidade da Água), além de existirem variações do mesmo adotadas em alguns estados e para cada tipo de medição (AGÊNCIA NACIONAL DE ÁGUAS, 2015; INSTITUTO AMBIENTAL DO PARANÁ, 2017).

Contudo, apesar do IQA demonstrar os níveis de qualidade e poluição da água, o mesmo apresenta limitações e não contempla a presença de substâncias tóxicas para organismos aquáticos, por exemplo. Portanto, foi desenvolvido por Zagatto et al. (1999) o IVA (Índice de proteção da Vida Aquática).

O IVA engloba os resultados das análises químicas, de clorofila e testes de toxicidade, no qual é possível estabelecer diferentes classes de qualidade de água. O resultado do IVA é dado pela composição de dois índices: IPMCA (Índice de Parâmetros Mínimos para proteção das Comunidades Aquáticas) e IET (Índice de Estado Trófico).

O IPMCA (COMPANHIA AMBIENTAL DO ESTADO DE SÃO PAULO, 1998), anteriormente denominado de IPCA (Índice de Parâmetros para proteção das Comu-

nidades Aquáticas) e desenvolvido por Zagatto et al. (1998), contempla a concentração de substâncias tóxicas (ST) - metais pesados e surfactantes - e parâmetros essenciais (PE) - oxigênio dissolvido, nível de pH e toxicidade - presentes nos corpos d'água. Cada parâmetro possui ponderações numéricas e limites definidos pelas legislações francesa (PERMANENT, 1973) e americana (USEPA, 1991) e pela resolução do Conselho Nacional do Meio Ambiente (2005).

O IET, desenvolvido por (CARLSON, 1977) e modificado por (TOLEDO et al., 1983), (TOLEDO, 1990) e (LAMPARELLI, 2004), tem como objetivo classificar os corpos d'água em diferentes graus de trofia, no qual são consideradas as elevações das concentrações de nutrientes, especialmente fósforo e clorofila, resultando no aumento da produtividade do ecossistema aquático.

Considerados o IPMCA e o IET, define-se o IVA é por meio da equação:

$$IVA = (IPMCA * 1,2) + IET. \tag{2.2}$$

Por meio dos resultados advindos dos cálculo dos índices IPMCA e IET, utiliza-se a Tabela 2.2 de referência para classificar a qualidade do ambiente aquático.

Tabela 2.2: Classificação em nome e cor da qualidade do ambiente aquático resultante do cálculo do índice IVA.

Classificação	IVA	Cores
Ótima	IVA $\leq$ 2,5	Azul
Boa	2,6 $\leq$ IVA $\leq$ 3,3	Verde
Regular	3,4 $\leq$ IVA $\leq$ 4,5	Amarelo
Ruim	4,6 $\leq$ IVA $\leq$ 6,7	Vermelho
Péssima	6,8 $\leq$ IVA	Lilás

Fonte: Adaptado de Zagatto et al. (1999)

A contaminação da água, muitas vezes, pode ser perceptível observando-a, porém, a utilização de dispositivos garante uma análise avançada e detalhada, permitindo determinar com precisão a temperatura, turbidez e valor do pH, por exemplo. Inúmeras são as consequências causadas pela contaminação, dentre elas: a mortandade de organismos; aumento ou diminuição de algas; e modificações biológicas e comportamentais em animais (ALVES; TERESA; NABOUT, 2014).

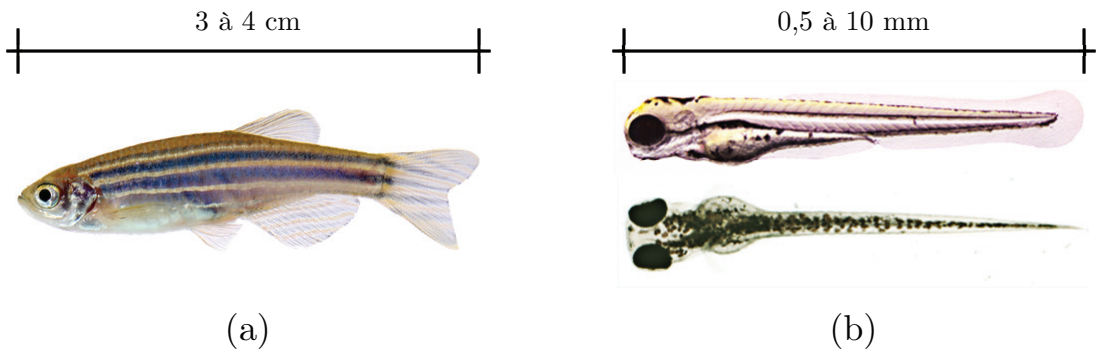
As análises comportamentais atualmente são uma das técnicas utilizadas por pesquisadores para a verificação de substâncias presentes na água. A ciência que realiza o estudo do comportamento animal em seu habitat natural é a Etologia. O estudo e análise de modelos animais alternativos, como é o caso de algumas espécies de peixes, permitem uma melhor compreensão de mecanismos neurobiológicos (estudo das células do sistema nervoso, envolvendo aspectos mor-

fológicos e organizacionais) e neuroquímicos (estudo dos processos químicos do sistema nervoso), que, por sua vez, auxiliam na triagem e desenvolvimento de novos medicamentos e conhecimento aprofundado de doenças (RESENDE; SIEBEL; BONAN, 2015).

Existem inúmeros compostos presentes na água e a utilização de dispositivos como sensores podem não verificar ou responder adequadamente as mudanças ocorridas no ambiente, ao contrário da análise comportamental. Neste processo geralmente utilizam-se peixes, que são considerados ferramentas experimentais, atuando como biosensores e bioindicadores (KUKLINA; KOUBA; KOZÁK, 2013). Os peixes possuem uma alta sensibilidade a variações de substâncias em seu ecossistema aquático, o que permite constatar rapidamente quaisquer variações abruptas apenas pela observação de seu comportamento. No processo de análise comportamental, diversas rotinas de teste com situações e condições ambientais simuladas são desenvolvidas, com o intuito de investigar diversos comportamentos: aprendizagem, medo, ansiedade, interação social, memória, agonísticos e esquiva (RESENDE; SIEBEL; BONAN, 2015).

Dentre as inúmeras espécies de peixes existentes, *Danio rerio*, popularmente intitulado como *zebrafish*, paulistinha ou peixe-zebra, é conhecido como um ótimo modelo experimental, sendo largamente utilizado em estudos de Etologia, Toxicologia e Genética. A utilização desta espécie se sobressai dentre as demais dadas as muitas vantagens: manutenção simplificada; pequeno porte (3 a 4 cm); alta reprodutibilidade; criação de baixo custo; elevada homologia genética em relação à dos mamíferos; e genoma sequenciado (RESENDE; SIEBEL; BONAN, 2015). A Figura 2.1 apresenta o peixe *zebrafish*.

Figura 2.1: *Zebrafish* e seus estágios de desenvolvimento: (a) adulto e (b) larva.



(a) (b)

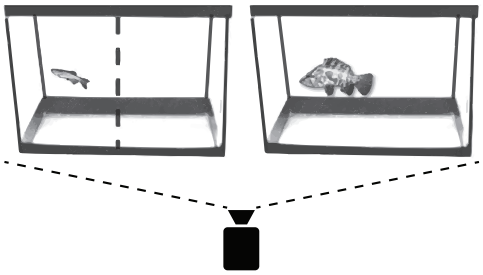
Fonte: Adaptado de Zebrafishfilm.org (2018).

A fim de realizar a análise comportamental de peixes (principalmente os da espécie *zebrafish*) e avaliar cada possível comportamento, protocolos foram desenvolvidos, aprimorados e otimizados no decorrer dos anos. Alguns dos protocolos

conhecidos e explorados demonstrados em Resende, Siebel e Bonan (2015) são descritos a seguir:

- Exposição ao predador: protocolo de avaliação do comportamento de medo, no qual aplica-se feromônios de alarme (substâncias sintéticas ou naturais) ou exposição visual à um predador para estimular este comportamento no peixe. Desta forma, vídeos são capturados e são medidas as distâncias do peixe em relação ao predador, a atividade natatória e o posicionamento (fundo ou superfície) do animal no aquário. Caso o peixe apresente respostas de afastamento do predador, *freezing* (congelamento, o peixe permanece em estado estático ou de baixa movimentação), alocação no fundo do aquário e movimentos erráticos, indica-se a presença da condição de medo no animal. Na Figura 2.2 é mostrado o protocolo de exposição ao predador;

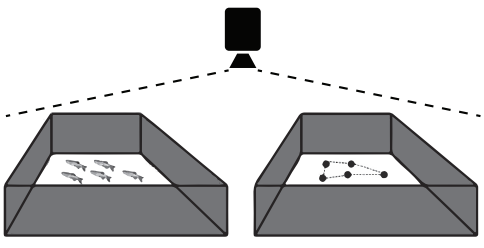
Figura 2.2: Teste de exposição ao predador.



Fonte: Adaptado de Resende, Siebel e Bonan (2015).

- *Shoaling*: *shoal* refere-se a cardume, sendo este um protocolo para avaliação do comportamento de interação social. Este protocolo consiste na aplicação de cinco peixes em um aquário, no qual são obtidas a distância entre os animais, isolamento de animais (os que se situam distante do grupo) e polarização do cardume (sincronia do nado). Captura-se a movimentação dos animais e, por meio desta, avaliam os parâmetros mencionados anteriormente, sendo possível estudar doenças comportamentais, efeitos de drogas, modulação farmacológica e estresse. Na Figura 2.3 é exibido o protocolo de *shoaling*;

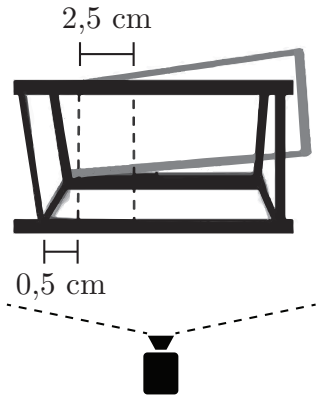
Figura 2.3: Teste de *shoaling*.



Fonte: Adaptado de Resende, Siebel e Bonan (2015).

- Agressividade: protocolo de avaliação do comportamento agonístico. Um peixe é inserido em um aquário de pequeno porte em conjunto de um espelho no fundo do aquário com um ângulo de 22,5° graus com uma de suas bordas posicionadas com o aquário, sendo possível refletir a imagem do animal e, a imagem torna-se maior conforme o animal realiza a aproximação da borda de contato do aquário. São gravadas as imagens da movimentação do animal e posteriormente são realizadas divisões no aquário para realização de analises e registros, sendo possível determinar o tempo que o animal mantêm uma aparência agressiva (postura, nado rápido e tentativas de ataque ao espelho). Na Figura 2.4 é ilustrado o protocolo de agressividade;

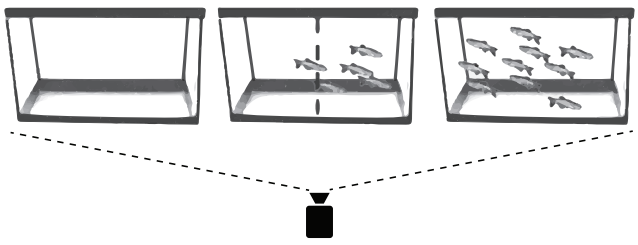
Figura 2.4: Teste de agressividade.



Fonte: Adaptado de Resende, Siebel e Bonan (2015).

- Interação social: protocolo para avaliação do comportamento de interação social quando um cardume da mesma espécie é exposto próximo a outro cardume e a um aquário vazio. Três aquários de pequeno porte são utilizados e inseridos próximos, sendo um aquário vazio, um aquário com 5 peixes (aquário-teste) e outro aquário com 15 peixes (aquário-estímulo). É realizada a gravação das imagens e uma linha central vertical é traçada no aquário-teste, que faz a divisão entre pontos de menor interação (cardume próximo ao aquário vazio) e maior interação (cardume próximo ao aquário-estímulo). Na Figura 2.5 é demonstrado o protocolo de interação social;

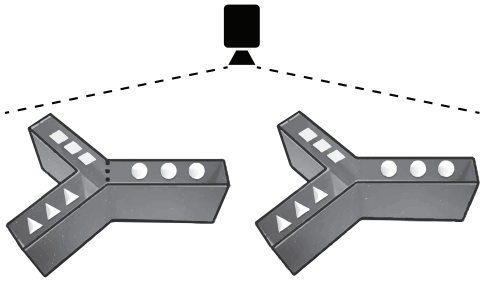
Figura 2.5: Teste de interação social.



Fonte: Adaptado de Resende, Siebel e Bonan (2015).

- Labirinto em Y: protocolo para avaliação da obtenção e consolidação de memória. Utiliza-se um aquário em formato de Y, possuindo três braços de comprimentos iguais, com o fundo branco e laterais pintadas de preto com pistas visuais geométricas transparentes (quadrado, triângulo e círculo). Inicialmente o peixe é inserido no aquário para a sessão de treino com um dos braços fechados, no qual o animal pode explorar entre os dois braços. Na sessão de teste o braço fechado é aberto, permitindo sua locomoção entre todos os braços e, assim, são registrados parâmetros de atividade natatória (distância percorrida, tempo imóvel, entre outros). Gravações das sessões são realizadas e analisadas, obtendo parâmetros como o tempo gasto no braço recém-aberto e número de entradas. Estes parâmetros simbolizam a capacidade de exploração de um novo ambiente (aspecto de novidade) e, consequentemente, sinaliza a memória espacial no animal. Na Figura 2.6 é exibido o protocolo de labirinto em Y;

Figura 2.6: Teste de labirinto em Y.

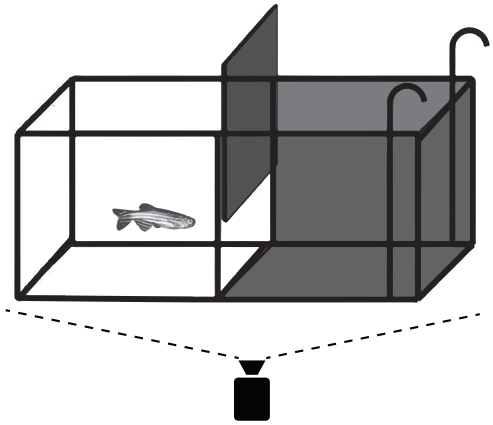


Fonte: Adaptado de Resende, Siebel e Bonan (2015).

- Esquiva inibitória: protocolo para avaliação da memória. Para este protocolo um pequeno aquário é dividido por uma barra divisória móvel em duas partes iguais, claro e escuro, no qual uma parte recebe um revestimento branco e a outra o revestimento preto. Além disso, no compartimento escuro, 2 eletrodos são conectados a uma fonte DC de 8V e geram um choque de ±0,2V. São realizadas sessões gravadas de treino e teste e, como protocolo inicial, o peixe é inserido no compartimento claro e aguarda-se a sua familiarização com o ambiente e, posteriormente, a divisória é elevada, fazendo com que o animal transite para o compartimento escuro (no qual geralmente possui preferência). Na sessão de treino o peixe passa pelo protocolo inicial e, quando há sua transição para o lado escuro, o animal recebe o choque por um curto período de tempo e posteriormente é transferido para o aquário de moradia, no qual permanecerá por 24 horas. Decorrido o tempo, é iniciado a sessão de teste, em que realiza-se o protocolo inicial e, no momento de abertura da barreira, contabiliza-se o tempo até o mesmo

realizar a transição para o lado escuro. São contabilizados os tempos para a realização da transição para ambas as sessões e, por meio desse valor, determina-se o índice de retenção de memória do animal. Na Figura 2.7 é mostrado o protocolo de esquiva inibitória;

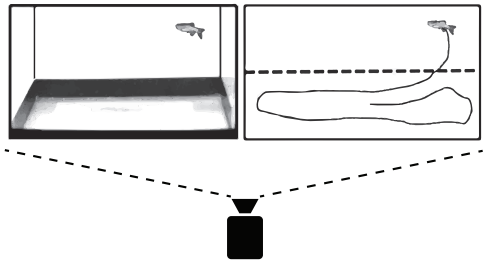
Figura 2.7: Teste de esquiva inibitória.



Fonte: Adaptado de Resende, Siebel e Bonan (2015).

- Teste de campo aberto: protocolo para avaliação de comportamentos de ansiedade, estresse e efeitos de fármacos. O peixe é inserido em um aquário de cor uniforme e sem divisórias, explorando-o livremente. Sua atividade natatória é registrada em vídeo para análises e consideram os parâmetros de velocidade média, movimentos erráticos, distância percorrida, congelamento, entre outros. Alterações como diminuição da atividade exploratória, permanência no fundo do aquário, aumento da frequência de ocorrência dos movimentos erráticos são alguns dos indicadores de ansiedade ou estresse. Na Figura 2.8 é ilustrado o protocolo de teste de campo aberto.

Figura 2.8: Teste de campo aberto.



Fonte: Adaptado de Resende, Siebel e Bonan (2015).

Os protocolos descritos acima comumente são realizados em ambientes controlados, com peixes adaptados às condições de cativeiro (não-naturais). Pesquisas demonstram que peixes selvagens possuem diferenças morfológicas se comparados a peixes de cativeiro (SARAIVA; POMPEU, 2014). Visando tornar o ecossistema aquático próximo ao natural e diminuir os impactos das condições impostas

pela criação em cativeiro, aplica-se a técnica de enriquecimento ambiental. Esta técnica em ascensão consiste na inserção de diversos estímulos no ambiente de localização do animal a fim de melhorar sua condição física e mental. Os seguintes estímulos descritos podem ser utilizados (HICKMAN et al., 2017; AZEVEDO; BARÇANTE, 2018):

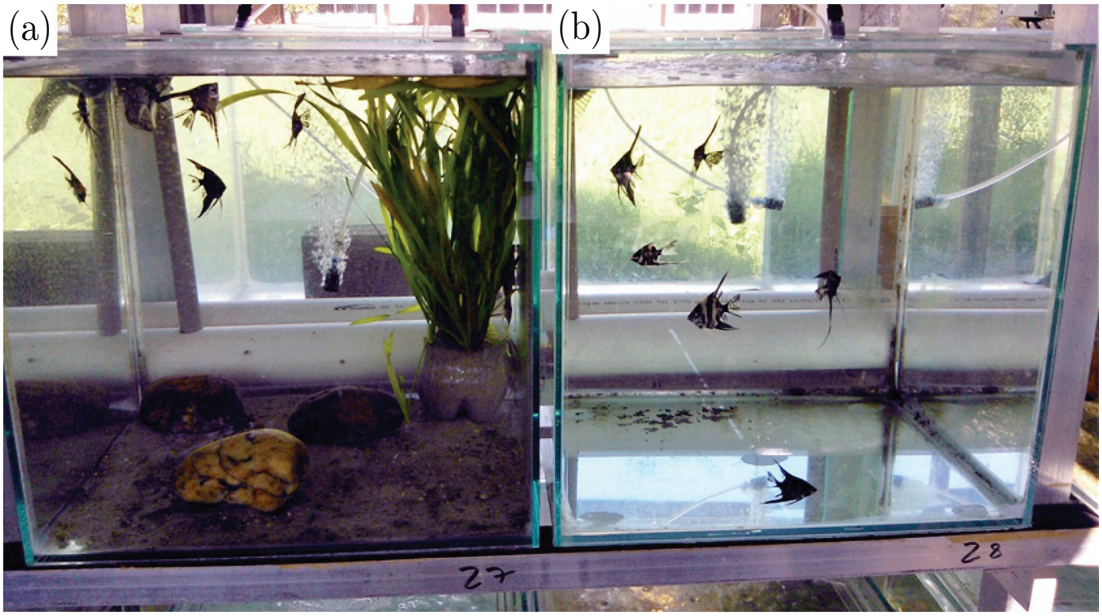
- Alimentar: refere-se aos meios para prover alimentos à um animal, sendo realizados de forma não-padronizadas (rotineiras), variando a dieta e inserida em locais de difícil acesso ou escondidas, por exemplo;
- Social: inserção no mesmo ambiente de indivíduos da mesma espécie ou de espécies diferentes;
- Físico: relacionado à questão estrutural do ambiente, no qual são alocados objetos como pedras, plantas artificiais e esconderijos;
- Sensorial: explora os sentidos do animal induzindo-o com sons, odores e imagens, por exemplo;
- Cognitivo: desafia a capacidade do animal para a resolução de problemas (acionamento de um dispenser para liberação de comida, por exemplo).

A aplicação dos estímulos citados anteriormente, garante benefícios ao animal como: desenvolvimento morfológico ligeiramente mais avançado, fácil manejo, alta reprodutibilidade, menor incidência de problemas de saúde, entre outros.

Um comportamento animal mais próximo do natural sinaliza boas condições de vida, por outro lado, comportamentos estereotipados e anormais por longos períodos de tempo sinalizam malefícios. A avaliação dessas manifestações são geralmente observadas por meio da análise comportamental (AZEVEDO; BARÇANTE, 2018). Contudo, dada a presença de mais fatores presentes no ambiente a ser realizado a análise comportamental, ferramentas automatizadas mais robustas (como redes neurais) garantem melhores resultados e são necessárias para a realização adequada desta tarefa (IPIÑA et al., 2019). A Figura 2.9 apresenta um exemplo de ambiente aquático com enriquecimento ambiental.

A utilização da tecnologia visando a facilitação, simplificação e automatização de processos têm se tornado frequente, e nos experimentos de análises comportamentais de peixes não é diferente (PARRA et al., 2018). Computadores, câmeras, sensores, atuadores e demais dispositivos são amplamente empregados nesta área, garantindo maior precisão (diminuição de erros humanos), repetibilidade, flexibilidade, controle e medição nos experimentos, padronização de processos, redução de

Figura 2.9: Exemplo de ambiente (a) com e (b) sem enriquecimento ambiental.



Fonte: Adaptado de Diniz et al. (2020).

custos, aumento da produtividade, entre outras vantagens. Por meio da aplicação destes dispositivos é possível gerar análises robustas e complexas, porém, esbarra-se na problemática do tratamento correto do grande volume de dados obtidos, necessitando de um elevado poder computacional para o processamento das muitas variáveis envolvidas (RESENDE; SIEBEL; BONAN, 2015).

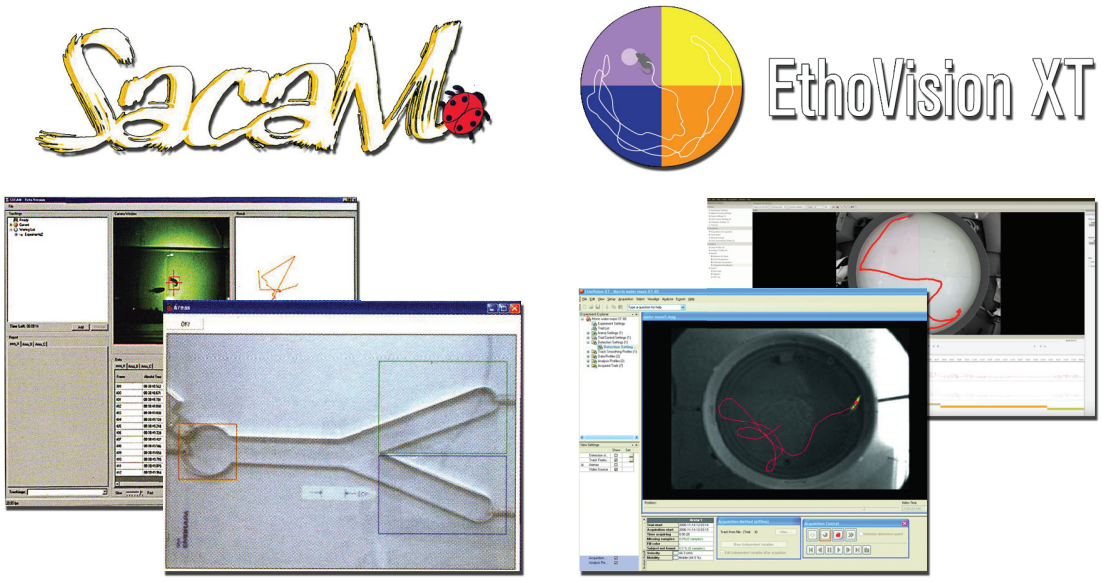
Atualmente o processo de análise comportamental de peixes foca-se na observação de seu movimento por meio de câmeras, realizando o rastreamento e monitoramento das ações do animal. Programas de computador como SACAM (EMBRAPA, 2009), ZebraTrack (ZEBRATRACK, 2017) e EthoVision XT (NOLDUS, 2019) visam auxiliar pesquisadores automatizando o processo, identificando os peixes em seu habitat, extraindo informações, gerando gráficos, realizando análises e gravações de imagens e vídeos, entre outros. Na Figura 2.10 é mostrado algumas imagens de operação dos softwares mencionados.

2.2 Visão computacional

Desde a década de 50, pesquisadores buscavam replicar o funcionamento da visão humana, algo extremamente difícil e complexo (BALLARD; BROWN, 1982). Surgiram, então, linhas de pesquisas relacionadas a reprodução da visão: funcionamento do olho, funcionamento do córtex visual e funcionamento do cérebro (MARR, 1982).

A linha de pesquisa referente ao funcionamento do olho, dentre as citadas, é

Figura 2.10: Softwares de análises comportamentais de animais: (a) SACAM e (b) EthoVision XT.



Fonte: Adaptado de EMBRAPA (2009) e Noldus (2019).

a mais avançada atualmente, onde foram desenvolvidos componentes, sensores, processadores e lentes, que permitem a captura da imagem com alta fidelidade (HERRERA; LABRAM, 2020). Porém, a réplica do olho apenas realiza a recepção das imagens e não é suficiente para reproduzir por completo a visão humana, pois fatores como armazenamento e identificação de objetos e padrões não estão presentes e estas fazem parte das linhas de pesquisa relacionadas ao cérebro (MARR, 1982; PALMER, 1999).

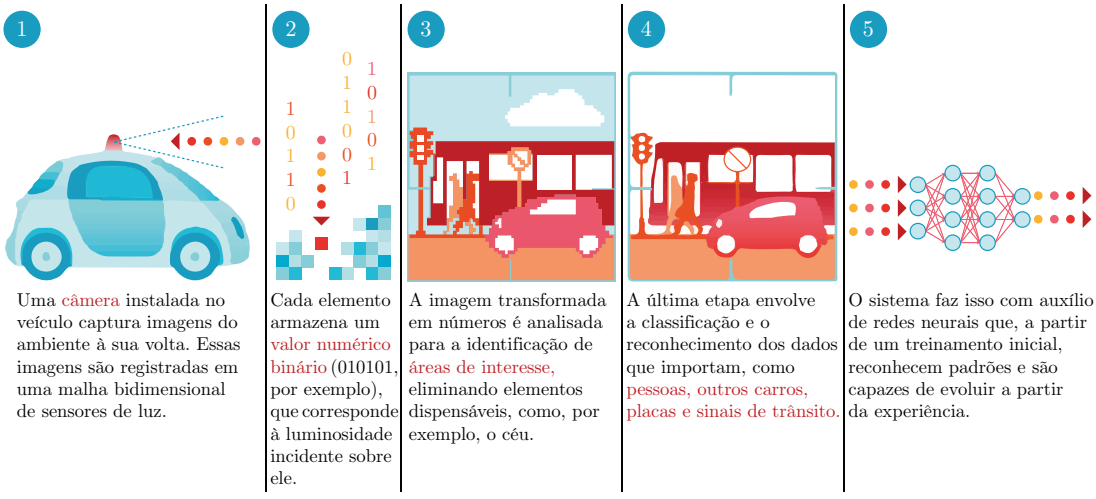
Estudos referentes ao funcionamento do cérebro continuam sendo desenvolvidos, porém, devido à alta complexidade envolvida, um completo entendimento e uma possível réplica fiel deste orgão encontram-se em uma realidade ainda distante. Contudo, dado os atuais adventos tecnológicos relacionados a área de computação, a aprendizagem de máquina (*machine learning*) permitiu que as máquinas aprendessem a partir de informações (imagens e sons, por exemplo), agindo de maneira similar ao cérebro - algoritmo de redes neurais - e tornando-se dispositivos considerados inteligentes (MARR, 1982; SONKA; HLAVAC; BOYLE, 2014).

A aprendizagem de máquina é um campo que pertence a área de inteligência artificial e extremamente utilizado em visão computacional atualmente. Baseia-se na ideia de que máquinas possam aprender com dados, identificar padrões e tomar decisões de forma independente, ou seja, sem qualquer interferência humana (MITCHELL, 1997).

O campo de estudo envolvendo processos visuais e computacionais é chamada de visão computacional. A visão computacional é um campo de estudo intimamente ligado à área de inteligência artificial e consiste em desenvolver técnicas para auxiliar os computadores a "enxergar" e compreender imagens e vídeos. Além disso, a visão computacional abrange o estudo de diversas tarefas: aquisição, extração, processamento, análise e entendimento de imagens digitais (SONKA; HLAVAC; BOYLE, 2014).

Basicamente, para que uma máquina consiga "enxergar" uma imagem, é necessário executar três etapas: aquisição, processamento e entendimento. Na etapa de aquisição, as imagens podem ser obtidas de uma fonte estática ou tempo real, advindas de vídeos, fotos ou renderizações 3D. A etapa de processamento é a qual o algoritmo de aprendizagem de máquina escolhido começa atuar e realiza operações para obter informações de interesse das imagens e algumas destas operações são: pré-processamento (manipulações de dimensões, sistema de cor, saturação, brilho e contraste, por exemplo), extração de características (*features*), segmentação e treinamento (indução de um modelo por meio de um processo de entrada de diversas imagens contendo o ponto de interesse na máquina, fazendo-a aprender as características próprias presentes nesses dados de entrada). Por fim, tem-se a etapa de entendimento, na qual a máquina modela os dados, identifica, classifica, detecta e rastreia quantitativamente e qualitativamente pontos de interesse (objetos, animais, pessoas, por exemplo) nas imagens de entrada (BALLARD; BROWN, 1982; SONKA; HLAVAC; BOYLE, 2014). Na Figura 2.11 é demonstrado um exemplo de visão computacional com aplicação em carros autônomos.

Figura 2.11: Aplicação da visão computacional em carros autônomos.



Fonte: Adaptado de Andrade (2019).

Existe uma vasta gama de algoritmos de aprendizagem de máquina e que podem ser aplicados para diversas áreas, incluindo a visão computacional. Os

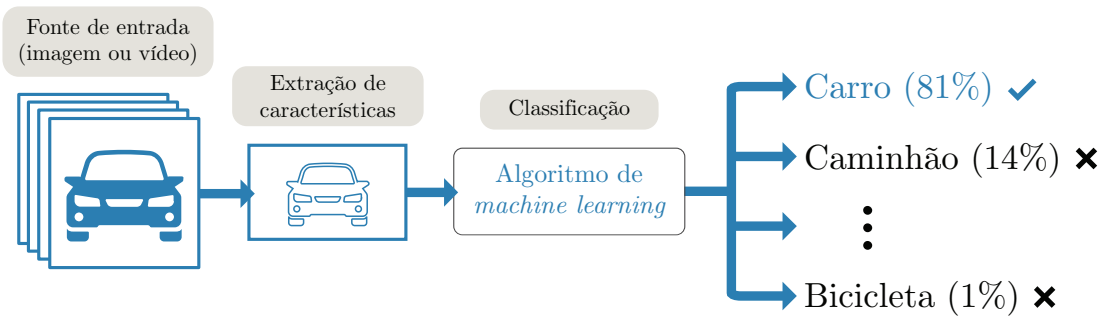
algoritmos de aprendizagem de máquina mais explorados e utilizados atualmente em visão computacional são: Redes Neurais, SVM (*Support Vector Machine*), KNN (*K Nearest Neighbors*), Naive Bayes, Regressão Linear e Regressão Lógistica (BRADSKI, 2008; SUN, 2016).

Devido ao grande avanço da visão computacional ao longo do tempo, esta se tornou uma ferramenta chave para diversas áreas na automatização de processos como: reconhecimento de pessoas, detecção de faces, diagnóstico de doenças, carros autônomos e controle de qualidade de produtos. As tarefas da visão computacional que abrange estas e muitas outras aplicações são o reconhecimento, detecção e rastreamento de objetos (KLETTE, 2014; SUN, 2016).

2.2.1 Reconhecimento de objetos

Reconhecimento de objetos ou também chamado de classificação de imagens é uma tarefa que envolve identificar objetos utilizando computadores em imagens ou vídeos obtidos por câmeras (SONKA; HLAVAC; BOYLE, 2014; KLETTE, 2014). A partir de uma fonte de dados de entrada (uma foto ou vídeo), o computador é treinado e aprende a extrair características (tamanho, forma, textura e cor por exemplo) associadas a cada tipo de objeto, sendo capaz de classificar o objeto presente em uma imagem ou vídeo, atribuindo-o a um único rótulo (*label*) de uma lista finita de classes e retornando suas probabilidades associadas (SONKA; HLAVAC; BOYLE, 2014; KLETTE, 2014; ROSEBROCK, 2017). A Figura 2.12 apresenta a aplicação desta tarefa.

Figura 2.12: Diagrama de reconhecimento de objetos.



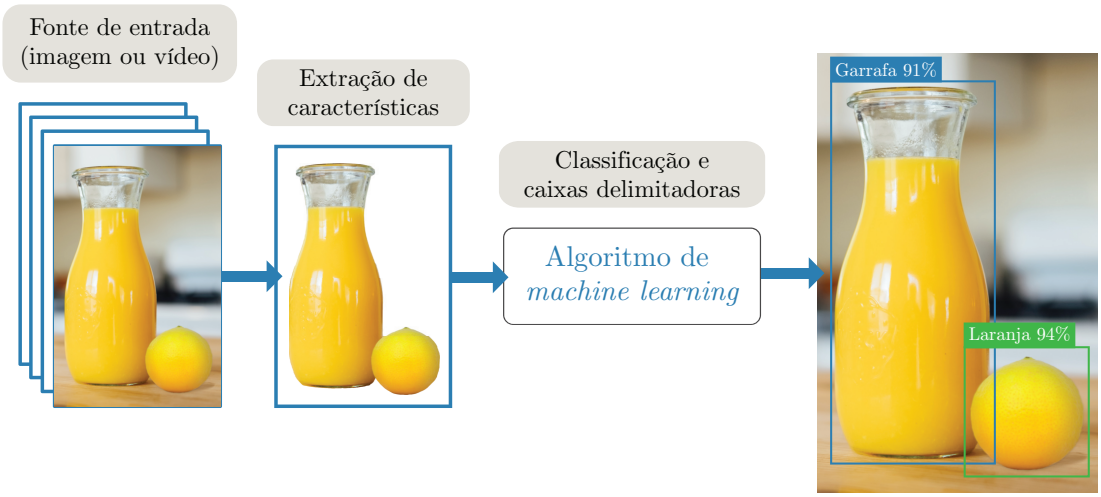
Fonte: Adaptado de MathWorks (2019).

2.2.2 Detecção de objetos

A tarefa de detecção de objetos é similar à de reconhecimento de objetos. Diferentemente do reconhecimento de objetos, a detecção de objetos torna possível identificar e localizar um ou mais objetos em uma imagem ou vídeo. Para os

objeto(s) detectado(s) são retornados os rótulos e as probabilidades associadas e utilizam-se caixas delimitadoras (do inglês, *bounding boxes*) para indicar suas localizações na imagem ou vídeo (KLETTE, 2014). A Figura 2.13 ilustra a execução desta tarefa.

Figura 2.13: Diagrama de detecção de objetos.



Fonte: Do autor.

2.2.3 Rastreamento de objetos

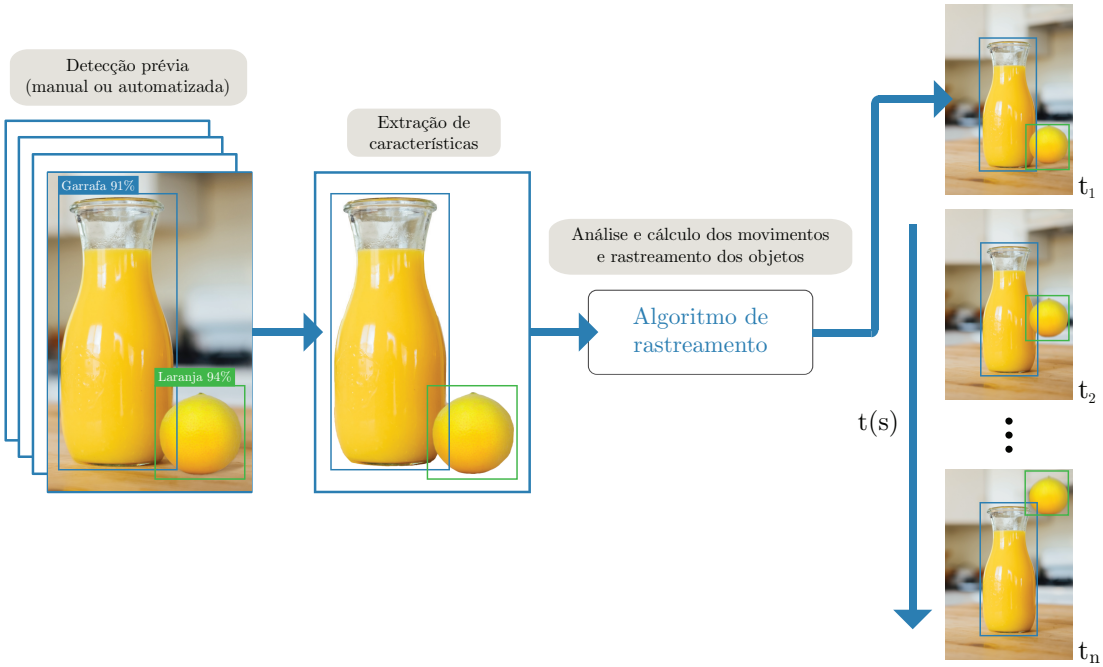
O rastreamento de objetos é uma tarefa que consiste em analisar, identificar e rastrear o movimento de objetos em uma sequência temporal de imagens (vídeo). Geralmente esta tarefa é utilizada em conjunto com a tarefa de detecção de objetos (detecção manual - definida humanamente - ou automatizada - *machine learning*), no qual o objeto é localizado na imagem e esta informação (caixas delimitadoras) é transferida para o rastreador (SONKA; HLAVAC; BOYLE, 2014).

Esta é uma tarefa ampla, envolvendo variados tipos de análises e problemas, como é o caso do comportamento do objeto a ser rastreado. A resolução de problemas como profundidade (a distância relativa do objeto no vídeo), oclusão (o objeto rastreado é oculto por outro objeto), informações de movimento do objeto (referente ao estado atual do objeto, podendo ser estático - parado - ou dinâmico - rápido ou lento) e movimento da câmera (referente ao estado atual da câmera, podendo ser estático ou dinâmico) são cruciais para a realização de um rastreamento bem sucedido (SONKA; HLAVAC; BOYLE, 2014).

Uma câmera captura de forma tridimensional (3D) as informações de movimento (direção, velocidade e distância) de um objeto e projeta seu vetor de velocidade em um plano bidimensional (2D) - imagem. Esta projeção é utilizada para estimar as informações de movimento advindas das imagens capturadas e,

consequentemente, efetuar o rastreamento do objeto (HILDRETH; ULLMAN, 1982). Técnicas como subtração de objeto do fundo (*background subtraction*) ou por extração de características são largamente aplicadas para rastreamento de objetos (LUO et al., 2014; SONKA; HLAVAC; BOYLE, 2014). A Figura 2.14 demonstra o funcionamento desta tarefa.

Figura 2.14: Diagrama do rastreamento de objetos.



Fonte: Do autor.

2.3 Redes neurais

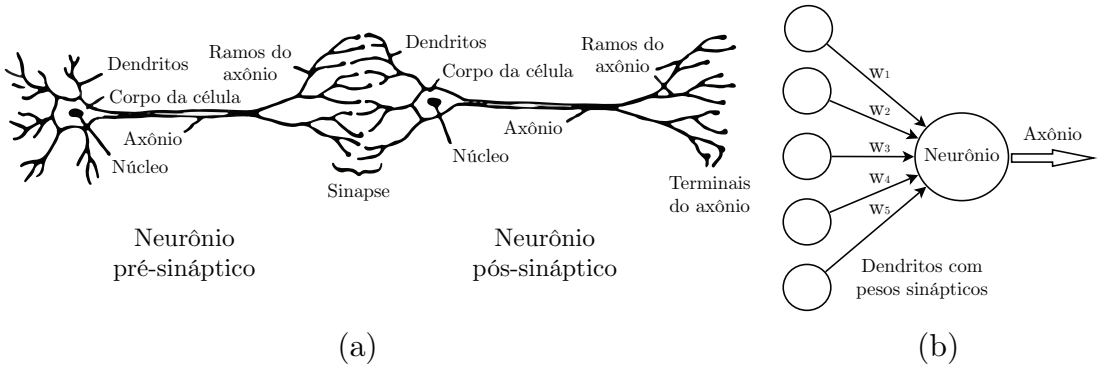
As redes neurais (NN, do inglês, *Neural Networks*) ou também chamadas de redes neurais artificiais (ANN, do inglês, *Artificial Neural Networks*) são modelos computacionais inspirados pelo funcionamento dos neurônios (células nervosas) presentes no cérebro dos animais. Como definido por Haykin (2009):

> "Uma rede neural é um processador distribuído paralelamente em massa, composto de unidades de processamento simples que têm uma propensão natural para armazenar conhecimento experimental e disponibilizá-lo para uso. Assemelha-se ao cérebro em dois aspectos:
>
> 1. O conhecimento é adquirido pela rede a partir de seu ambiente por meio de um processo de aprendizagem.
> 2. Os pontos fortes da conexão interneurônio, conhecidos como pesos sinápticos, são utilizados para armazenar os conhecimentos adquiridos."

O sistema nervoso consiste de neurônios que são conectados a outros por meio de axônios e dendritos. A região de conexão entre estes é chamada de sinapse e a intensidade da sinapse está intimamente ligada a resposta ao estímulo externo.

Por meio do conhecimento deste mecanismo, adaptou-se o processo e foram desenvolvidas as redes neurais artificiais. Nas ANN, as unidades de processamento (neurônios) são conectadas à outras por meio dos pesos (intensidade das sinapses), dentre os quais são escalados (multiplicados) para cada entrada das unidades de processamento e computados nesta por uma função. O resultado computado é propagado para o próximo neurônio e assim por diante. A aprendizagem é realizada quando a ANN recebe dados de treinamento (estímulos externos), como por exemplo pixels de uma imagem, e modifica os pesos que conectam os neurônios, visando garantir predições corretas para uma determinada tarefa (AGGARWAL, 2018). A Figura 2.15 apresenta um exemplo de rede neural biológica e rede neural artificial.

Figura 2.15: Exemplos de (a) rede neural biológica e (b) rede neural artificial.

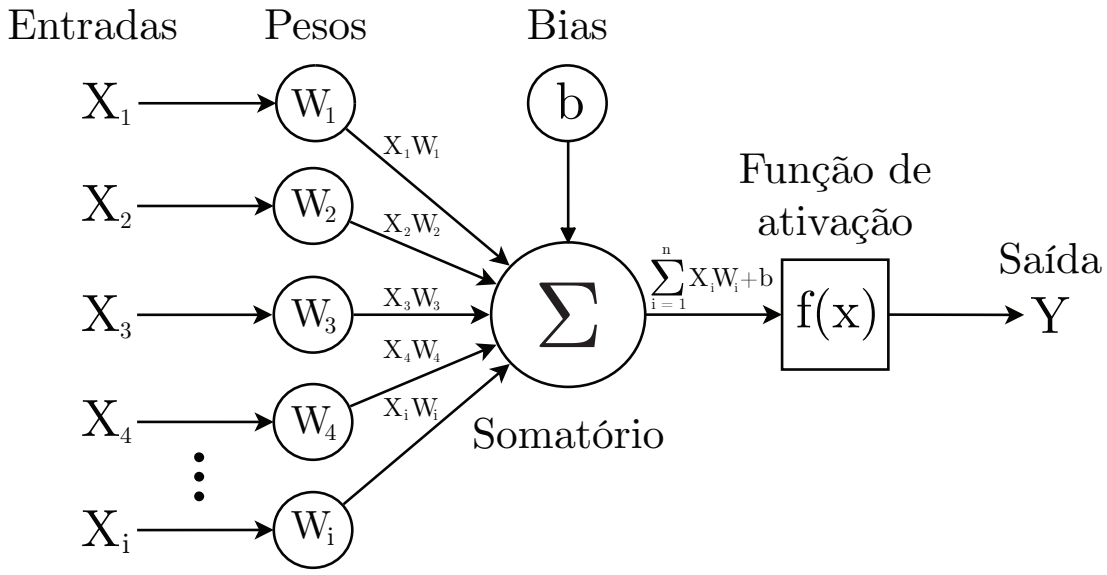


Fonte: Adaptado de Aggarwal (2018).

Por meio deste mecanismo base de rede neural artificial, Rosenblatt (1958) propôs o *perceptron*, considerado o tipo mais simples de rede neural com apenas uma camada. O *perceptron* recebe em suas entradas valores que são denotados por x_i, dentre os quais são multiplicados pelos pesos w_i e (comumente) somados a um viés (*bias*, b). O resultado do somatório de pesos multiplicados de n ramos com o *bias* é propagado para a função matemática de ativação (degrau, sigmoide, tangente hiperbólica, *softmax* ou ReLU - *Rectified Linear Unit*), responsável por limitar o resultado à um certo intervalo e permitindo ou não a passagem do sinal, gerando o resultado final na saída y. A Figura 2.16 apresenta o esquema do *perceptron*.

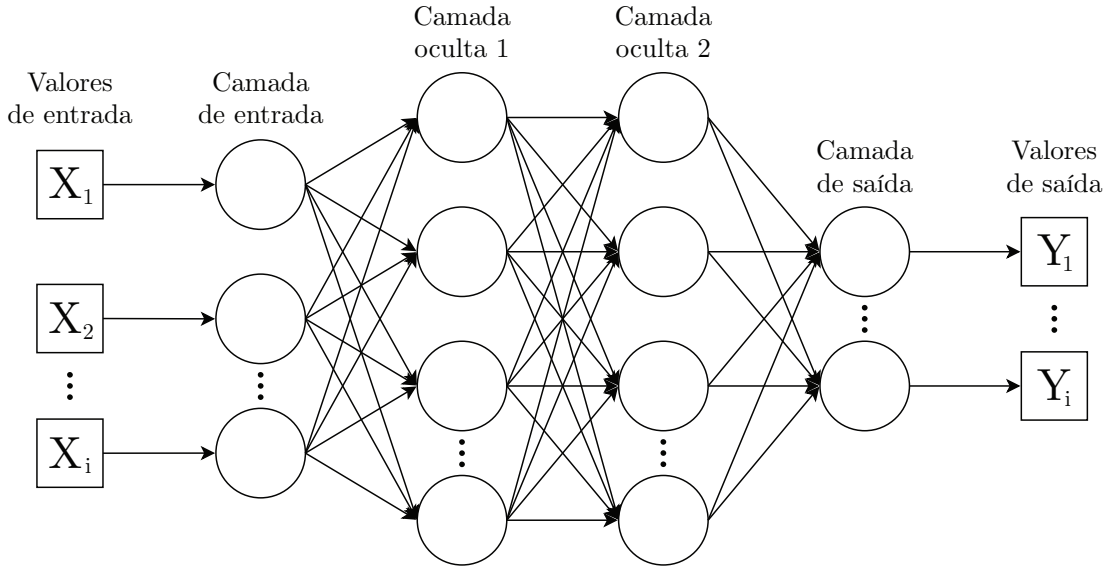
Entretanto o *perceptron* é limitado na resolução de problemas mais complexos, principalmente devido a simplicidade de sua estrutura. Logo, foi desenvolvido o *Multilayer Perceptron* (MLP), que consiste na combinação de *perceptrons* em várias camadas. No caso do MLP a estrutura (camadas) é dividida da seguinte forma: entrada, escondida (camada oculta, do inglês, *hidden layer*) e saída (BUDUMA; LOCASCIO, 2017). A Figura 2.17 apresenta o esquema do *multilayer perceptron*.

Figura 2.16: O *Perceptron*.



Fonte: Adaptado de Buduma e Locascio (2017).

Figura 2.17: O *Multilayer Perceptron*.



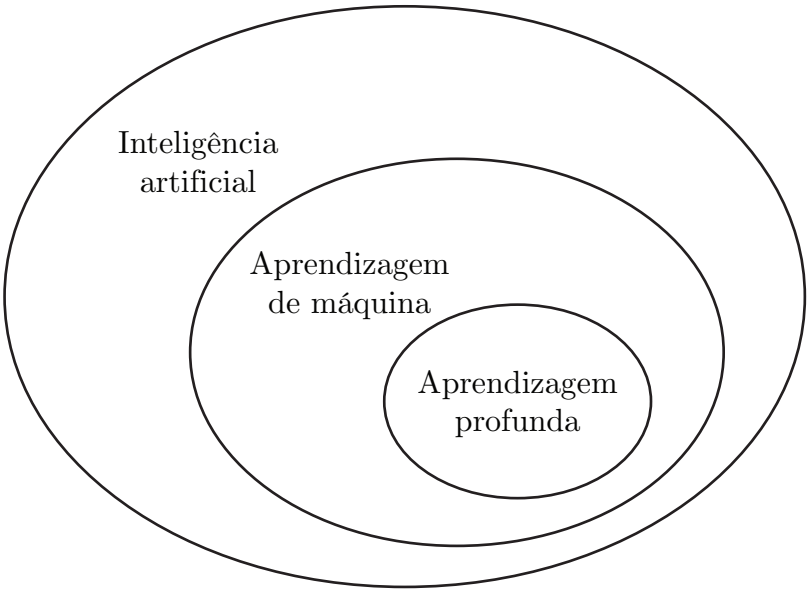
Fonte: Adaptado de Patterson e Gibson (2017).

Dado que o MLP é um conjunto de *perceptrons*, seu funcionamento, portanto, é similar. Na camada de entrada são recebidos valores (x_i), estes são propagados para a primeira camada oculta, dentre os quais serão ali associados a pesos (w_i) e *biases* (b_i), calculados pela funções de somatório e os resultados destas aplicados nas funções de ativação. Estes valores computados da camada anterior, são propagados para a próxima camada oculta, que realiza o mesmo processo. O resultado da última camada oculta é transmitido para a camada de saída e esta emite os dados finais (PATTERSON; GIBSON, 2017; AGGARWAL, 2018).

2.3.1 Aprendizagem profunda

O conceito de aprendizagem profunda (DL, do inglês, *Deep Learning*) é antigo mas apenas se estabeleceu fortemente nos últimos anos com o advento das redes neurais. O termo aprendizagem profunda refere-se à ideia de profundidade, ou seja, quantas camadas sucessivas contribuem para um modelo de dados, como definido por Chollet (2017) "uma forma multiestágios para aprender representações de dados". Como ilustrado pela Figura 2.18 a aprendizagem profunda pertence à área da inteligência artificial, localizada na subárea de aprendizagem de máquina.

Figura 2.18: Inteligência artificial, aprendizagem de máquina e aprendizagem profunda.



Fonte: Adaptado de Chollet (2017).

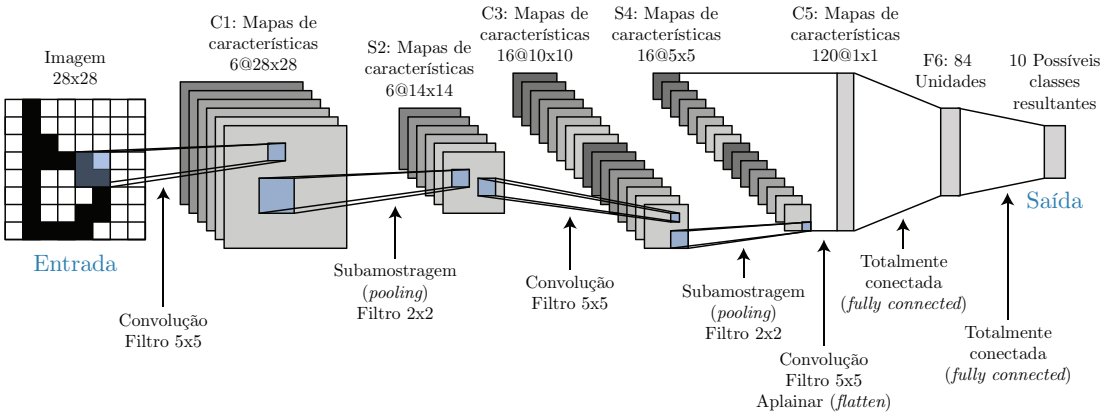
O MLP é considerado o ponto de partida, permitindo o desenvolvimento desta área, assim como as redes neurais convolucionais (CNN, em inglês, *Convolutional Neural Networks*). A partir destas redes mais complexas (estruturalmente e organizacionalmente), com mais parâmetros, neurônios e mais camadas ocultas, foi possível alcançar excelentes resultados, próximos de desempenhados por um humano, em áreas consideradas difíceis para o aprendizado de máquina como: classificação de imagens, reconhecimento de fala, direção autônoma, assistentes digitais e tradução de textos (CHOLLET, 2017; PATTERSON; GIBSON, 2017; AGGARWAL, 2018).

2.3.2 Redes neurais convolucionais

As redes neurais convolucionais também chamadas *ConvNets* ou CNN são uma variação dos MLP que utilizam operações matemáticas de convolução com o

objetivo de aprender características de ordem elevada presentes nos dados. Assim como nas ANN, seu desenvolvimento foi inspirado em modelos biológicos, mais precisamente a organização do córtex visual dos animais. Atualmente as CNN são amplamente empregadas em tarefas de visão computacional como reconhecimento de objetos e classificação de imagens, devido ao seu alto e consistente desempenho (HAYKIN, 2009; PATTERSON; GIBSON, 2017). A Figura 2.19 apresenta um exemplo de rede neural convolucional para reconhecimento de dígitos: a LeNet-5.

Figura 2.19: Arquitetura da CNN LeNet-5, utilizada para o reconhecimento de dígitos.



Fonte: Adaptado de LeCun et al. (1998).

O alto desempenho em tarefas visuais está ligado principalmente à característica das CNN em conseguir manipular com robustez distorções como redimensionamento, ruído, rotação, dentre outras, além de possuir um alto grau de invariância a translação. Estas propriedades são garantidas pelas camadas de convolução e subamostragem (do inglês, *subsampling* e também conhecido como *pooling*) presentes na arquitetura da CNN (LECUN et al., 1998).

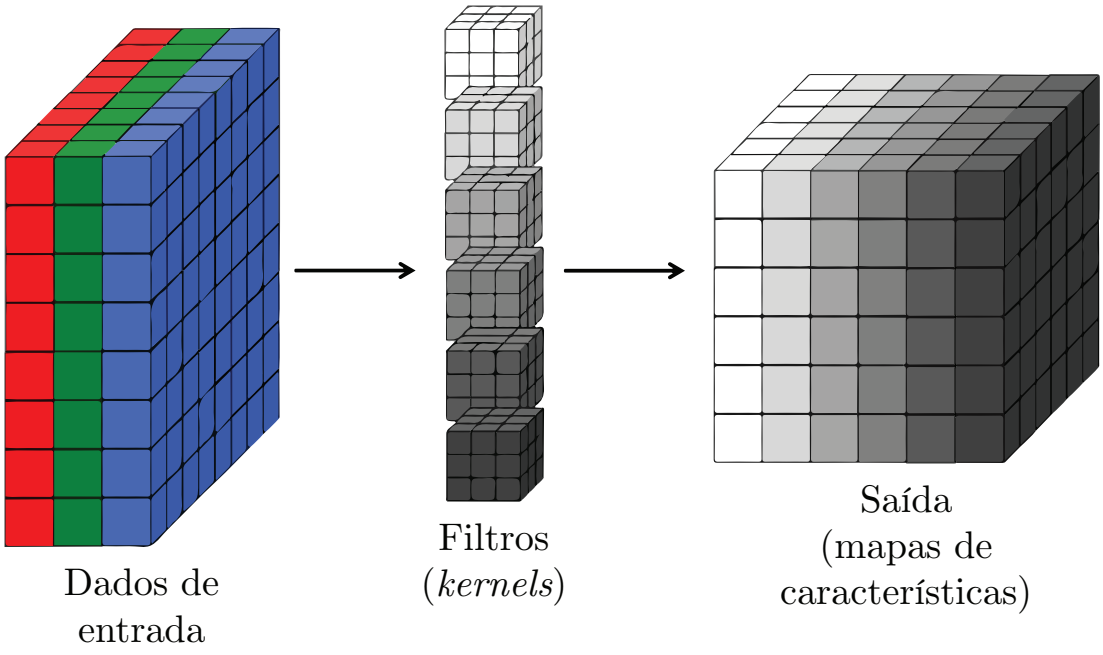
A camada de convolução, aplica a operação de convolução nos dados de entrada (por exemplo, uma imagem), resultando na geração de mapas de características (do inglês, *feature maps*, também conhecido como *activation maps*). Estes mapas podem ser considerados imagens de características extraídas dos dados, que são obtidas por meio da aplicação de filtros (do inglês, *kernels*) e, para cada filtro diferente aplicado em um dado, é gerado um mapa de característica (HAYKIN, 2009; GOODFELLOW; BENGIO; COURVILLE, 2016).

Filtros são pequenas matrizes utilizadas para funções de processamento (detecção de bordas, embaçamento, nitidez, dentre outros) e extração de características. Para funções estabelecidas os valores dos filtros são conhecidos (PARKER, 2010), no entanto, para o caso das CNN, os filtros são aprendidos de acordo

com os dados de entrada, ou seja, os valores dos filtros são os pesos que são aprendidos durante o processo de treinamento (BUDUMA; LOCASCIO, 2017).

A convolução geralmente é aplicada sobre uma imagem que possui atributos de comprimento (em inglês, *width*), altura (em inglês, *height*) e três canais de cores RGB (do inglês, *red*, *green*, *blue*) que são a profundidade (em inglês, *depth*). Este processo resulta em um volume de dados tridimensional (3D) de mesma ou menor dimensão do dado de entrada (imagem) com um (comum) aumento da terceira dimensão (profundidade). A Figura 2.20 ilustra um exemplo do processo de convolução tridimensional.

Figura 2.20: Exemplo de um processo de convolução tridimensional.

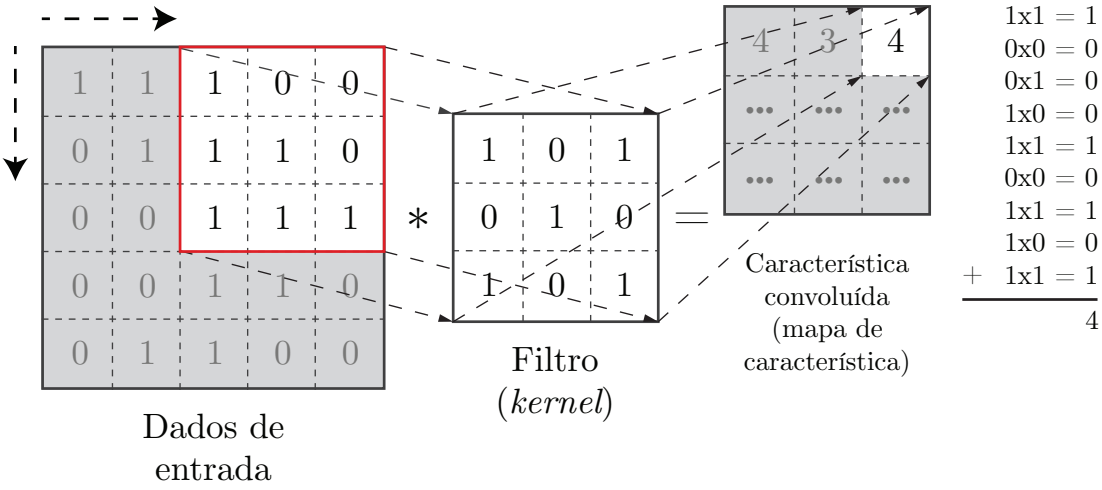


Fonte: Adaptado de Buduma e Locascio (2017).

O processo de convolução engloba os hiperparâmetros de tamanho do filtro, profundidade de saída, passo (do inglês, *stride*) e o preenchimento (do inglês, *padding*). A Figura 2.21 apresenta um exemplo do processo de convolução: um filtro de tamanho 3x3x1 percorre um dado de entrada sem preenchimento de tamanho 5x5x1 da esquerda para a direita com o passo de 1, realizando multiplicações de elemento por elemento e aplicando soma de produtos, resultando em valores únicos que compõem o mapa de característica.

A camada de subamostragem é utilizada para diminuir a dimensão dos mapas de características, garantindo a propriedade de invariância na CNN. Além disso, sua aplicação tem o intuito de diminuir a complexidade computacional. Para este tipo de operação apenas os hiperparâmetros de tamanho da subamostragem e passo são configuráveis. Usualmente aplicam-se operações de subamostragem de média (em inglês, *average pooling*) e máximo (em inglês, *max pooling*) (GO-

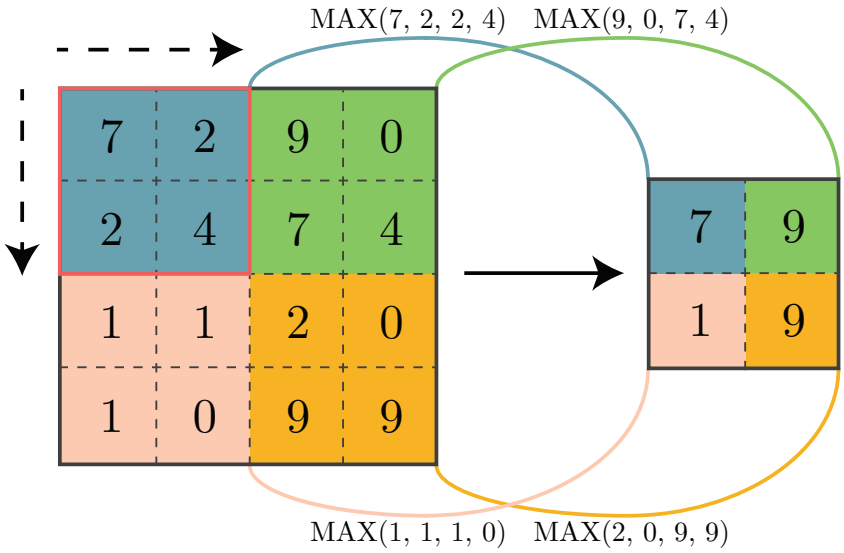
Figura 2.21: Processo de convolução.



Fonte: Adaptado de Patterson e Gibson (2017).

ODFELLOW; BENGIO; COURVILLE, 2016; BUDUMA; LOCASCIO, 2017). A Figura 2.22 demonstra um exemplo da operação *max pooling*: uma janela de subamostragem com tamanho 2x2 percorre um dado de entrada sem preenchimento de tamanho 4x4 da esquerda para a direita com o passo de 2, aplicando a operação matemática de máximo que realiza uma varredura para encontrar o maior valor entre os elementos.

Figura 2.22: Processo de *max pooling*.

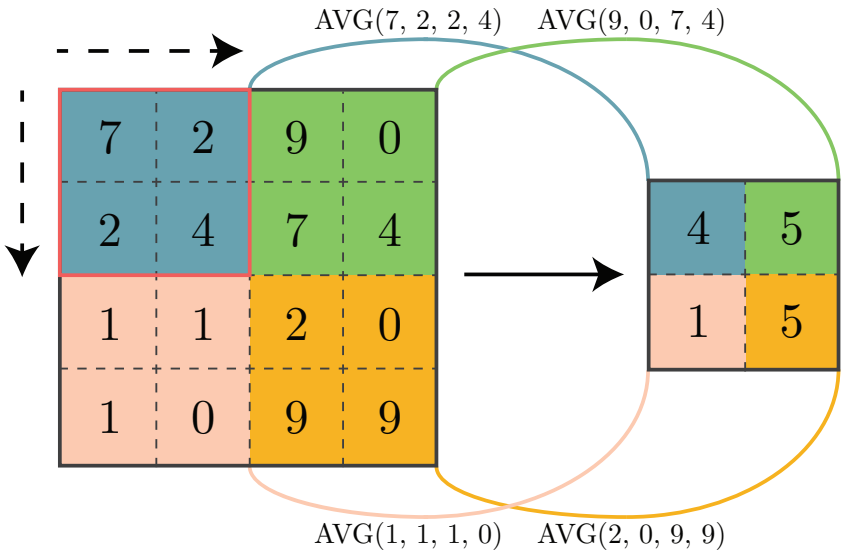


Fonte: Adaptado de Lea (2018).

Na Figura 2.23 um exemplo da operação *average pooling* é demonstrado: uma janela de subamostragem com tamanho 2x2 percorre um dado de entrada sem preenchimento de tamanho 4x4 da esquerda para a direita com o passo de 2, lendo os elementos e aplicando a operação matemática de média aritmética.

Após serem realizadas as etapas de convolução e subamostragem os mapas de características resultantes são propagados para realizar o processo de classificação.

Figura 2.23: Processo de *average pooling*.



Fonte: Adaptado de Lea (2018).

Inicialmente os dados são transformados unidimensionalmente (aplainados, do inglês, *flatten*) e seguem para as camadas totalmente conectadas (do inglês, *fully connected layers*), no qual são computados. Por fim, os resultados computados vão para a camada de saída com uma função de ativação aplicada (geralmente utiliza-se a função *softmax*) que retorna a distribuição de probabilidade das classes de saída (PATTERSON; GIBSON, 2017; BERNICO, 2018).

O processo de treinamento das redes neurais convolucionais é realizado em duas etapas: alimentação direta (do inglês, *feed-forward*) e retroalimentação (do inglês, *feed-backward*). Na primeira etapa (*feed-forward*) os dados de entrada propagam-se desde a camada de entrada até a camada de saída. Na segunda etapa (*feed-backward*) aplica-se o algoritmo de retropropagação (do inglês, *back-propagation*), que consiste em propagar reversamente pela rede o valor de erro calculado pela camada de saída, a fim de realizar ajustes adequados aos pesos visando a classificação correta dos dados. Executadas as duas etapas, têm-se uma época ou iteração (PATTERSON; GIBSON, 2017).

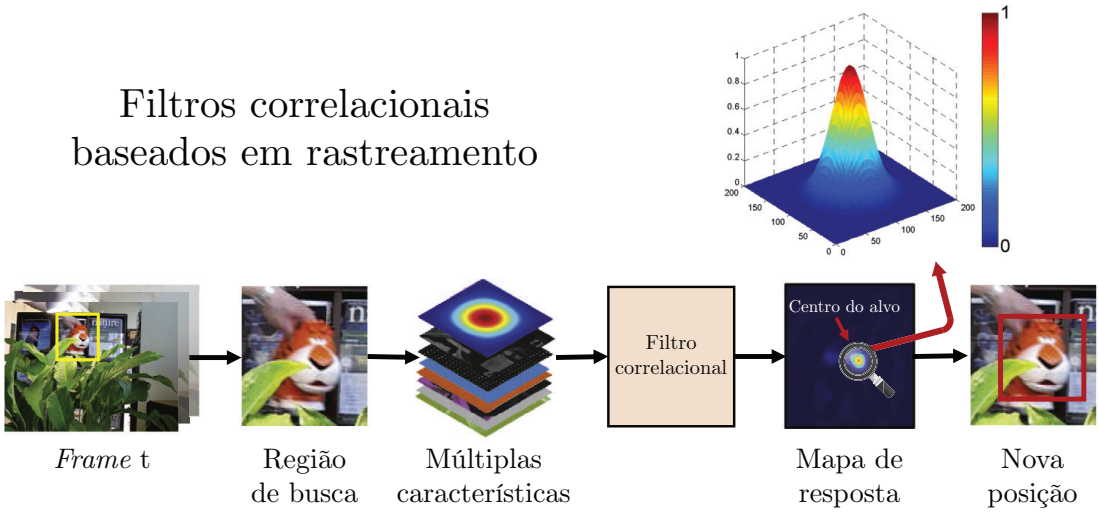
Problemas como sobreajuste (do inglês, *overfitting*), que é a aprendizagem limitada ao conjunto de dados de treinamento (capacidade de generalização pobre), e subajuste (do inglês, *underfitting*), que é o baixo nível de aprendizagem pela rede, aflingem frequentemente as CNN. Os dados de treinamento devem ser arranjados de maneira aleatória, para que a capacidade de aprendizagem e generalização da rede seja aprimorada. Além disso, técnicas como aumento de dados (do inglês, *data augmentation*), no qual aplicam-se manipulações (distorções) nas imagens visando a aprendizagem de mais características pela rede, e *dropout*, que consiste em probabilidades associadas para desativar os neurônios da rede

tornando-a mais dinâmica, auxiliam na resolução do problema de sobreajuste e também aumentam a capacidade de generalização da rede (ROSEBROCK, 2017; BERNICO, 2018).

2.4 Filtros Correlacionais

Atualmente, para a tarefa de rastreamento de objetos, diversas técnicas e estratégias podem ser adotadas a fim de executá-la com êxito, e os filtros correlacionais são uma delas. Por serem computacionalmente eficientes e robustos, os filtros correlacionais tornaram-se atrativos e são amplamente aplicados em trabalhos de variadas áreas. A ideia base dessa técnica é estimar um filtro de imagem ideal, modelando-o de acordo com a aparência do objeto, produzindo, assim, uma resposta na saída. Picos Gaussianos são gerados a partir do objeto de interesse (alvo) selecionado (por meio da detecção automatizada ou manual), no qual o valor máximo deste pico é o objeto e os valores baixos o fundo da imagem (BOLME et al., 2010; LIU et al., 2016a). Na Figura 2.24 é exemplificado a técnica de filtros correlacionais aplicado em rastreamento de objetos.

Figura 2.24: Filtros correlacionais aplicados em rastreamento de objetos.



Fonte: Adaptado de Zhang et al. (2018) e Zuo et al. (2019).

Os processos de rastreamento e treinamento da filtragem correlacional ocorrem em tempo real. O filtro é treinado a partir de imagens teste e instâncias deslocadas do alvo selecionado. A correlação é computada no domínio de Fourier utilizando a FFT (do inglês, *Fast Fourier Transformation*). Inicialmente computa-se a transformada 2D de Fourier na imagem de entrada e no filtro e, o resultado deste processo, é transformado de volta no domínio espacial utilizando a FFT inversa. O teorema da convolução afirma que a correlação no domínio de Fourier é realizada em multiplicações de elementos, fazendo com que o tempo de

processamento seja reduzido, garantindo maior velocidade na tarefa de rastreamento (BOLME et al., 2010).

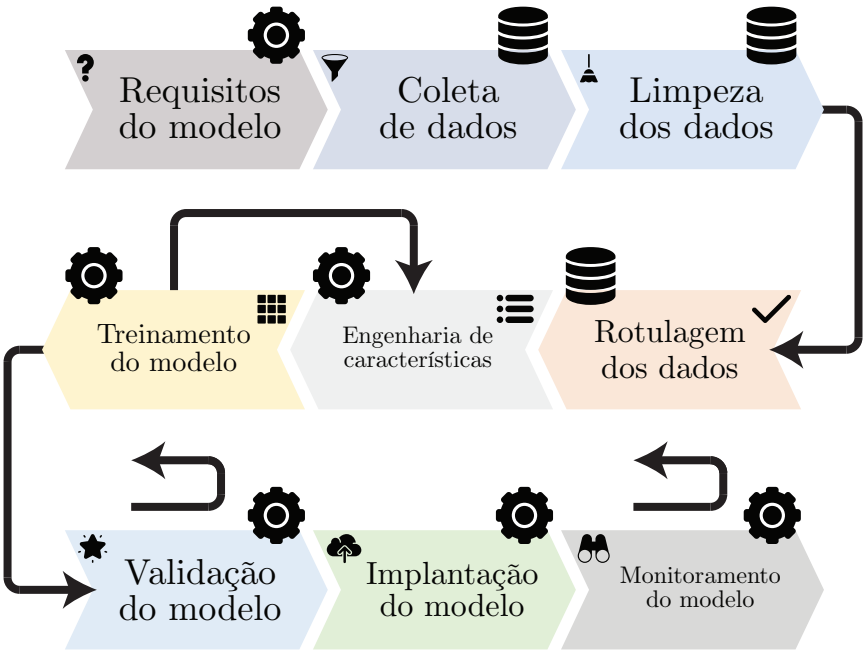
Com o decorrer dos anos esta técnica de filtros correlacionais foi aplicada e explorada de diversas formas, modificando e incrementando elementos, produzindo, assim, algoritmos rastreadores com grande acurácia e rápidos.

3 Metodologia

Este capítulo apresenta quais foram as técnicas e ferramentas adotadas para o desenvolvimento e aplicação do trabalho. São abordadas a metodologia utilizada para a escolha dos equipamentos, *frameworks* e bibliotecas aplicadas, captação de imagens para o treinamento das redes neurais convolucionais, processos de detecção e rastreamento, abrangendo as técnicas, algoritmos e suas particularidades, e sobre o desenvolvimento e execução do *framework* para detecção e rastreamento de peixes.

O desenvolvimento deste trabalho foi baseado nas metodologias de aprendizagem de máquina demonstradas em Amershi et al. (2019) e Google (2019). A Figura 3.1 apresenta um dos fluxos de trabalho utilizado neste trabalho.

Figura 3.1: Fluxo de trabalho para aprendizagem de máquina e seus estágios.



Fonte: Adaptado de Amershi et al. (2019).

Como observado na Figura 3.1, a aplicação da aprendizagem de máquina em um sistema é composta de nove estágios. Alguns destes são orientados a dados (ícone da base de dados) e os demais são orientados a modelo (ícone da engrenagem). Cada estágio é descrito a seguir:

- Requisitos do modelo: análise de viabilidade da implementação de recursos utilizando aprendizagem de máquina e se esses são úteis para a aplicação em produtos. Analisa-se também os tipos mais apropriados de modelos para atacar determinado problema;
- Coleta de dados: são pesquisados e integrados conjunto de dados (do inglês, *dataset*) disponíveis publicamente (*open-source*) ou coletados de maneira individual. Comumente utilizam-se modelos treinados parcialmente em conjuntos de dados genéricos (ImageNet ou Open Images Dataset, por exemplo) e, posteriormente, aplica-se a transferência de aprendizado para treinar e obter um modelo mais específico (para a detecção de ratos, por exemplo);
- Limpeza dos dados: removem-se os dados ruidosos ou imprecisos, limpando o conjunto de dados;
- Rotulagem dos dados: são atribuídos rótulos verdadeiros e absolutos para cada dado. Os rótulos são fornecidos por engenheiros, especialistas e parceiros. Alguns conjuntos de dados não possuem rótulos, para tanto é necessário rotulá-los. Na maioria das técnicas de aprendizagem de máquina a presença de rótulos é obrigatória;
- Engenharia de características: refere-se às atividades executadas para selecionar e extrair características para modelos de aprendizagem de máquina. Em alguns modelos (de redes neurais convolucionais, por exemplo) este estágio pode estar mesclado no próximo (treinamento);
- Treinamento do modelo: por meio dos modelos escolhidos, das características obtidas e de acordo com o conjunto de dados coletado e seus respectivos rótulos é realizada a operação de treinamento. É neste estágio que ocorre o processo de aprendizagem;
- Validação do modelo: avalia-se a capacidade do modelo de saída aplicando conjuntos de dados de testes e utilizando métricas pré-definidas;
- Implantação do modelo: o modelo é aplicado nos dispositivo(s) e/ou sistema(s);
- Monitoramento do modelo: monitora-se continuamente o modelo, analisando sua capacidade e possíveis erros que podem ocorrer durante sua execução.

O fluxo de trabalho da Figura 3.1 é apenas uma visão simplificada e linear do processo de aprendizagem de máquina, não abordando alguns possíveis loops

de realimentação existentes no processo. Alguns dos loops são apresentados pois consideram-se mais frequentes e comuns de serem realizados. As setas maiores do loop de realimentação indicam que o monitoramento e a validação do modelo podem voltar a quaisquer dos estágios anteriores, demonstrando uma possível correção em algum processo do modelo. A seta menor indica que o treinamento pode voltar ao estagio da engenharia de características, buscando uma melhor representação dos dados.

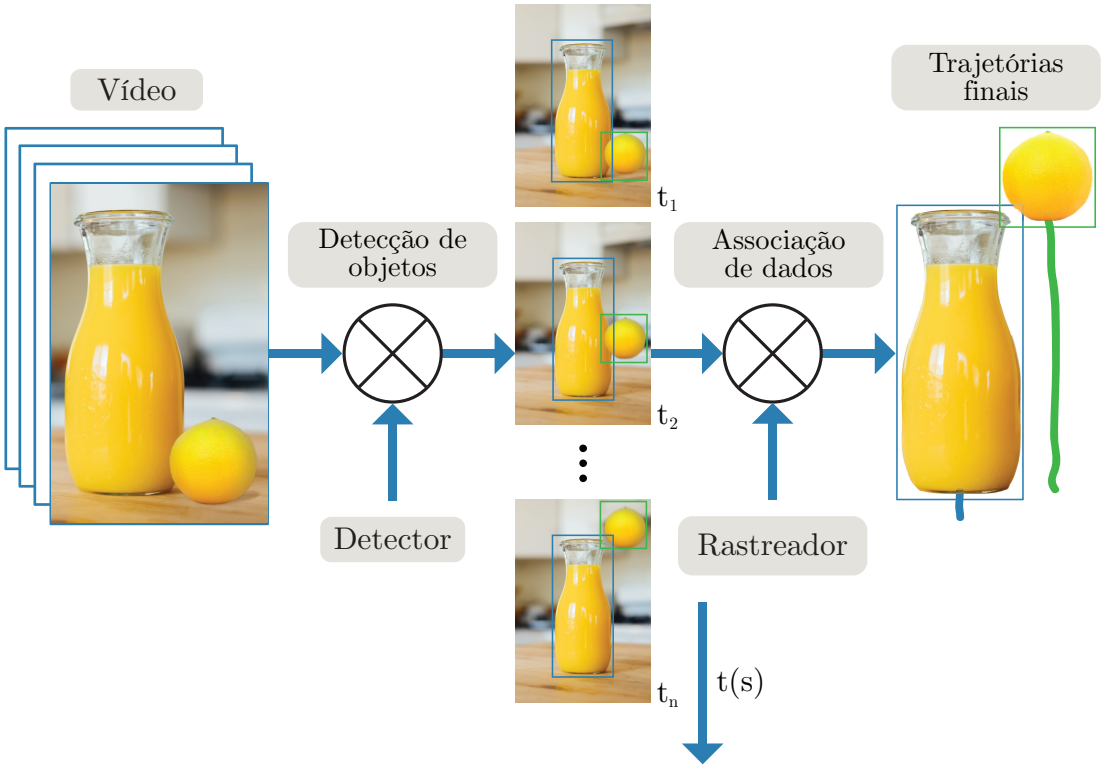
Este fluxo de trabalho pode se tornar cada vez mais complexo se houver múltiplos componentes de aprendizagem de máquina atuando paralelamente. Contudo, para o presente o trabalho e a existência de apenas uma instância de aprendizagem de máquina, o fluxo de trabalho apresentado é adotado, sendo considerado adequado e satisfatório para nortear o desenvolvimento do *framework*.

Outra metodologia adotada neste trabalho é a de rastreamento-por-detecção (do inglês, *tracking-by-detection*). Utilizando apenas detectores ou detectores e rastreadores, esta metodologia consiste em rastrear objetos em um vídeo por meio da detecção de objetos e associação de dados. Primeiramente, o detector atua em um quadro (do inglês, *frame*) individual, retornando a localização dos objetos no vídeo e, por último, ocorre nos próximos quadros o rastreamento ou a associação das detecções, permitindo compor trajetórias completas dos objetos (MILAN et al., 2016; FEICHTENHOFER; PINZ; ZISSERMAN, 2018).

Neste trabalho são utilizados detectores associados à rastreadores para obter a trajetória dos peixes. Um período fixo de detecção é definido (a cada 30, 50, 80 ou 110 *frames*), no qual a cada certa quantidade de *frames* percorridos realiza-se a detecção dos objetos presentes na cena pelo detector e nos demais são efetuados pelo rastreador o rastreamento, sendo este perpetuado até o próximo período de detecção. Sabendo da alta complexidade e custo computacional envolvido no processo de detecção se comparado ao rastreamento, preferiu-se adotar por esse tipo de metodologia. Na Figura 3.2 é ilustrado as etapas da metodologia de rastreamento-por-detecção.

Dado os avanços nas tarefas de detecção de objetos e rastreamentos de objetos, essa metodologia tornou-se largamente utilizada, principalmente devido a capacidade de resolução de problemas, custo computacional e bons resultados de precisão e acurácia. Contudo, como os algoritmos de detecção e rastreamento não são perfeitos, a união das tarefas mencionadas ou a associação de dados possuem desafios a serem superados e alguns destes são (MILAN et al., 2016; LEAL-TAIXE, 2016; FEICHTENHOFER; PINZ; ZISSERMAN, 2018):

Figura 3.2: Metodologia de rastreamento-por-detecção.



Fonte: Adaptado de Leal-Taixe (2016).

- Perdas de detecções e rastreamento: um objeto de interesse pode não ser detectado dado a capacidade do modelo (acurácia e precisão), impedindo a geração de sua trajetória pelo rastreador. Além disso, há o problema da oclusão, no qual um objeto é detectado mas é camuflado por outro objeto, prejudicando a sua detecção, assim como o rastreamento, podendo perder o objeto e compor uma trajetória incompleta;

- Alarme falso: um falso positivo (alarme falso) pode ser produzido pelo detector, captando uma região da imagem em que não corresponde ao objeto de interesse e, consequentemente, o rastreador também poderá produzir um alarme falso e gerar uma trajetória incongruente;

- Aparência semelhante: objetos de aparência similar podem trocar de trajetórias e de identificação, gerando, assim, uma informação incorreta;

- Associação correta dos dados: a metodologia utilizada para realizar a associação dos dados pode gerar trajetórias incorretas. Detecções realizadas em momentos distintos (a cada 20, 30 ou 50 quadros) podem impedir que o rastreador atue e complete a trajetória do objeto, assim como pode fazer com que o detector não localize novos objetos entre quadros;

Pesquisas estão sendo conduzidas visando sanar os desafios mencionados ante-

riormente, todavia, mesmo com os problemas apresentados, o rastreamento-por-detecção produz resultados promissores (BOCHINSKI; EISELEIN; SIKORA, 2017; SHKURTI et al., 2017; CHAHYATI; FANANY; ARYMURTHY, 2017; ALGETHAMI; REDFERN, 2020).

3.1 Equipamentos

Atualmente, pesquisadores de análises comportamentais utilizam computadores de mesa em conjunto com outros periféricos (placa de captura, câmera, mouse, teclado, monitor, entre outros), gerando uma grande quantidade de aparatos. Além destes necessitarem de uma infraestrutura para instalação, serem de médio porte e possuírem geralmente um custo relativamente elevado, há uma preocupação referente a escolha dos periféricos considerados corretos e adequados para o desenvolvimento dessas pesquisas. Portanto, visando beneficiar e auxiliar os pesquisadores em análises comportamentais, optou-se pelo utilização de um dispositivo embarcado. A escolha pela utilização e a aplicação do *framework* em um sistema embarcado é dado pelas suas seguintes vantagens: viabilidade econômica, durabilidade, obsolescência, praticidade, tamanho e peso.

Geralmente o custo agregado de um sistema embarcado é menor, tornando-o mais acessível para aquisição, assim como no caso de uma possível reposição (perda ou quebra do dispositivo, por exemplo). Possui um custo de prototipagem e desenvolvimento mais baixo, permitindo o desenvolvimento de uma maior quantidade de projetos sob medida e para nichos específicos. Comumente, para atender os requisitos das funcionalidades de uma aplicação e esta operar de forma adequada, não é necessário componentes de última geração e uma grande quantidade de recursos ou periféricos, no qual acarreta em um alto custo e desperdício de recursos. Um sistema embarcado, por outro lado, contém apenas o que é essencial para o funcionamento da aplicação e com componentes mais baratos. Além disso, por compreender e utilizar apenas os recursos necessários, garante, consequentemente, um baixo consumo energético (HOLT; HUANG, 2014; TOLLERVEY, 2017).

Os sistemas embarcados são desenvolvidos para terem uma longa durabilidade, e podem ter sua vida útil prolongada se forem acoplados a objetos robustos. Em diversas situações do cotidiano é possível notar dispositivos embarcados como calculadoras, alarmes, centrais telefônicas, catracas eletrônicas, entre outros, operando corretamente há vários anos (HOLT; HUANG, 2014; TOLLERVEY, 2017).

Em relação a obsolência, um sistema embarcado pode possuir uma vida útil maior se comparado a outros dispositivos como, por exemplo, um celular (a níveis de software). Dependendo da aplicação, a substituição de um dispositivo embarcado pode demorar a ocorrer, pois o mesmo ainda pode realizar a tarefa com êxito, seja devido sua durabilidade ou na questão de funcionalidades (HOLT; HUANG, 2014; TOLLERVEY, 2017).

Um sistema embarcado pode fornecer uma solução simplificada, completa e totalmente integrada, não havendo a necessidade do usuário se preocupar ou ter conhecimento de especificações complexas à níveis de software e hardware e que muitas vezes estão situadas fora do seu escopo de trabalho, permitindo, portanto, que se atente apenas se o dispositivo selecionado é capaz de resolver o problema da aplicação em questão (HOLT; HUANG, 2014; TOLLERVEY, 2017).

Outras vantagens existentes em uma ampla gama de sistemas embarcados são o tamanho e peso. Muitos dispositivos possuem dimensões compactas e de peso reduzido, podendo ser facilmente transportados e instalados nos mais variados locais (HOLT; HUANG, 2014; TOLLERVEY, 2017).

Tendo em vista os pontos levantados anteriormente, acredita-se que a utilização de um sistema embarcado é mais vantajosa e benéfica. Para tanto, uma pesquisa de mercado foi realizada, objetivando selecionar um dispositivo que abrange essas características e suportasse aplicações de aprendizagem de máquina (principalmente redes neurais convolucionais).

Vários dispositivos foram retornados nesta pesquisa, logo, tornou-se necessária a criação e implementação de filtros para escolher o dispositivo mais adequado. Os seguintes filtros foram utilizados: dimensões; peso; especificações técnicas, com maior ênfase na capacidade de processamento e memória RAM e portas de entrada/saída; documentação; e suporte a ferramentas de processamento de imagens e *frameworks* de aprendizagem de máquina. Considerando esses requisitos, os seguintes dispositivos foram pré-selecionados: NVIDIA Jetson Nano, Google Coral Dev Board e Raspberry Pi 3 B+. A seguir, na Tabela 3.1, são apresentadas as especificações consideradas mais relevantes dos três dispositivos.

Dentre os dispositivos apresentados, a placa NVIDIA Jetson Nano foi selecionada para a aplicação do *framework*. Seu conjunto de características faz com que essa seja a opção mais viável dentre as demais, devido principalmente a sua maior capacidade de memória RAM, por possuir uma GPU (do inglês, *Graphics Processing Unit*) com suporte ao CUDA (do inglês, *Compute Unified Device Architecture*) e o preço intermediário. A escolha foi baseada, em sua grande parte,

Tabela 3.1: Especificações mais importantes para os dispositivos disponíveis no mercado.

Especificação \ Dispositivo	**NVIDIA Jetson Nano**	Google Coral Dev Board	Raspberry Pi 3 Model B+
CPU	**ARM Cortex-A57**	ARM Cortex-A53	ARM Cortex-A53
GPU	**Maxwell 128 núcleos CUDA**	GC7000 Lite Gráficos Integrados	VideoCore IV Gráficos Integrados
RAM	**4 GB**	1 GB	1 GB
Acelerador MLE	**Aceleração por GPU**	Google Edge TPU	-
Dimensões (C x L x A mm)	**100 x 80 x 29**	88 x 60 x 24	82 x 56 x 19,5
Peso (g)	**140**	77	50
Preço (US$)	**99**	149	35

Fonte: Do autor.

por estes três fatores dadas suas crucialidades. A memória RAM, de baixa capacidade, causa problemas na execução e carregamento de diversos modelos de aprendizagem de máquina existentes (principalmente de redes neurais), retornando erros de falta de memória (OOF, do inglês, *Out Of Memory*), portanto, uma memória RAM de alta capacidade é essencial. No caso da GPU com suporte ao CUDA, o processamento do algoritmo de aprendizagem de máquina é acelerado, garantindo uma execução e treinamento mais rápido. Apesar de haver a TPU (Google Coral Dev Board), que também atua como um acelerador, de acordo com *benchmarks* apresentados pelos fabricantes, os resultados são regulares se comparados ao de uma GPU (NVIDIA Jetson Nano), além de, no caso da TPU, existirem limitações na aplicação e execução de alguns modelos.

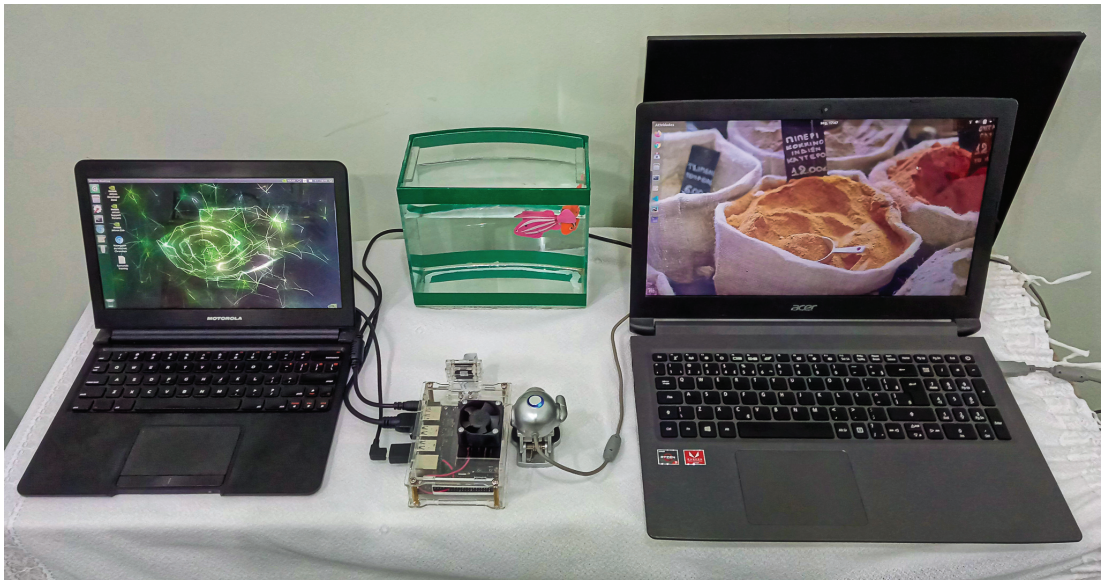
O desenvolvimento e execução do *framework*, assim como treinamento das redes (em nuvem), foram realizados em um computador pessoal (laptop) que possui uma CPU (do inglês, *Central Processing Unit*) AMD Ryzen 5 2500u, GPU Radeon Vega 8 (sem suporte ao CUDA), 12 GB de memória RAM e com o sistema operacional Linux Ubuntu 18.04 LTS x64 instalado.

O *framework* foi executado na NVIDIA Jetson Nano. Neste dispositivo foi utilizado um sistema operacional Linux denominado JetPack 4.3 (sistema proprietário disponível pela própria NVIDIA com base no Linux Ubuntu 18.04 LTS aarch64), que traz inclusas as ferramentas de aceleramento dos algoritmos de

aprendizagem de máquina: CUDA 10.0 e cuDNN 7.6.3. O acesso ao NVIDIA Jetson Nano pode ser feito de inúmeras formas, no entanto, como neste trabalho é necessário a inspeção visual da execução do *framework*, o acesso foi realizado por meio do *dock* Motorola Atrix.

Para testar o desempenho do *framework* desenvolvido, peixes robôs foram utilizados, inserindo-os em um aquário de pequeno porte e simulando testes natatórios. Referente a captura de imagens e vídeos dos peixes, uma câmera CSI Sony IMX219-77 foi acoplada no dispositivo NVIDIA Jetson Nano e no computador pessoal uma *webcam* USB foi utilizada. A Figura 3.3 apresenta os equipamentos utilizados.

Figura 3.3: Equipamentos de aplicação e desenvolvimento do *framework*.



Fonte: Do autor.

3.2 Dataset

Para o desenvolvimento de um *framework* com a implantação de técnicas de aprendizagem de máquina, um *dataset* é indispensável. Dificuldades como a elaboração e localização de um *dataset* adequado foram deparadas no decorrer do desenvolvimento deste trabalho. Devido à limitação de equipamentos e à dificuldade na captura adequada de imagens de peixes, optou-se por utilizar um *dataset* externo disponibilizado por especialistas da área de aprendizagem de máquina. Dentre os *datasets* que são estabelecidos, largamente utilizados, com uma grande quantidade de imagens distribuídas em várias classes, gratuitos (*open-source*) e que possuem a classe "peixe (*fish*)" em sua composição, foram encontrados o Open Images Dataset e o ImageNet.

O ImageNet (DENG et al., 2009) é um grande *dataset* designado para a utilização em pesquisas de reconhecimento visual de objetos. Mantido e criado pelas Universidade de Stanford e Universidade de Princeton, este *dataset* engloba mais de 14 milhões de imagens distribuídas em mais de 21 mil categorias. Para a categoria "peixe (*fish*)", que é o foco deste trabalho, constam 1.307 imagens, com apenas 365 anotações (caixas delimitadoras em torno dos objetos e seus devidos rótulos).

O Open Images Dataset (OID) (KRASIN et al., 2016), mantido e criado pela Google LLC, surgiu com o mesmo intuito do ImageNet - ser uma grande base de dados visual. Mais de 9 milhões de imagens distribuídas em 19.995 classes estão presentes no OID. No caso da categoria "peixe (*fish*)" e para a tarefa de detecção de objetos (dados anotados), estão disponíveis 5.434 imagens de treinamento, 340 imagens de validação e 914 imagens de teste.

Como o número de imagens presente na categoria "peixe (*fish*)" no OID é superior ao ImageNet, além da hospedagem, estabilidade e organização do site do OID também serem melhores, o OID foi o *dataset* escolhido para a utilização e aplicação. Foi utilizado neste trabalho o *dataset* OID versão 4 (GOOGLE, 2018).

Apesar do site do OID possuir as vantagens mencionadas e ter vários recursos, não existe a opção de download integral de uma única classe, ou seja, é necessário realizar o download manual das imagens e anotações ou do *dataset* completo. Dado esse problema, ferramentas de automatização foram desenvolvidas visando facilitar o processo de coleta dos dados no site do OID. A OIDv4 ToolKit de Vittorio (2018) é uma destas ferramentas e foi utilizada neste trabalho, permitindo o download personalizado de uma classe, assim como todas suas imagens e anotações relacionadas (teste, treinamento e validação).

Cada *dataset* possui um padrão de anotações para a aplicação no processo de detecção de objetos. No OID as anotações são compostas de:

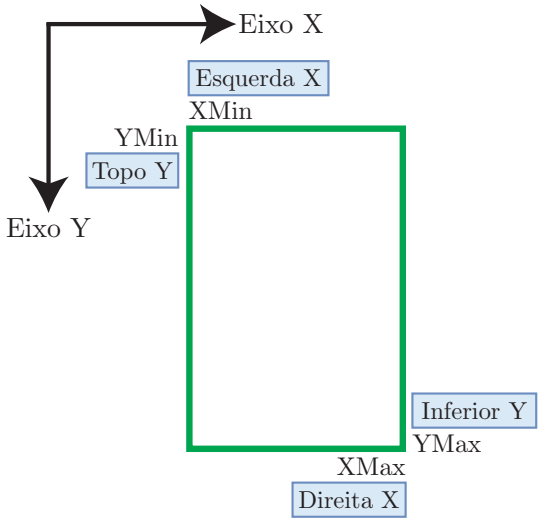
- ID da imagem: número de identificação de referência da imagem que possui a caixa delimitadora;
- Fonte: diz respeito ao modo (manual ou gerado pela máquina) de desenho das caixas delimitadoras;
- ID da *label*: código de identificação de referência do rótulo do objeto;
- Confiança: um valor fictício (*dummy*), sempre 1;

- XMin: coordenada (em pixels) em X localizada no canto superior esquerdo da caixa delimitadora;
- XMax: coordenada (em pixels) em X localizada no canto inferior direito da caixa delimitadora;
- YMin: coordenada (em pixels) em Y localizada no canto superior esquerdo da caixa delimitadora;
- YMax: coordenada (em pixels) em Y localizada no canto inferior direito da caixa delimitadora;
- Está ocluso: sinaliza (0 para falso e 1 para verdadeiro) se o objeto foi ocluso por outro objeto na imagem;
- Está truncado: sinaliza (0 para falso e 1 para verdadeiro) se o objeto ultrapassa os limites da imagem;
- É um grupo de: sinaliza (0 para falso e 1 para verdadeiro) se a caixa delimitadora engloba um grupo de objetos;
- É uma representação: sinaliza (0 para falso e 1 para verdadeiro) se o objeto na imagem é uma representação (caricatura ou desenho);
- Está dentro: sinaliza (0 para falso e 1 para verdadeiro) se a foto foi tirada a partir do interior do objeto.

No processo de treinamento de certas redes neurais, muitos destes parâmetros tornam-se opcionais. Geralmente apenas os parâmetros de ID da *label* e as coordenadas da caixa delimitadora (XMin, XMax, YMin, YMax) são utilizados. A OIDv4 ToolKit permite a escolha personalizada dos parâmetros, sendo efetuado o download da anotação de acordo com a necessidade do usuário. A Figura 3.4 apresenta o formato de download padrão das anotações realizadas pela OIDv4 ToolKit: nome da classe, esquerda, topo, direita, inferior.

Além disso, para algumas redes, é necessário a normalização dos parâmetros, como é o caso das coordenadas, que, para o treinamento correto das CNN, deve estar na faixa de valores entre 0 e 1. O OIDv4 ToolKit possui a funcionalidade de normalização implementada, sendo esta utilizada neste trabalho.

Figura 3.4: Formato de anotação utilizado.



Fonte: Adaptado de Vittorio (2018).

3.3 Detecção

Nesta seção são apresentadas as CNN escolhidas para a realização da tarefa de detecção dos peixes. Critérios foram criados para um processo adequado de escolha das redes, dentre os quais:

- Linguagem de programação suportada: é comum serem aplicadas as linguagens C, Python e MATLAB para o treinamento e aplicação de redes neurais convolucionais. Dado o atual crescimento e popularidade no uso de Python na área de aprendizagem de máquina, por ser multiplataforma, gratuita, facilmente legível, simplificada e disponibilizar uma grande diversidade de bibliotecas, essa foi a linguagem selecionada;

- Quantidade de memória e poder de processamento necessários para o carregamento da rede e a realização do processo de inferência: a partir das redes popularmente aplicadas no processo de detecção de objetos, avaliou-se suas estruturas e o tamanho do arquivo de pesos, no qual notou-se que estes são diretamente proporcionais à quantidade de processamento e memória exigidos para a correta operação. Portanto, foram priorizadas as CNN mais populares na literatura e que possuíam uma estrutura e arquivo de pesos reduzidos, resultando na escolha das redes YOLOv3, YOLOv3-tiny e Mobilenet-V2;

- Disponibilidade e *frameworks*: existem diversos *frameworks* para o treinamento e execução de CNN como o TensorFlow, PyTorch, CNTK, Caffe e Darknet. Porém, para as CNN e a linguagem escolhida, optou-se pela

utilização do TensorFlow (Mobilenet-V2) e Darknet (YOLOv3 e YOLOv3-tiny) devido estes apresentarem uma boa documentação, compatibilidade (redes e *dataset*) e facilidade de uso;

- Facilidade de treinamento e ajuste de hiperparâmetros: a possibilidade de realizar diferentes ajustes nos hiperparâmetros e efetuar treinamentos de uma maneira prática e rápida, permite extrair bons resultados das CNN.
- Capacidade e flexibilidade no processo de integração de detectores no *framework*: algumas soluções são integradas ao próprio *framework*, limitando o desenvolvimento de um sistema customizado, todavia, as CNN e *frameworks* escolhidos, permitem sua integração sem maiores problemas.

Nas próximas seções são apresentadas uma visão geral das CNN, arquiteturas e suas particularidades, assim como os ajustes realizados para o treinamento e aplicação no *framework*.

3.3.1 YOLOv3

A rede neural convolucional YOLO (do inglês, *You Only Look Once*) foi desenvolvida com o intuito de ser uma nova abordagem para a tarefa de detecção de objetos. Sua ideia base é tratar esta tarefa como um problema de regressão, obtendo as caixas delimitadoras e suas probabilidades associadas de uma única vez. Desde Redmon et al. (2015) aprimoramentos continuam sendo aplicados nesta rede visando melhorar sua acurácia e performance, no qual novas versões foram desenvolvidas até chegarmos na YOLOv3 de Redmon e Farhadi (2018).

Na YOLOv3, a estrutura da rede, denominada de Darknet-53, é maior que as anteriores, possuindo 53 camadas convolucionais. Além disso, outros recursos foram adicionados: predições das caixas delimitadoras utilizando regressão logística, retornando a pontuação de cada objeto relacionado com sua caixa delimitadora; predições de classes por meio de classificadores logísticos, permitindo uma classificação multi-rótulo aos objetos; predições em escalas, no qual realiza-se a extração das caixas delimitadoras preditas em 3 escalas diferentes e utilizando um conceito similar ao de redes de pirâmides de características (do inglês, *feature pyramid networks*) - a partir de uma imagem de entrada são extraídas características em escalas diferentes, visando ampliar a atuação da rede em várias escalas e extrair um maior número de características.

A YOLOv3 é considerada uma rede neural convolucional rápida e com boa acurácia, possuindo resultados próximos ao de outras redes do estado da arte

como visto nos resultados apresentados em Redmon e Farhadi (2018). A Tabela 3.2 apresenta a arquitetura da rede.

Tabela 3.2: Arquitetura da rede neural convolucional YOLOv3.

Multiplicador	Operação	Filtros	Tamanho/Passo	Saída
	Convolução	32	3 x 3 / 1	256 x 256
	Convolução	64	3 x 3 / 2	128 x 128
	Convolução	32	1 x 1 / 1	
1x	Convolução	64	3 x 3 / 1	
	Residual			128 x 128
	Convolução	128	3 x 3 / 2	64 x 64
	Convolução	64	1 x 1 / 1	
2x	Convolução	128	3 x 3 / 1	
	Residual			64 x 64
	Convolução	256	3 x 3 / 2	32 x 32
	Convolução	128	1 x 1 / 1	
8x	Convolução	256	3 x 3 / 1	
	Residual			32 x 32
	Convolução	512	3 x 3 / 2	16 x 16
	Convolução	256	1 x 1 / 1	
8x	Convolução	512	3 x 3 / 1	
	Residual			16 x 16
	Convolução	1024	3 x 3 / 2	8 x 8
	Convolução	512	1 x 1 / 1	
4x	Convolução	1024	3 x 3 / 1	
	Residual			8 x 8
	Avgpool		Global	
	Totalmente Conectada		1000	
	Softmax (Classificador)			

Fonte: Adaptado de Redmon e Farhadi (2018).

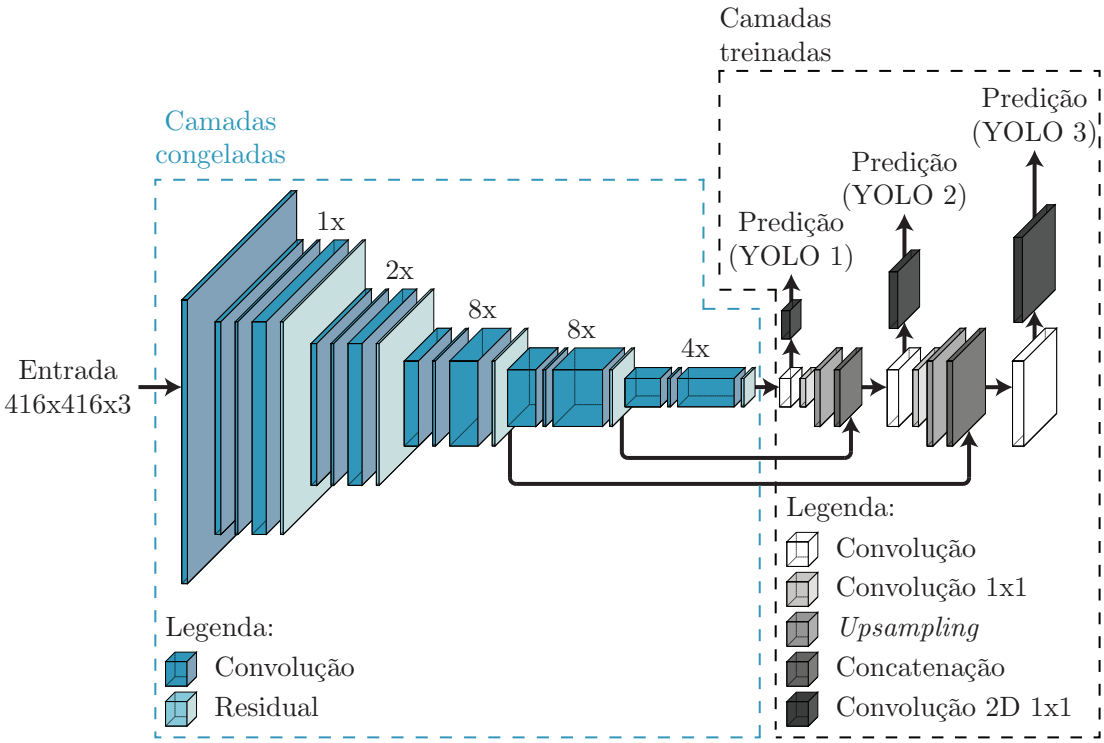
O treinamento da YOLOv3 foi realizado no *framework* Darknet, desenvolvido pelo próprio criador da rede. Apesar de existirem adaptações dessa rede para outros *frameworks*, optou-se por utilizar o Darknet (nativo das redes YOLO) por

questões de compatibilidade e a fim de evitar erros.

A técnica de transferência de aprendizado (do inglês, *transfer learning*) foi aplicada, dado que pesos randômicos podem prejudicar o aprendizado de características, prolongar o tempo de treinamento e ignorar pesos de características benéficas aprendidas previamente em um processo de treinamento em outro *dataset*.

Complementarmente foi aplicada a técnica de ajuste fino (do inglês, *fine-tuning*), que consiste em realizar o congelamento (do inglês, *freezing*) de camadas inferiores da rede neural para preservar o aprendizado (características extraídas, pesos) de um treinamento prévio e treinar apenas as camadas superiores, aprendendo nestas novas abstrações e, consequentemente, adaptando-se à nova aplicação. A utilização da técnica de ajuste fino garante uma boa acurácia com um menor tempo de treinamento e gera bons resultados em pequenos *datasets* (CHOLLET, 2017). De acordo com a documentação em Redmon (2020) foi aplicado o ajuste fino na YOLOv3 e na Figura 3.5 é apresentado as camadas congeladas e treinadas.

Figura 3.5: Aplicação da técnica de ajuste fino na rede neural convolucional YOLOv3.



Fonte: Do autor.

Os arquivos de configuração da rede, pesos pré-treinados e da lista de classes foram obtidos no site do *framework* Darknet. O arquivo de configuração da rede e da lista de classes está ajustado para o *dataset* MS COCO, portanto, é

necessário adequar ao *dataset* de classe única "peixe (*fish*)" que é o utilizado neste trabalho. Os hiperparâmetros localizados no arquivo de configuração da rede foram ajustados conforme as orientações descritas na documentação do *framework* Darknet de Redmon (2020). A Tabela 3.3 apresenta os hiperparâmetros que foram modificados.

Tabela 3.3: Valores ajustados dos hiperparâmetros para o treinamento da rede neural convolucional YOLOv3.

Hiperparâmetro	Valor
Batch	64
Subdivisions	16
Width	416
Height	416
Burn-in	400
Max batches	4000
Steps	3200, 3600
Filters (Camadas 81, 93 e 105)	18
Classes (Camadas 82, 94 e 106)	1

Fonte: Do autor.

Os hiperparâmetros da Tabela 3.3 são descritos como:

- *Batch*: refere-se à um lote composto por um número de imagens para o cálculo da perda média. Um valor alto de *batch* garante uma maior capacidade de generalização, porém, uma memória de alta capacidade é necessária;
- *Subdivisions*: refere-se à divisão do lote em mini lotes (*sub-batches*) a fim de diminuir a demanda de memória exigida no processo de treinamento;
- *Width*: resolução (em pixels) referente ao comprimento das imagens de entrada na rede. Utilizada geralmente para aumentar a precisão da rede e detectar pequenos objetos, porém, quanto maior este valor, maior será o impacto no tempo de treinamento, consumo de memória e execução/teste;
- *Height*: resolução (em pixels) referente a altura das imagens de entrada na rede. O valor para este hiperparâmetro deve ser igual ao do *width*;
- *Burn-in*: controla a taxa de aprendizagem (do inglês, *learning rate*), a fim de garantir a estabilização ("aquecimento") e acelerando o treinamento;
- *Max batches*: número máximo de iterações que será realizado no processo de treinamento da rede neural. A documentação indica que este valor é

definido como:

$$Max\ batches = Classes * 2000, \tag{3.1}$$

não podendo ser menor que 4000.

- *Steps*: ajusta a *learning rate* multiplicando-a por um escalar após um determinado número de iterações;
- *Filters*: número de filtros que serão produzidos na camada de convolução. Na documentação este valor é dado como:

$$Filters = (Classes + 5) * 3. \tag{3.2}$$

- *Classes*: quantidade de classes de objetos que serão aplicadas no processo de treinamento.

Por meio do download apenas da classe "peixe (*fish*)" presente no *dataset* OID v4, adaptações foram realizadas nas anotações, visando o treinamento adequado da rede. O padrão de anotações exigido pelo Darknet e utilizado para cada imagem é o correspondente ao demonstrado na Figura 3.4: nome da classe, esquerda, topo, direita, inferior.

Para executar o processo de treinamento da YOLOv3 utilizou-se o serviço em nuvem Google Colab da Google LLC, no qual 12 horas de GPU de alta performance (GPU's da linha NVIDIA Tesla, variando entre os modelos K80, T4, P4 e P100) são disponibilizadas gratuitamente para a aplicação de algoritmos de aprendizagem de máquina. O *dataset* referente à classe "peixe (*fish*)" e os arquivos relacionados a rede YOLOv3 foram carregados na nuvem utilizando o Google Drive e permitindo o acesso à esses pelo Google Colab. Instalações de dependências e do *framework* Darknet, assim como as suas devidas configurações, foram efetuadas, para que o processo de treinamento pelo Colab pudesse ser realizado.

3.3.2 YOLOv3-tiny

A YOLOv3-tiny é uma versão compacta da YOLOv3. Desenvolvida para a aplicação em dispositivos com baixo poder computacional (limitados), a YOLOv3-tiny permite que estes dispositivos realizem a detecção de objetos em tempo real de forma mais rápida (comparadas à outras CNN existentes) com apenas pequenos sacrifícios na acurácia (HE et al., 2019). A Tabela 3.4 apresenta a arquitetura da YOLOv3-tiny.

Tabela 3.4: Arquitetura da rede neural convolucional YOLOv3-tiny.

Camada	Operação	Filtros	Tamanho/Passo	Entrada	Saída
0	Convolução	16	3 x 3 / 1	416 x 416 x 3	416 x 416 x 16
1	*Maxpool*	-	2 x 2 / 2	416 x 416 x 16	208 x 208 x 16
2	Convolução	32	3 x 3 / 1	208 x 208 x 16	208 x 208 x 32
3	*Maxpool*	-	2 x 2 / 2	208 x 208 x 32	104 x 104 x 32
4	Convolução	64	3 x 3 / 1	104 x 104 x 32	104 x 104 x 64
5	*Maxpool*	-	2 x 2 / 2	104 x 104 x 64	52 x 52 x 64
6	Convolução	128	3 x 3 / 1	52 x 52 x 64	52 x 52 x 128
7	*Maxpool*	-	2 x 2 / 2	52 x 52 x 128	26 x 26 x 128
8	Convolução	256	3 x 3 / 1	26 x 26 x 128	26 x 26 x 256
9	*Maxpool*	-	2 x 2 / 2	26 x 26 x 256	13 x 13 x 256
10	Convolução	512	3 x 3 / 1	13 x 13 x 256	13 x 13 x 512
11	*Maxpool*	-	2 x 2 / 1	13 x 13 x 512	13 x 13 x 512
12	Convolução	1024	3 x 3 / 1	13 x 13 x 512	13 x 13 x 1024
13	Convolução	256	1 x 1 / 1	13 x 13 x 1024	13 x 13 x 256
14	Convolução	512	3 x 3 / 1	13 x 13 x 256	13 x 13 x 512
15	Convolução	255	1 x 1 / 1	13 x 13 x 512	13 x 13 x 255
16	YOLO	-	-	-	-
17	**Rota para 13**	-	-	-	-
18	Convolução	128	1 x 1 / 1	13 x 13 x 256	13 x 13 x 128
19	*Up-sampling*	-	2 x 2 / 1	13 x 13 x 128	26 x 26 x 128
20	**Rota para 19 e 8**	-	-	-	-
21	Convolução	256	3 x 3 / 1	13 x 13 x 384	13 x 13 x 256
22	Convolução	255	1 x 1 / 1	13 x 13 x 256	13 x 13 x 255
23	YOLO	-	-	-	-

Fonte: Adaptado de He et al. (2019)

O processo de treinamento da YOLOv3-tiny é realizado da mesma forma que a YOLOv3, com exceções apenas nos seguintes aspectos: os arquivos de configuração e pesos pré-treinados utilizados, que são diferentes da YOLOv3; e os valores dos hiperparâmetros. A Tabela 3.5 apresenta os novos valores de hiperparâmetros.

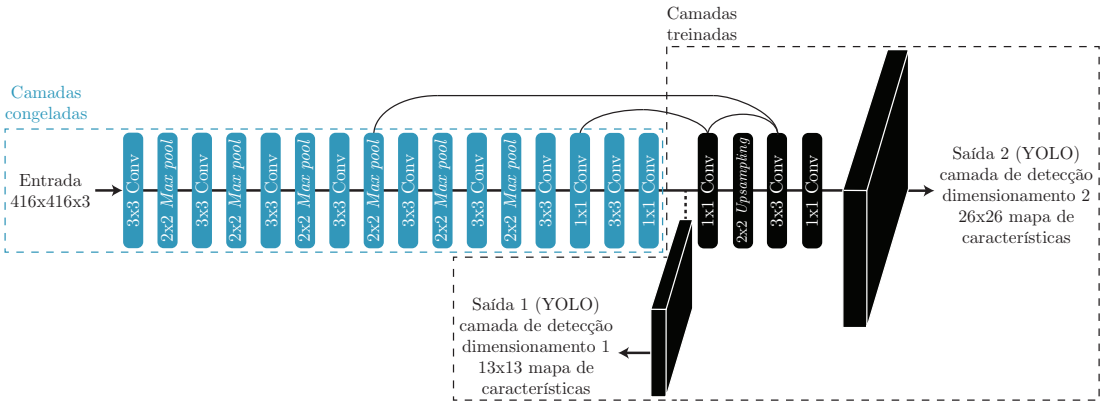
Tabela 3.5: Valores ajustados dos hiperparâmetros para o treinamento da rede neural convolucional YOLOv3-tiny.

Hiperparâmetro	Valor
Batch	64
Subdivisions	4
Width	416
Height	416
Burn-in	400
Max batches	4000
Steps	3200, 3600
Filters (Camadas 15 e 22)	18
Classes (Camadas 16 e 23)	1

Fonte: Do autor.

Assim como aplicado na YOLOv3, na YOLOv3-tiny também optou-se por utilizar a técnica de ajuste fino. Na Figura 3.6 são demonstradas as camadas congeladas e treinadas.

Figura 3.6: Aplicação da técnica de ajuste fino na rede neural convolucional YOLOv3-tiny.



Fonte: Do autor.

3.3.3 MobileNetV2

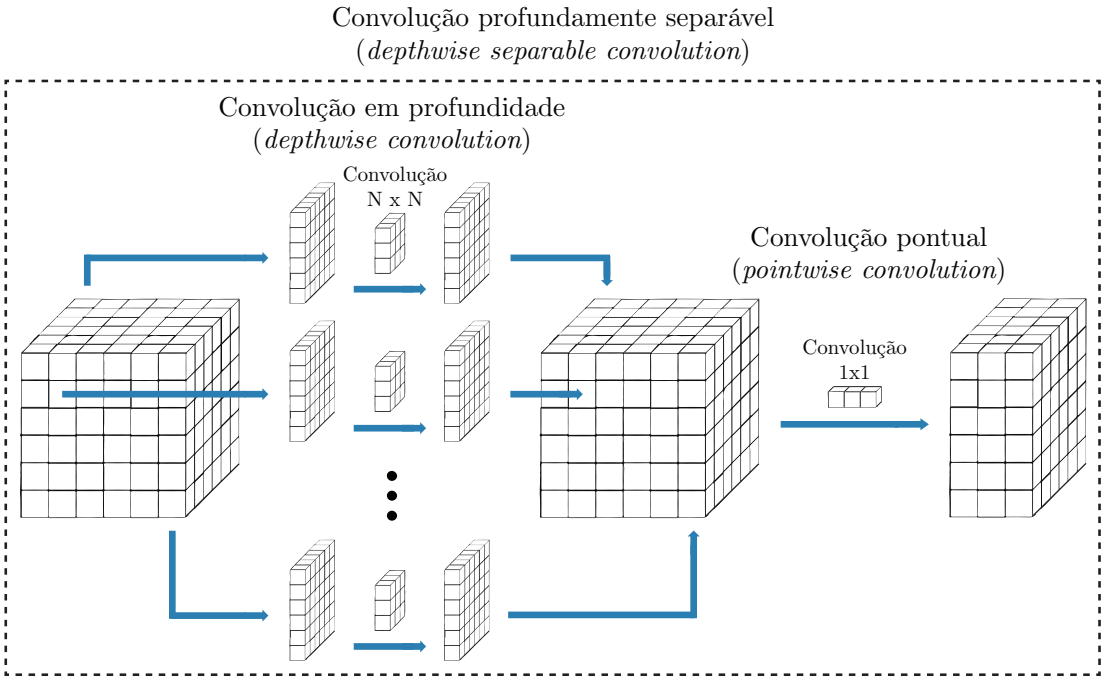
As MobileNets, desenvolvidas pela Google, são arquiteturas de rede neural convolucional pequenas, com níveis de acurácia competitivos e de baixa latência com o uso voltado para dispositivos (móveis e embarcados) de baixo poder computacional. Estas arquiteturas são passíveis de aplicação em diversas tarefas, dentre elas: classificação, detecção e segmentação (HOWARD et al., 2017). A MobileNetV2, utilizada neste trabalho, é baseada no método SSD (do inglês, *Single Shot Detector*) e na rede MobileNetV1 (versão anterior).

O SSD, como seu próprio nome faz referência, realiza a detecção dos objetos (por meio da aplicação de pequenos filtros convolucionais nos mapas de características, retornam-se as coordenadas das caixas delimitadoras e porcentagens relacionados as classes dos objetos presentes) com apenas uma única propagação direta pela rede durante o processo de inferência. Esta operação direta é o que faz o SSD ser rápido e eficiente. Além disso, o SSD possui uma alta acurácia, que é garantida por meio de predições obtidas da geração de mapas de características de diferentes dimensões, permitindo que objetos de tamanhos distintos sejam detectados (LIU et al., 2016b; ROSEBROCK, 2017).

A MobileNetV1 é uma arquitetura desenvolvida com base na técnica de convolução profundamente separável (do inglês, *depthwise separable convolution*), que é uma fatoração da convolução padrão e consiste na combinação da convolução em profundidade (*depthwise convolution*) seguida pela convolução pon-

tual (*pointwise convolution*). A convolução em profundidade, conhecida também como etapa de filtragem, refere-se à aplicação da convolução por canais na qual aplica-se um único filtro para cada canal de entrada. A convolução pontual, conhecida também como etapa de combinação, é a aplicação de uma convolução de dimensão 1x1, combinando as saídas da convolução em profundidade. A aplicação da técnica de convolução profundamente separável, comparado com a convolução padrão, garante um menor custo computacional com uma pequena redução da acurácia (HOWARD et al., 2017). A Figura 3.7 ilustra as convoluções em profundidade e pontual.

Figura 3.7: Convolução profundamente separável: convolução em profundidade e convolução pontual.



Fonte: Adaptado de Howard et al. (2017).

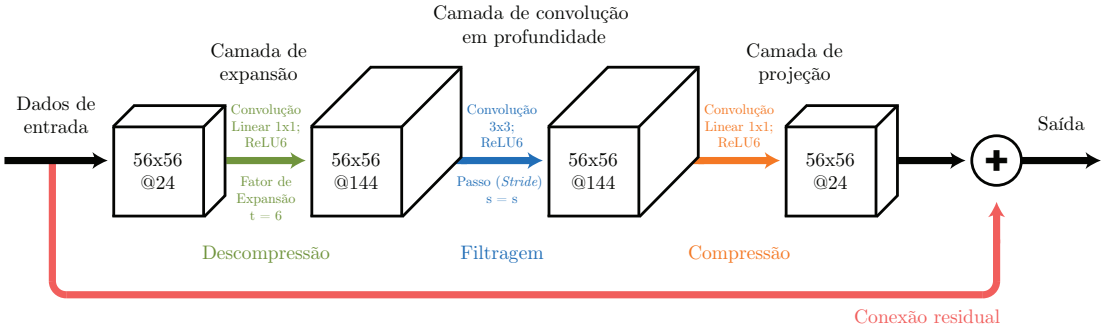
Além disso, a MobileNetV1 apresenta dois hiperparâmetros ajustáveis: multiplicador de largura (do inglês, *width multiplier*) e multiplicador de resolução (do inglês, *resolution multiplier*). O multiplicador de largura (α, valor entre 0 e 1) afina a rede uniformemente em cada camada, tornando manipulável o tamanho dos canais de entrada e saída e, para o caso do multiplicador de resolução (ρ, valor entre 0 e 1), controla-se a resolução da imagem de entrada. Ambos os parâmetros foram desenvolvidos visando a diminuição da complexidade da rede para determinadas aplicações, reduzindo o número de parâmetros e o custo computacional, tornando, assim, a rede enxuta e mais rápida (HOWARD et al., 2017).

A MobileNetV2 foi desenvolvida seguindo o mesmo conceito da MobileNetV1, porém, introduzindo duas novas técnicas para a arquitetura: a camada linear de gargalo (do inglês, *linear bottleneck layers*) e a conexão de desvio entre os

gargalos (do inglês, *bottleneck*). A camada linear de gargalo é assim denominada pois têm como finalidade reduzir o fluxo de dados propagados pela rede. Além disso, a linearidade desta camada preserva a representação dos dados, evitando a destruição de informações. A conexão de desvio entre os gargalos permite que o gradiente propague-se pela rede, fazendo que a rede aprenda mais e, assim, tenha um ganho de acurácia (SANDLER et al., 2018).

O conjunto destas técnicas resultou no desenvolvimento do bloco de gargalo residual (do inglês, *bottleneck residual block*). Este bloco é composto por uma camada de expansão, seguido por uma camada de convolução em profundidade e, por fim, uma camada de projeção. Na Figura 3.8 é demonstrado um exemplo do bloco de gargalo residual com as novas técnicas inseridas e aplicadas na arquitetura da MobileNetV2.

Figura 3.8: Bloco de gargalo residual: camada de expansão, camada de convolução em profundidade e camada de projeção.



Fonte: Adaptado de Howard et al. (2017) e Hollemans (2020).

A camada de expansão é composta por uma convolução 1x1 seguida da função de ativação ReLU6 e esta camada têm como objetivo expandir o número de canais dos dados de entrada, sendo esta expansão controlada por meio de um hiper parâmetro ajustável denominado de fator de expansão (t). Esta operação visa expandir os dados de entrada para ser possível extrair uma maior quantidade de características pela próxima camada (convolução em profundidade). Nesta, aplica-se uma convolução em profundidade de 3x3 seguida pela função de ativação ReLU6, realizando a filtragem dos dados. Por último, a camada de projeção realiza uma convolução linear de 1x1 projetando (comprimindo) os canais, tornando-os pequenos e propagando-os para as próximas camadas da rede (HOWARD et al., 2017; HOLLEMANS, 2020).

Na Tabela 3.6 é apresentada a arquitetura completa da MobileNetV2. Devido a aplicação das novas técnicas mencionadas anteriormente, a MobileNetV2 alcançou resultados melhores que a MobileNetV1: menor custo computacional (número de parâmetros reduzido); maior acurácia; e maior eficiência.

Tabela 3.6: Arquitetura da rede neural convolucional MobileNetV2.

Entrada (resolução, em pixels)	Operação	Fator de expansão (t)	Canais de saída (c)	Número de repetições da mesma camada (n)	Passo (s)
224x224x3	conv2d	-	32	1	2
112x112x32	bottleneck	1	16	1	1
112x112x16	bottleneck	6	24	2	2
56x56x24	bottleneck	6	32	3	2
28x28x32	bottleneck	6	64	4	2
14x14x64	bottleneck	6	96	3	1
14x14x96	bottleneck	6	160	3	2
7x7x160	bottleneck	6	320	1	1
7x7x320	conv2d 1x1	-	1280	1	1
7x7x1280	avgpool 7x7	-	-	1	-
1x1x1280	conv2d 1x1	-	k	-	-

Fonte: Adaptado de Sandler et al. (2018)

O treinamento da MobileNetV2 também foi realizado em seu *framework* nativo, que neste caso é o TensorFlow. Para realizar o treinamento dessa rede e treinar outros modelos de detecção de objetos no TensorFlow, é obrigatório realizar o download da API (do inglês, *Application Programming Interface*) de detecção de objetos (do inglês, *Object Detection API*). Esta API engloba diversas ferramentas como: treinamento, validação e teste para diversas redes neurais; processamento, manipulação e tratamento de *dataset*; entre outras. Além disso, a API de detecção de objetos também possui arquivos base de configuração das redes neurais. Por meio dessa, foi obtido o arquivo de configuração da CNN MobileNetV2, no qual também foram realizados os devidos ajustes dos hiperparâmetros (número de classes, resolução em comprimento e altura das imagens de entrada, lotes e o número máximo de iterações para o processo de treinamento) e que podem ser visualizados na Tabela 3.7.

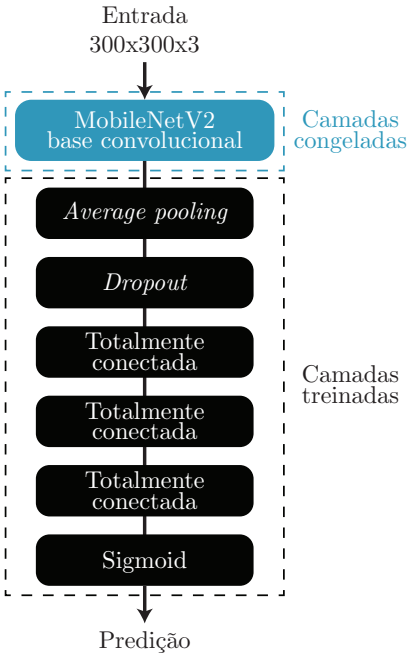
Tabela 3.7: Valores ajustados dos hiperparâmetros para o treinamento da rede neural convolucional MobileNetV2.

Hiperparâmetro	Valor
Num. classes	1
Width	300
Height	300
Batches	64
Num. steps	10000

Fonte: Do autor.

Semelhante a YOLOv3 e a YOLOv3-tiny, na MobileNetV2 também foi utilizado a técnica de ajuste fino. Na Figura 3.9 são apresentadas as camadas congeladas e treinadas referentes a aplicação do ajuste fino.

Figura 3.9: Aplicação da técnica de ajuste fino na rede neural convolucional MobileNetV2.



Fonte: Do autor.

O *dataset* da classe "peixe (*fish*)" do OID v4 possui seu próprio formato de anotações e, para que esse *dataset* pudesse ser utilizado no treinamento da MobileNetV2 no TensorFlow, foi necessário convertê-lo para o formato proprietário do *framework* denominado de TFRecord. Este formato de dados armazena uma sequência de registros binários, salvando-os em um conjunto de arquivos que podem ser lidos linearmente e que garantem, assim, uma leitura eficiente dos dados.

A execução do processo de treinamento também foi realizada na plataforma do Google Colab, no qual foram instaladas as dependências necessárias, o *framework* do TensorFlow e a API de detecção de objetos. O *dataset* convertido e o arquivo de configuração da rede foram carregados para o Google Drive, com acesso permitido ao Google Colab.

3.3.4 Métricas de detecção

Para avaliar o desempenho de redes neurais e, consequentemente, o processo de detecção de objetos, métricas são utilizadas. No decorrer dos anos diversas métricas foram desenvolvidas e algumas delas tornaram-se predominantes, sendo aplicadas em larga escala na literatura, como é o caso da IoU (*Intersect Over Union*), *Precision* (precisão), Recall (revocação) e AP (*Average Precision*).

A IoU é aplicada a fim de mensurar a acurácia do detector de objetos (rede neural). A partir das caixas delimitadoras obtidas no processo de detecção (pre-

ditas) e as verdadeiras (humanamente desenhada, *ground truth*), as seguintes relações são obtidas: área de união (área total, englobando a área da caixa delimitadora detectada e a área de *ground truth*) e área de sobreposição (área de intersecção entre a área da caixa delimitadora detectada e a área de *ground truth*). Cada relação retorna valores, dentre os quais são utilizados para efetuar o cálculo do IoU utilizando a relação dada por Everingham et al. (2010),

$$IoU = \frac{A \cap B}{A \cup B} * 100, \tag{3.3}$$

sendo A as caixas delimitadoras preditas e B o *ground truth*.

A *Precision* corresponde ao valor das caixas delimitadoras preditas que estão realmente corretas. Everingham et al. (2010) define como,

$$Precision = \frac{A \cap B}{A}, \tag{3.4}$$

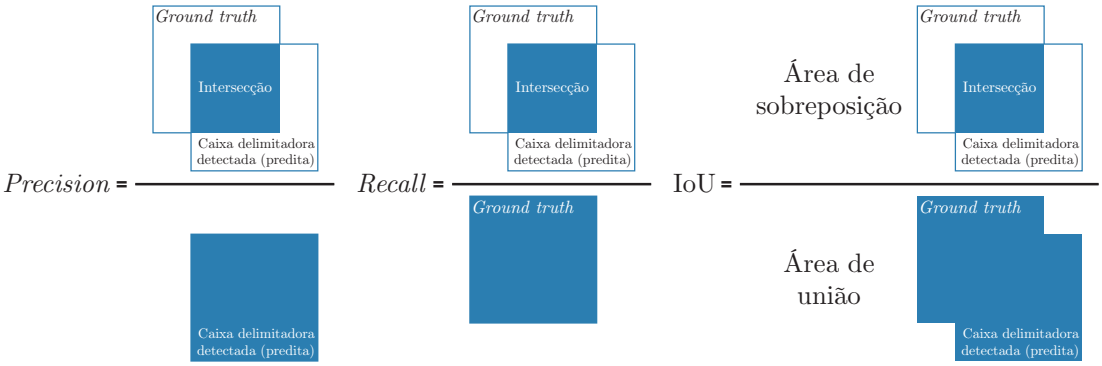
sendo A as caixas delimitadoras preditas e B o *ground truth*.

Recall é a métrica que faz referência à quantia de objetos detectados verdadeiramente positivos, descrevendo a integridade destas de acordo com o *ground truth*. Em Everingham et al. (2010) é define-se como,

$$Recall = \frac{A \cap B}{B}. \tag{3.5}$$

sendo A as caixas delimitadoras preditas e B o *ground truth*. A Figura 3.10 ilustra visualmente, para um fácil entendimento, as métricas *Precision*, Recall e IoU.

Figura 3.10: Representação gráfica das métricas de *Precision*, Recall e IoU.



Fonte: Adaptado de Redmon (2016).

Por fim, a AP corresponde ao valor médio de 11 pontos sobre a curva PR (*Precision*-Recall) para cada limite possível (cada probabilidade de detecção) para a mesma classe. Everingham et al. (2010) definem AP como sendo,

$$AP = \frac{1}{11} \sum_{r\epsilon\{0,0;0,1;...;1,0\}} p_{interp}(r) \tag{3.6}$$

no qual, $p_{interp}(r)$ é a máxima precisão interpolada para quaisquer valores de Recall maior que r, sendo r o conjunto de 11 níveis de Recall igualmente espaçados [0,0; 0,1;...;1].

3.4 Rastreamento

Nesta seção os rastreadores utilizados na tarefa de rastreamento de objetos são apresentados. Critérios similares aos da tarefa de detecção de objetos também são aqui utilizados para a escolha dos rastreadores, dentre os quais:

- Linguagem de programação suportada: a tarefa de rastreamento de objetos pode ser implementada em diversas linguagens. A fim de manter padronizado o *framework* desenvolvido, além de aproveitar as vantagens propiciadas pela linguagem Python, esta também foi adotada para o rastreamento;
- Quantidade de memória e poder de processamento necessários para a execução do algoritmo: algoritmos de rastreamento, ao contrário dos algoritmos de detecção, exigem (geralmente) um menor recurso computacional. No entanto, ainda assim, optou-se pela escolha de algoritmos com modelagens matemáticas simples e com operações menos custosas;
- Disponibilidade e *frameworks*: existem inúmeros algoritmos de rastreamento de objetos, no entanto, muitos possuem as limitações de: disponibilidade; linguagem de programação; e implementação e integração complexa, difícil e não intuitiva (falta de documentação). Portanto, optou-se por utilizar algoritmos estabelecidos, como o dlib e os que estão presentes na própria biblioteca OpenCV. Entre os algoritmos de rastreamento presentes no OpenCV, testes empíricos foram realizados e, apenas os algoritmos que utilizam filtros correlacionais (MOSSE, KCF e CSRT) em sua base, apresentaram um bom desempenho, tanto em velocidade quanto em acurácia, conseguindo rastrear, manipular oclusões e mudanças de objetos de forma satisfatória;
- Facilidade, capacidade e flexibilidade no processo de integração de rastreadores no *framework*: tanto o OpenCV quanto o dlib permitem uma integração e prototipagem simples, rápida e fácil, logo, a implementação desses

foram realizadas sem maiores problemas.

Nas seções adiante são descritas as particularidades dos rastreadores e as bibliotecas utilizadas para a aplicação desses no *framework* desenvolvido.

3.4.1 MOSSE

O MOSSE (do inglês, *Minimum Output Sum of Squared Error*) proposto por Bolme et al. (2010) é um filtro baseado na técnica de filtragem correlacional, presente na biblioteca OpenCV e que foi desenvolvido para ser aplicado na tarefa de rastreamento de objetos. Utilizando a técnica de minimizar a soma do erro quadrático em relação da saída, filtros correlacionais adaptativos estáveis são gerados a partir de um único *frame*. O MOSSE é veloz e apresenta robustez a variações de dimensão, pose, iluminação e deformações não rígidas (BOLME et al., 2010).

3.4.2 dlib

O dlib (KING, 2009) é um conjunto de ferramentas *open-source* para criação de softwares complexos e algoritmos de aprendizagem de máquina. Utilizado na academia e em indústrias, o dlib pode ser aplicado desde dispositivos com limitações de recursos (celulares e dispositivos embarcados, por exemplo) até os de alta performance (supercomputadores). Neste trabalho, o dlib é utilizado para efetuar o rastreamento de objetos a partir do algoritmo de filtros correlacionais implementado nessa biblioteca e que foi proposto por Danelljan et al. (2014).

A técnica proposta por Danelljan et al. (2014) foi desenvolvida com base no rastreador MOSSE de Bolme et al. (2010). Por meio dos filtros correlacionais aprendidos e que são utilizados para localizar o objeto em um novo *frame*, estendem-se os filtros para o espaço multidimensional, como uma pirâmide de dimensões. São utilizados filtros unidimensionais para estimar a dimensão, filtros bidimensionais para a translação e filtros tridimensionais para a localização exaustiva das dimensões do objeto no espaço. Utilizando esta técnica multidimensional proposta é possível estimar com acurácia a dimensão de um objeto depois de uma translação ideal encontrada e podendo, até mesmo, essa ser realizada em tempo real.

3.4.3 KCF

Disponível e implementável por meio da biblioteca OpenCV, o KCF (do inglês, *Kernelized Correlation Filters*) proposto por Henriques et al. (2015), é um rastreador de objetos rápido e robusto baseado nas técnicas "*kernel trick*" de Vapnik, Golowich e Smola (1996), matrizes circulantes e filtros correlacionais.

A partir do problema de regressão de Ridge (método dos mínimos quadrados com regularização, para evitar o sobreajuste) aplicam-se matrizes circulantes (matriz quadrada composta por elementos reais, com cada linha formada por um deslocamento cíclico de $i - 1$ posições para a direita) em amostras (imagens) base e ciclicamente transladadas (verticalmente e horizontalmente) e, realizando as devidas manipulações matemáticas no domínio de Fourier, é obtido a mesma solução da técnica de filtros correlacionais. Contudo, a utilização das matrizes circulantes, garantem o armazenamento de uma grande quantia de dados, além de permitir a substituição de operações matemáticas convencionais custosas pelas operações rápidas de Fourier. Por fim, o "*kernel trick*" (uma ligação entre a linearidade e não linearidade de algoritmos desde que haja computações de produtos escalares, permitindo que os dados sejam mapeados e separados em um maior espaço dimensional por meio de funções *kernels*) é aplicado, utilizando a correlação de *kernel* para determinar a localização do objeto (HENRIQUES et al., 2015).

3.4.4 CSRT

O CSRT (do inglês, *Channel and Spatial Reliability Tracker*), também conhecido como CSR-DCF (do inglês, *Discriminative Correlation Filter with Channel and Spatial Reliability*), foi proposto por Lukežič et al. (2018) como outro método de rastreamento de objetos baseado na técnica de filtros correlacionais e também está presente na biblioteca OpenCV. Por meio das técnicas propostas de confiabilidade de canal e espacial implementadas em conjunto dos filtros correlacionais, o CSRT garante uma alta acurácia no processo, apresentando apenas pequenas quedas de desempenho durante a execução.

A confiabilidade espacial é aplicada por meio de um mapa, que permite um suporte e ajuste do filtro correlacional em relação à parte selecionada da região de interesse (alvo), garantindo um aumento desta região e, assim, aprimorando o processo de rastreamento de objetos não retangulares. No caso da confiabilidade de canal, pontuações são utilizadas para ponderar cada um dos canais de resposta

do filtro, dentre os quais são somados e produzem a resposta final com a nova localização do objeto (LUKEŽIČ et al., 2018).

3.4.5 Rastreador de Centroide

O rastreador de centroide é um algoritmo simples de rastreamento de objetos, utilizado em conjunto com outro rastreador ou com um detector. Implementável utilizando a linguagem Python, este rastreador tem a função de calcular o ponto central (centroide) dos objetos e assimilar um número de identificação para cada objeto presente em um *frame*. A centroide é utilizada como um ponto de referência do objeto e suas coordenadas auxiliam na composição de um gráfico de trajetória (informações de movimento na cena).

As coordenadas das caixas delimitadoras de cada um dos objetos oriundas da detecção ou do rastreamento são, geralmente, de formato retangular e, por meio dessas, aplica-se a fórmula do centro de massa do retângulo para realizar o cálculo da centroide. No presente trabalho, as coordenadas das caixas delimitadoras utilizadas são advindas do rastreador.

O processo de assimilação de um número de identificação para cada objeto existente em um *frame* é baseado no cálculo da distância Euclidiana (distância entre dois pontos). Um dado objeto poderá se mover nos próximos *frames*, logo, assume-se que a distância da centroide para um mesmo objeto nos *frames* posteriores será menor que a distância entre outro objeto. Portanto, por meio do cálculo mínimo da distância Euclidiana é possível atribuir um número de identificação a cada objeto localizado em um *frame*. Para o caso de um objeto sair do campo de visão, esse é apagado e, caso retorne a cena, um novo número de identificação é atribuído. A fim de realizar ajustes no algoritmo, é possível regular a mínima distância Euclidiana do objeto e, também, o número de *frames* que deverão ser percorridos para confirmar que o objeto não está mais presente no *frame*.

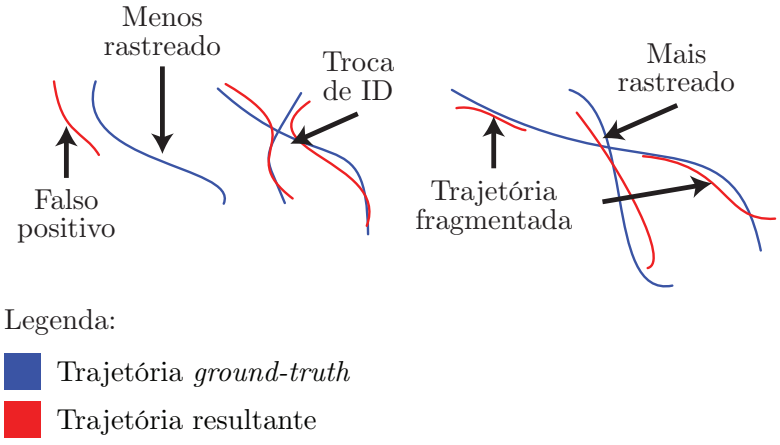
3.4.6 Métricas de rastreamento

Assim como na tarefa de detecção de objetos, o rastreamento também possui métricas estipuladas para avaliar seu desempenho. Dentre as variadas métricas existentes e empregadas na literatura, optou-se por utilizar neste trabalho as métricas CLEAR (do inglês, *CLassification of Events, Activities and Relationships*) (STIEFELHAGEN et al., 2007), um conjunto de medidas de qualidade de rastreamento definido por Wu e Nevatia (2006) e FPS (do inglês, *Frames Per Second*).

As métricas CLEAR foram desenvolvidas objetivando serem métricas universais para o rastreamento de múltiplos objetos (MOT, do inglês, *Multiple Object Tracking*). Pesquisadores comumente utilizam tarefas, *datasets* (incluindo anotações), métricas e processos de avaliação proprietários, impedindo, consequentemente, de realizar comparações (*benchmarks*) e demonstrar as vantagens da pesquisa desenvolvida. As métricas CLEAR foram desenvolvidas em workshops por pesquisadores relacionados ao desenvolvimento de tecnologias para análise de pessoas (atividades, comportamentos e interações), visando criar tarefas e métricas unificadas e bem estabelecidas. As tarefas que as métricas CLEAR podem avaliar são: rastreamento (rostos, pessoas, veículos, bidimensionais, tridimensionais, acústico, visual e áudio-visual), identificação de pessoas (acústico, visual e áudio-visual), estimação da posição da cabeça (vista única e vista múltipla) e análise de cenas acústicas (STIEFELHAGEN et al., 2007; BERNARDIN; STIEFELHAGEN, 2008). Dado a robustez apresentada pelas métricas CLEAR, tornou-se comum sua utilização em pesquisas de aplicações diversas.

Para avaliar a qualidade do rastreamento de objetos, Wu e Nevatia (2006) introduziu cinco critérios que atualmente são amplamente empregados na literatura, dentre os quais estão: número de trajetórias "mais rastreadas" (do inglês, *mostly tracked*), número de trajetórias "menos rastreadas" (do inglês, *mostly loss*), número de trajetórias fragmentadas (do inglês, *trajectory fragment*), número de falsas trajetórias (falso positivo, do inglês, *false alarm*), e a frequência da troca de números de identificação (ID) entre objetos (do inglês, *ID switch*). Na Figura 3.11 são demonstrados com exemplos os critérios desenvolvidos por Wu e Nevatia (2006).

Figura 3.11: Exemplos de trajetórias para avaliação de qualidade do rastreamento de objetos.



Fonte: Adaptado de Wu e Nevatia (2006).

Em MOTChallenge (2020) - um desafio desenvolvido que fornece ferramen-

tas para validar algoritmos de rastreamento de múltiplos objetos e auxiliar nos avanços de diversos campos de pesquisa - são utilizadas as métricas CLEAR e o conjunto de medidas de Wu e Nevatia (2006) e, no caso deste trabalho, optou-se por utilizar a metodologia e ferramentas fornecida deste desafio a fim de avaliar a aplicação de detecção e rastreamento de peixes.

A ferramenta disponibilizada por MOTChallenge (2020) é desenvolvida na linguagem do software MATLAB, no entanto, Heindl (2020) adaptou a ferramenta para a linguagem Python, desenvolvendo a py-motmetrics, sendo esta utilizada neste trabalho. As métricas utilizadas e presentes em MOTChallenge (2020) e Heindl (2020) são citadas e descritas a seguir:

- IDF1: medida de F1 (média harmônica entre precisão e recall) relacionado ao objeto identificado (do inglês, *IDentity F1*), retornando o custo mínimo global de F1. Quanto maior o valor, melhor;
- IDP: medida de precisão relacionado ao objeto identificado (do inglês, *IDentity Precision*), sendo obtido o custo mínimo global de precisão. Quanto maior o valor, melhor;
- IDR: medida de revocação relacionada ao objeto identificado (do inglês, *IDentity Recall*), no qual é obtido o valor de custo mínimo de recall global. Quanto maior o valor, melhor;
- RCLL: revocação (do inglês, ReCaLL) refere-se em um vídeo ao número de detecções e rastreamentos realizados em relação ao número de objetos existentes. Quanto maior o valor, melhor;
- PRCN: precisão (do inglês, *PReCisioN*) representa o número de objetos detectados e rastreados em um vídeo em relação a soma dessas detecções e rastreamentos e os falsos positivos. Quanto maior o valor, melhor;
- GT: do inglês, *Ground Truth*, refere-se ao número de objetos verídicos existentes em um vídeo;
- MT: mais rastreado (do inglês, *Mostly Tracked*) está relacionado ao número de objetos rastreados por pelo menos 80% de todo o seu período de duração ao longo do vídeo. Quanto maior o valor, melhor;
- PT: parcialmente rastreado (do inglês *Partially Tracked*) refere-se ao número de objetos rastreados entre 20% a 80% de todo o seu período de duração no decorrer do vídeo. Quanto maior o valor, melhor;

- ML: menos rastreado (do inglês, *Mostly Loss*) corresponde ao número de objetos rastreados por menos de 20% de todo o seu período de duração ao longo do vídeo. Quanto menor o valor, melhor;

- FP: falso positivo (do inglês, *False Positive*) corresponde ao número total alarmes falsos, ou seja, um rastreamento de objeto que não é genuíno. Quanto menor o valor, melhor;

- FN: falso negativo (do inglês, *False Negative*) refere-se ao número total objetos perdidos (não-rastreados ou perda do rastreamento). Quanto menor o valor, melhor;

- IDS: troca de identidade (do inglês, *IDentity Switch*) refere-se ao número total de trocas de trajetórias e identidades entre objetos rastreados. Quanto menor o valor, melhor;

- FM: trajetória fragmentada (do inglês, *track FragMentations*) representa o número total de trocas relacionadas ao estado do objeto, no qual uma trajetória verdadeira é interrompida e o objeto altera de rastreado para não-rastreado. Quanto menor o valor, melhor;

- MOTA: a acurácia do rastreamento de múltiplos objetos (do inglês, *Multiple Object Tracker Accuracy*) avalia quantos erros distintos como objetos perdidos, trajetórias de alarme falso e trocas de identidade são realizadas sobre todos os *frames* de um vídeo. A soma destes erros resulta em uma porcentagem que varia entre $-\infty$ à 100%. O cálculo do MOTA é realizado utilizando a fórmula:

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDS_t)}{\sum_t GT_t}, \tag{3.7}$$

  sendo FN, FP, IDS, GT e t falsos negativos, falsos positivos, trocas de identidade, verdades terrestres e o índice do *frame* em análise, respectivamente. Quanto maior o valor, melhor;

- MOTP: a precisão do rastreamento de múltiplos objetos (do inglês, *Multiple Object Tracker Precision*) avalia a habilidade do rastreador em estimar com precisão a localização do objeto. Dado em porcentagem, obtêm-se o erro total por meio da localização estimada para um objeto com sua suposta localização obtida em relação à todos os *frames*, sobre a média do número total de correspondências (objeto com sua suposta localização) realizadas. A seguinte fórmula é utilizada para o cálculo do MOTP:

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t}, \tag{3.8}$$

sendo t, i, d e c o índice do *frame* em análise, o objeto de interesse, a sobreposição da caixa delimitadora detectada e rastreada com a caixa delimitadora de *ground truth* e o número de correspondências (objeto com sua suposta localização), respectivamente. Quanto maior o valor, melhor;

- IDT: é contabilizada a quantidade de vezes em que ocorreu a transferência do número de identificação (do inglês, *IDentity Transfer*) de um objeto antigo para um outro objeto. Quanto menor o valor, melhor;
- IDA: é contabilizada a quantidade de vezes em que um número de identificação novo foi atribuído à um objeto (do inglês, *IDentity Ascend*). Quanto menor o valor, melhor;
- IDM: é contabilizada a quantidade de vezes em que um número de identificação novo foi atribuído à um objeto novo (do inglês, *IDentity Migrate*). Quanto menor o valor, melhor.

Para que fosse possível validar o desempenho da aplicação de detecção e rastreamento de peixes referente as métricas citadas anteriormente, é obrigatório um *dataset* composto de vídeos com verdadeiras (*ground truth*) detecções e rastreamentos dos peixes. Para tanto, por meio da ferramenta DarkLabel (DARKLABEL, 2020) - ferramenta para anotação de imagens e vídeos - foram realizadas anotações (desenho das caixas delimitadoras, com número de identificação e nome da classe) *frame* a *frame* em todos os vídeos utilizados neste trabalho e com formato adequado para a utilização na ferramenta do py-motmetrics. Na Tabela 3.8 são apresentadas as principais informações dos vídeos utilizados.

Tabela 3.8: Dados dos vídeos utilizado no processo de obtenção das métricas de rastreamento.

Nome	Frames por segundos	Resolução (em pixels)	Duração (*Frames*/Tempo)	Peixes	Caixas Delimitadoras
Vídeo 1	30	640x480	900 (00:30)	1	900
Vídeo 2	30	640x480	900 (00:30)	2	1.800
Vídeo 3	30	640x480	900 (00:30)	22	7.645
Vídeo 4	30	640x480	900 (00:30)	13	4.646

Fonte: Do autor.

As anotações (*ground truth*) realizadas pela ferramenta DarkLabel e as anotações advindas da execução da aplicação de detecção e rastreamento dos peixes possuem o formato exemplificado e indicado na Tabela 3.9.

Tabela 3.9: Formato de anotações utilizado para a obtenção das métricas de rastreamento MOT.

Número do *frame*	ID do objeto	Coordenada esquerda da caixa delimitadora (em pixels)	Coordenada de topo da caixa delimitadora (em pixels)	Comprimento da caixa delimitadora (em pixels)	Altura da caixa delimitadora (em pixels)	Confiança	Coordenada X para 3D (em pixels)	Coordenada Y para 3D (em pixels)	Coordenada Z para 3D (em pixels)
1	3	558	382	72	56	1	-1	-1	-1
2	4	74	397	48	82	1	-1	-1	-1

Fonte: Do autor.

A métrica de FPS define o desempenho do rastreamento realizado pelo rastreador por meio da contagem de quadros processados por segundo. Cada *frame* é processado em um determinado tempo e, após serem processados todos os *frames* de uma determinada sequência de vídeo, realiza-se uma média aritmética, obtendo, assim, um valor de FPS médio. A partir deste valor de FPS médio é determinado o nível de desempenho de um rastreador. O FPS médio é definido como,

$$FPS_{Medio} = \frac{Frame_1 + Frame_2 + Frame_3 + ... + Frame_n}{Quantidade\ de\ Frames}, \tag{3.9}$$

sendo $Frame_n$ o tempo necessário para processar (rastrear o objeto) o *frame* n e $Quantidade\ de\ Frames$ o número total de *frames* processados.

Outra forma comum de avaliar a tarefa de rastreamento de objetos e que também foi aplicada neste trabalho é por meio da inspeção visual, na qual é possível constatar as possíveis falhas ocorridas no processo de rastreamento como: oclusões; perda de objeto na cena; trocas de objetos por sobreposição; entre outras.

3.5 *Framework* de Detecção e Rastreamento

O *framework* de detecção e rastreamento proposto neste trabalho consiste em um script desenvolvido utilizando a linguagem Python 3.6.9 e aplicando o seguinte conjunto de bibliotecas: NumPy 1.18.0 (OLIPHANT, 2006), imutils 0.5.3 (ROSEBROCK, 2017), Pandas 1.0.4 (TEAM, 2020) (MCKINNEY, 2010), dlib 19.19.0 (KING, 2009), OpenCV 4.2.0 (BRADSKI, 2000) e TensorFlow 1.15.2 (ABADI et al., 2015).

A NumPy engloba uma vasta quantidade de funções e operações para realizar cálculos numéricos, sendo utilizada largamente para manipulações de matrizes e *arrays* multidimensionais, que é o caso da aplicação neste trabalho. A biblioteca

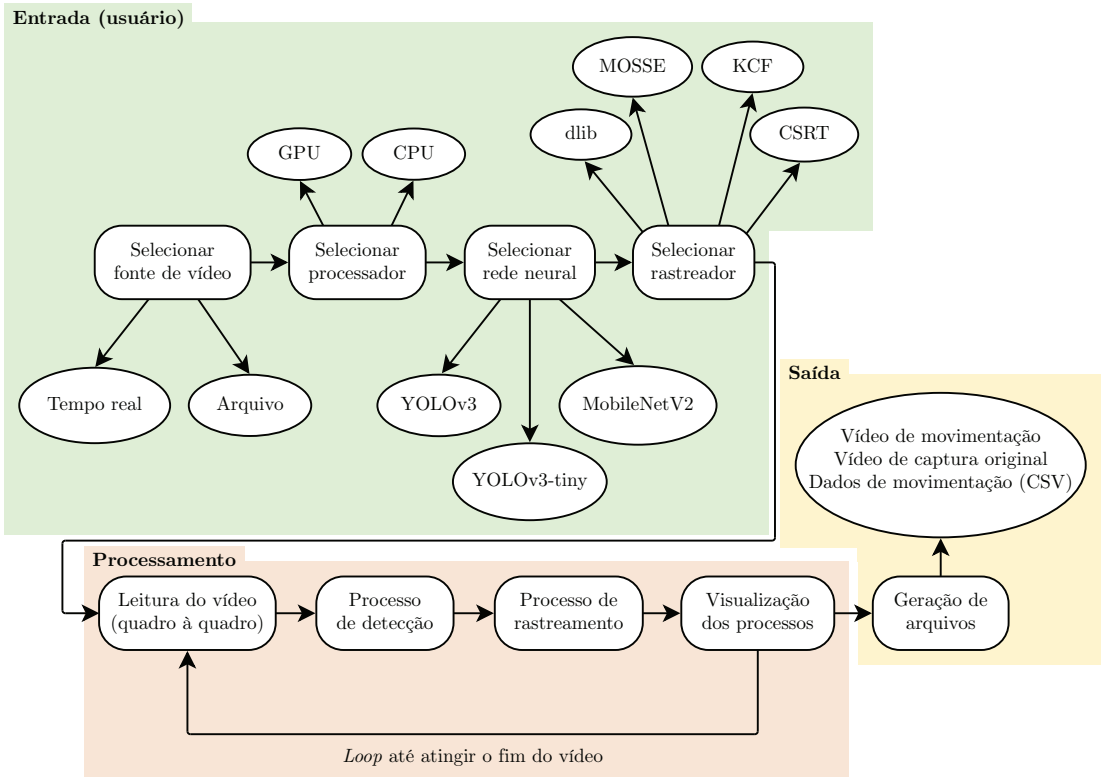
imutils contém ferramentas para efetuar operações básicas de processamento de imagens e vídeos, sendo essa utilizada neste trabalho para o cálculo de FPS (do inglês, *Frames Per Second*). A biblioteca Pandas é aplicada para manipular e analisar dados e, no caso deste trabalho, é utilizada para armazenar os dados de detecção e rastreamento em um arquivo CSV (do inglês, *Comma Separated Values*). A dlib é uma biblioteca ampla, envolvendo diversas ferramentas e, neste trabalho, foi aplicada para realizar a tarefa de rastreamento. A OpenCV, biblioteca *open-source* e *cross-plataform* voltada para o desenvolvimento de aplicações na área de visão computacional, contém uma larga quantidade de ferramentas (algoritmos de aprendizagem de máquina, ferramentas voltadas para o processamento de imagens e vídeos, interface de usuário, operações matemáticas, entre outros), sendo utilizada neste trabalho para realizar a tarefa de detecção de objetos (carregamento e execução das redes YOLOv3 e YOLOv3-tiny), para a tarefa de rastreamento de objetos (carregamento e execução dos algoritmos CSRT, KCF e MOSSE), manipulações de *frames* e uma interface básica de usuário (GUI, do inglês, *Graphical User Interface*). E, por fim, a TensorFlow, utilizada também para a tarefa de detecção de objetos (carregamento e execução da rede MobileNetV2), é uma biblioteca *open-source* para desenvolvimento de modelos de aprendizagem de máquina.

O *framework* desenvolvido via script foi realizada de maneira simplificada e com instruções de uso, buscando praticidade e facilidade no momento da execução do *framework*. Na Figura 3.12 é apresentado o fluxograma com as etapas de execução do *framework*.

Como observado na Figura 3.12 o *framework* desenvolvido foi dividido em três partes, contendo nove etapas no total. A primeira parte, nomeada de entrada (usuário), refere-se à interface entre o usuário e o *framework*, no qual o usuário envia comandos ao *framework*, realizando as configurações prévias necessárias para sua execução. A segunda parte, nomeada de processamento, engloba as etapas referentes ao processamento realizado pelo dispositivo para a detecção e rastreamento de objetos no vídeo, assim como o feedback do processo ao usuário por meio de uma janela de exibição do vídeo. Por fim, na terceira etapa, denominada de saída, é realizada à geração dos arquivos de registro das informações de detecção e rastreamento (em vídeo e dados). A seguir, todas as etapas são descritas em sequência:

- 1) Selecionar fonte de vídeo: o usuário escolhe e indica a fonte de vídeo a ser utilizada, podendo esta ser em tempo real (*webcam*, por exemplo) ou

Figura 3.12: Fluxograma básico do *framework* desenvolvido.



Fonte: Do autor.

de um arquivo previamente gravado. São suportados diversos modelos de câmeras e *codecs* de vídeo;

- 2) Selecionar processador: o usuário escolhe e indica qual processador será utilizado para o processamento (detecção e rastreamento de objetos), optando entre a GPU (placa de vídeo, utilizada única e exclusivamente na etapa de detecção) ou a CPU;

- 3) Selecionar rede neural: o usuário escolhe e indica qual rede neural será utilizada para a etapa de detecção de objetos dentre as três opções fornecidas - YOLOv3 (rede complexa, custo computacional alto), YOLOv3-tiny (rede simples, custo computacional baixo) ou MobileNetV2 (rede simples, custo computacional intermediário);

- 4) Selecionar rastreador: o usuário escolhe e indica dentre as quatro opções fornecidas - dlib (custo computacional baixo), MOSSE (custo computacional muito baixo), KCF (custo computacional intermediário), CSRT (custo computacional alto) - um rastreador que será utilizado na etapa de rastreamento de objetos;

- 5) Leitura do vídeo (quadro a quadro): a fonte de vídeo advinda de um arquivo ou tempo real é carregada e, cada *frame* (quadro), é lido individu-

almente até ser encerrado/atingir seu fim;

- 6) Processo de detecção: a rede neural escolhida é carregada e realiza o processo de detecção no *frame* atual, identificando e localizando o(s) objeto(s). As informações de detecção (caixas delimitadoras, rótulos e pontuações de precisão da detecção) são armazenadas e propagadas para o rastreador;

- 7) Processo de rastreamento: o rastreador é inicializado e recebe as informações advindas da detecção, habilitando-o para efetuar o rastreamento do(s) objeto(s). Atribui-se ao(s) objeto(s) detectado(s) número(s) de identificação único(s) e calcula-se sua(s) centroide(s). Após percorrido um número fixo (configurável) de *frames*, o rastreador é atualizado com novas informações do detector, garantindo que possíveis falhas de rastreamento sejam corrigidas (perdas de objeto, por exemplo);

- 8) Visualização dos processos: uma janela proporcional a resolução do vídeo é exibida ao usuário, permitindo que este visualize os *frames* em tempo real e acompanhe a realização do processo de detecção e rastreamento de objetos pela máquina.

- 9) Geração de arquivos: arquivos de vídeo com e sem as informações de detecção e rastreamento são gerados no formato (*codec*) escolhido. Também é gerado um arquivo CSV com os dados de detecção e rastreamento do(s) objeto(s), contendo o(s) número(s) de identificação do(s) objeto(s) e as coordenadas de localização nos eixos X e Y (em pixels).

Considerando as etapas anteriores e visando auxiliar os pesquisadores em análises comportamentais (usuários do *framework*) de acordo com suas necessidades, algumas funcionalidades foram implementadas no *framework*, dentre elas:

- Detecção e rastreamento automatizado de múltiplos objetos: permite que experimentos envolvendo um único objeto ou múltiplos sejam realizados, possuindo estes um número de identificação único para a diferenciação;

- Escolha de processamento via GPU ou CPU: possibilita a execução do *framework* nos mais variados tipos de dispositivos, sejam eles de baixa ou alta capacidade de processamento;

- Suporte à diversos modelos de câmeras: a flexibilidade fornecida por utilizar diversos tipos e modelos de câmeras permite que os experimentos possam ser executados utilizando câmeras limitadas ou robustas e com diferentes tipos de conexão;

- Captura em tempo real com armazenamento do vídeo com e sem informações de detecção e rastreamento: por meio de uma câmera é realizada a gravação e armazenamento do arquivo de vídeo com e sem informações de detecção e rastreamento (a fim de preservar a captura de vídeo original), sendo realizado a execução e processamento todo em tempo real. Além disso, o *framework* tem a possibilidade de atuar apenas como um simples gravador de experimentos, ficando restrito apenas à gravação e armazenamento do vídeo com o registro das informações de detecção e rastreamento;

- Suporte a vídeos de experimentos gravados previamente com geração do vídeo de informações de detecção e rastreamento: permite que o pesquisador utilize um experimento previamente gravado (com suporte à diversos formatos de vídeo) para obter a análise automatizada e extrair os dados de detecção e rastreamento do objeto, gerando um arquivo de vídeo com estas informações;

- Escolha da resolução e *codec* (formato) para o armazenamento do vídeo: permite que arquivos de vídeo (captura em tempo real e/ou registros das informações de detecção e rastreamento) de maior ou menor qualidade e de uma ampla variedade de formatos sejam gerados e, consequentemente, efetuando o controle de tamanho (espaço de armazenamento necessário);

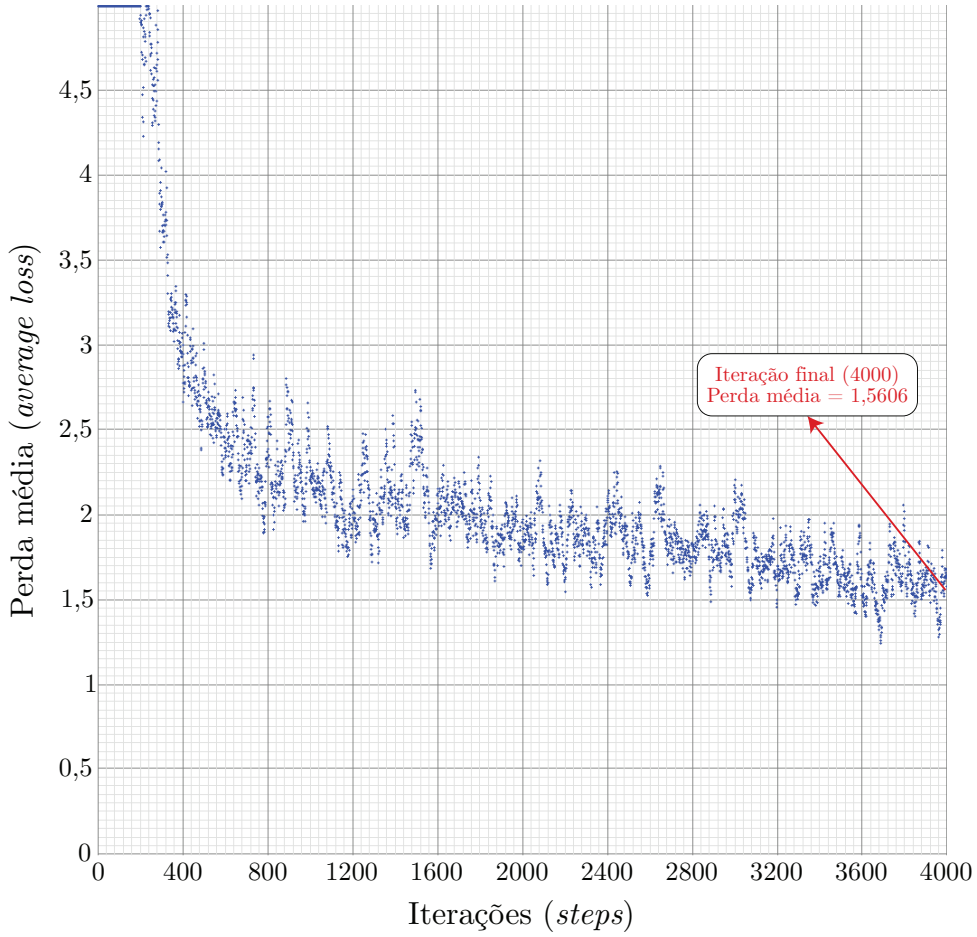
- Arquivo .CSV contendo a quantidade dos peixes detectados e os pontos de rastreamento: produz um arquivo de formato CSV que pode ser manipulável em diversas aplicações a fim de efetuar análises futuras (extração de gráficos de movimentação dos objetos, por exemplo);

A fim de avaliar o funcionamento e o comportamento do *framework* desenvolvido, incluindo os processos de detecção, rastreamento e as funcionalidades implementadas, utilizou-se inicialmente as CNN YOLOv3, YOLOv3-tiny e MobileNetV2 previamente treinadas no *dataset* MS COCO (LIN et al., 2014) e, após constatado o funcionamento, aplicou-se as redes treinadas para a detecção de peixes.

4 Resultados e Discussões

O processo de treinamento envolveu diversas tentativas, buscando extrair o máximo potencial das redes de acordo com o *dataset* aplicado e os hiperparâmetros selecionados. Para o treinamento da rede YOLOv3, a GPU NVIDIA Tesla K80 (contendo 2.496 núcleos CUDA) era o dispositivo disponível para a utilização na plataforma do Google Colab, resultando em um processo de treinamento de aproximadamente 4 horas. Na Figura 4.1 é apresentado o gráfico do processo de treinamento gerado pelo *framework* Darknet, no qual é possível observar a aprendizagem da rede no decorrer das iterações por meio do decaimento da perda média (do inglês, *average loss*).

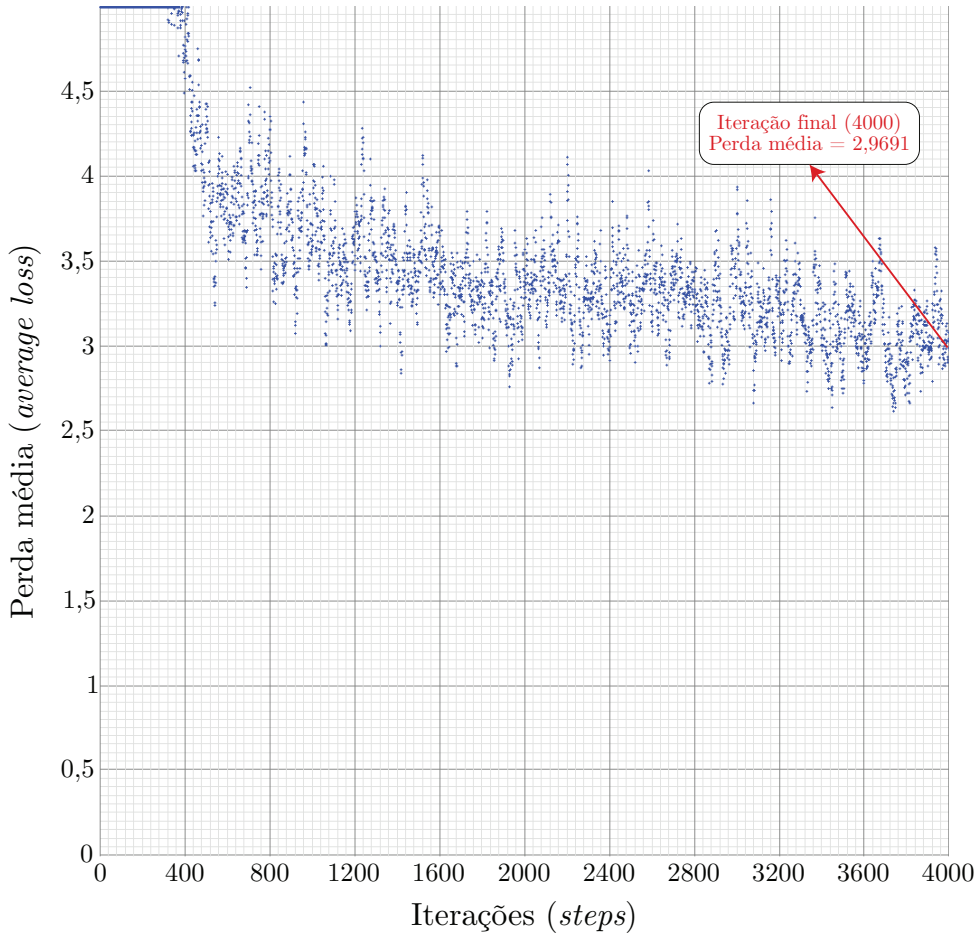
Figura 4.1: Gráfico de treinamento da rede YOLOv3.



Fonte: Do autor.

No processo de treinamento da rede YOLOv3-tiny também foi utilizada a NVIDIA Tesla K80 porém, como esta é uma rede com uma arquitetura mais simplificada e o número de *subdivisions* adotado é menor, o tempo de treinamento foi menor, sendo realizado em aproximadamente 3 horas. A Figura 4.2 apresenta o gráfico de treinamento gerado pelo *framework* Darknet para a rede YOLOv3-tiny.

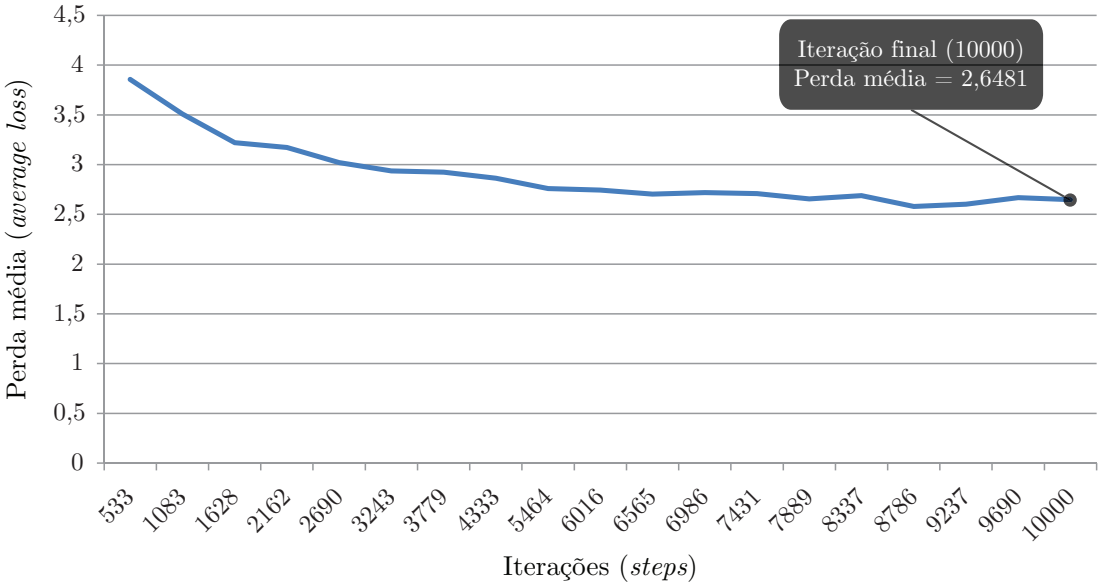
Figura 4.2: Gráfico de treinamento da rede YOLOv3-tiny.



Fonte: Do autor.

Por fim, para o treinamento da MobileNetV2, a GPU NVIDIA Tesla P100 (contendo 3.584 núcleos CUDA) estava disponível e foi utilizada neste processo, que necessitou de aproximadamente 3,5 horas para ser finalizado. A Figura 4.3 apresenta o gráfico gerado pelo TensorBoard (kit de ferramentas de visualização do *framework* TensorFlow) referente ao treinamento da rede MobileNetV2, no qual é visualizado o decaimento da perda média de treinamento no decorrer das iterações.

Figura 4.3: Gráfico de treinamento da rede MobileNetV2.



Fonte: Do autor.

Além disso, foram avaliadas as três CNN baseadas nas métricas de detecção descritas na seção de metodologia. Observando os gráficos de perda média, nota-se que os valores mais baixos de perdas de treinamento sem evidências de sobreajuste, subajuste e a estabilização do processo de treinamento encontram-se próximas da iteração final. Desta forma, foram utilizados os arquivos de pesos da iteração final de cada uma das redes treinadas. A Tabela 4.1 apresenta os valores resultantes de cada uma das métricas para cada CNN treinada.

Tabela 4.1: Métricas e valores resultantes dos processos de treinamento para as redes neurais convolucionais.

Rede neural convolucional	Métricas			
	AP	IoU	*Precision*	Recall
YOLOv3	53,64%	56,42%	0,74	0,45
YOLOv3-tiny	37,73%	46,80%	0,64	0,28
MobileNetV2	52,91%	53,10%	0,84	0,59

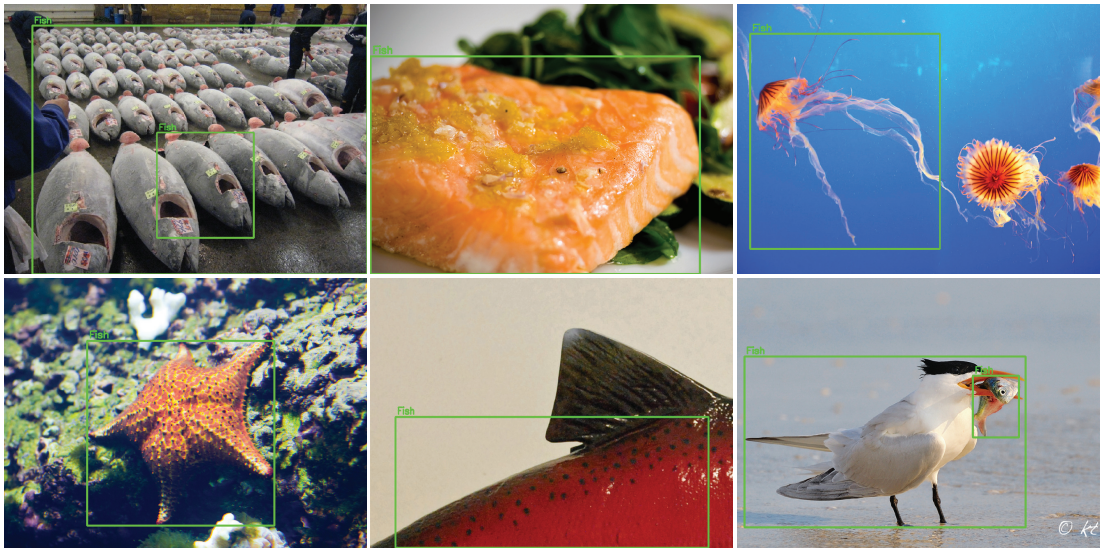
Fonte: Do autor.

Por meio dos resultados apresentados na tabela acima e nos gráficos de perdas, constatou-se que o processo de treinamento obteve resultados abaixo do esperado. Idealmente buscam-se perdas de treinamento abaixo de 1, contudo, notou-se que estas foram relativamente elevadas, impactando diretamente nos valores das métricas e, notoriamente, na capacidade de detecção dos peixes pelas redes. Assim, decidiu-se por retornar algumas etapas do fluxo de trabalho adotado.

Apesar do *dataset* ser verificado humanamente e ser disponibilizado em uma

fonte confiável, uma conferência minuciosa (imagem por imagem) foi realizada, no qual foi constatado que haviam imagens que se enquadravam na classe "peixe (*fish*)" mas não eram relevantes para o *framework* desenvolvido (imagens de peixes como pratos culinários, águas-vivas, cavalos-marinhos, estrelas-do-mar e afins) ou possuíam caixas delimitadoras desenhadas incorretamente (não focada no objeto em questão ou objeto incorreto selecionado). A Figura 4.4 demonstra alguns exemplos de imagens consideradas irrelavantes encontradas no *dataset*.

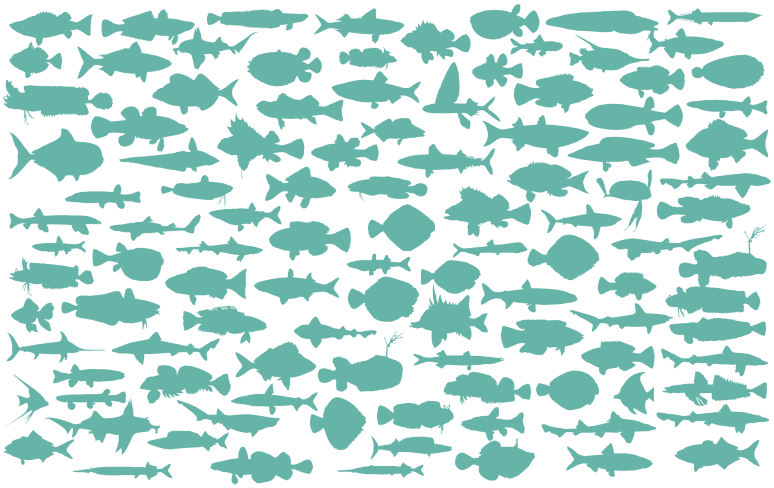
Figura 4.4: Exemplos de imagens consideradas ruins e/ou irrelevantes.



Fonte: Adaptado de Google (2018).

Sendo assim, foi realizado um processo de limpeza (remoção das imagens), buscando manter apenas as imagens que possuem a clássica silhueta de um peixe, mesmo que houvesse variações (diferentes espécies e características próprias). A Figura 4.5 engloba os formatos utilizados como parâmetro para manter as imagens no *dataset*.

Figura 4.5: Exemplos de silhuetas adotadas para a escolha da composição de imagens do *dataset*.

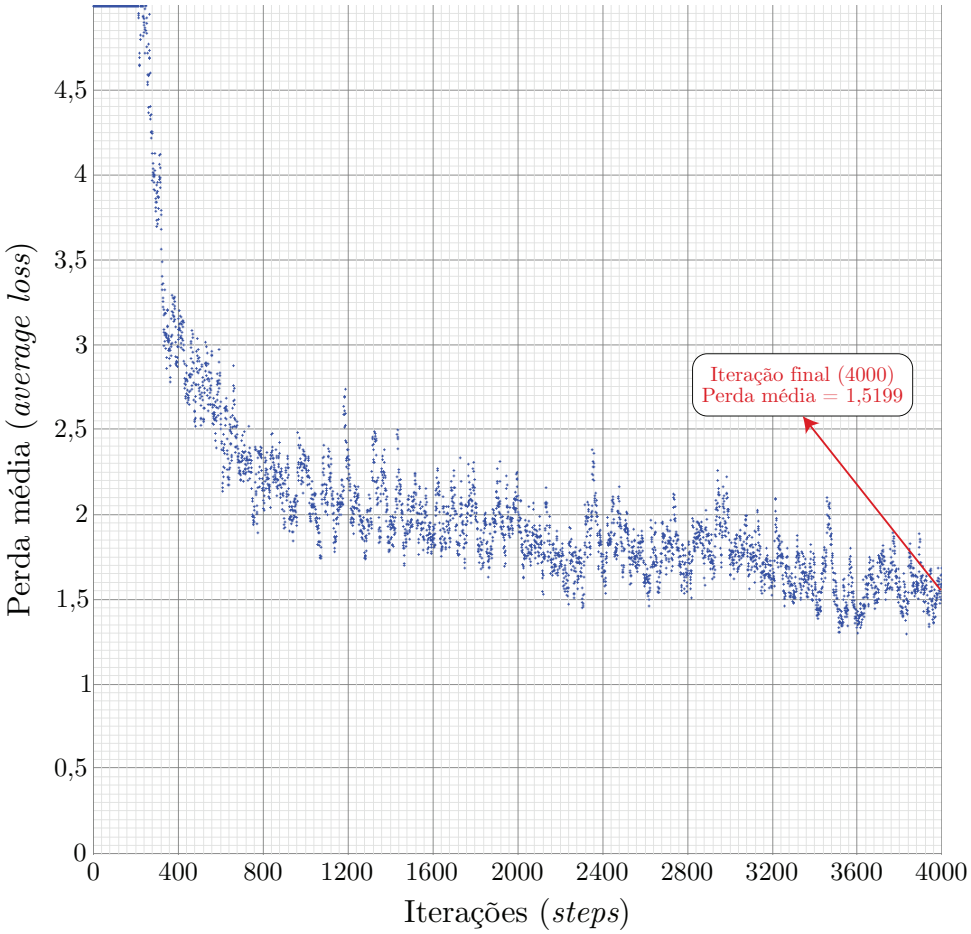


Fonte: Adaptado de Freepik (2020).

Ao todo, 1.372 imagens foram removidas, entre elas 876 de treinamento, 121 de validação e 375 de teste. Sendo assim, portanto, o *dataset* final (após limpeza) é composto por 4.558 imagens de treinamento, 219 de validação e 539 de teste.

Com o *dataset* limpo, novos processos de treinamento foram realizados. A Figura 4.6 apresenta o gráfico de treinamento da rede YOLOv3 para o *dataset* limpo.

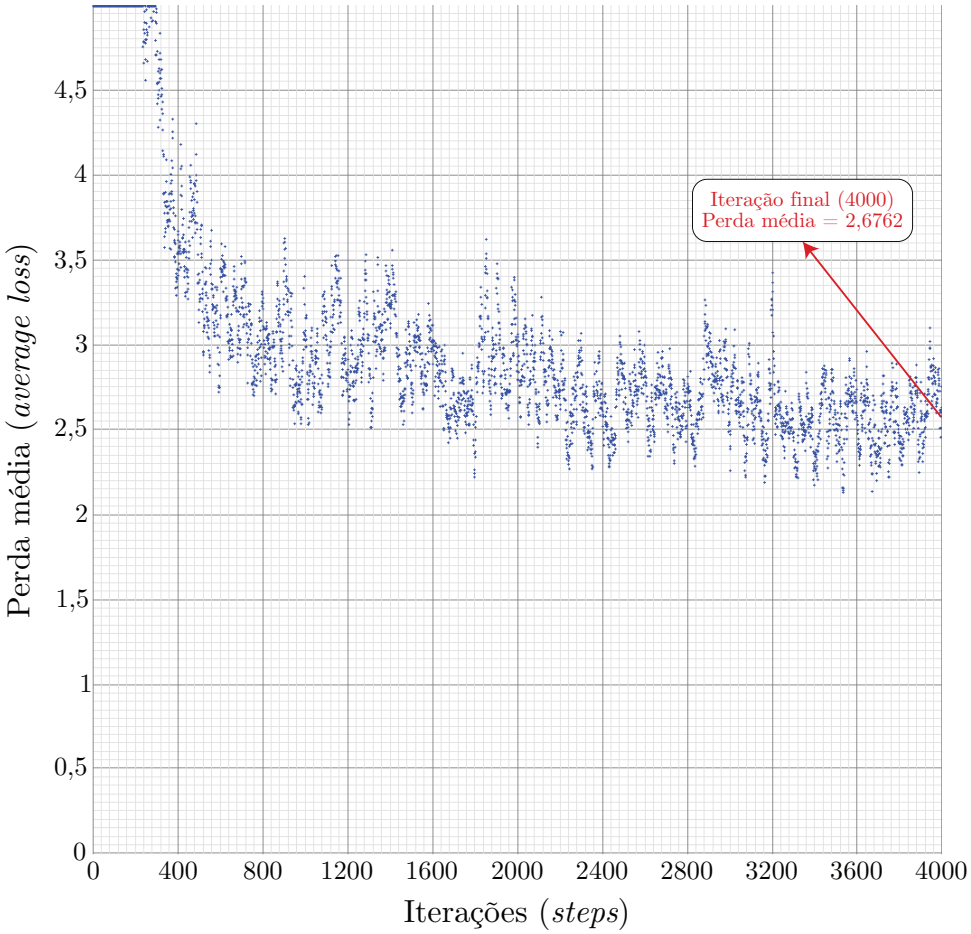
Figura 4.6: Gráfico de treinamento da rede YOLOv3 utilizando o *dataset* limpo.



Fonte: Do autor.

Também foi realizado novamente o processo de treinamento para a rede YOLOv3-tiny aplicando o *dataset* limpo. O gráfico de treinamento pode ser visualizado na Figura 4.7.

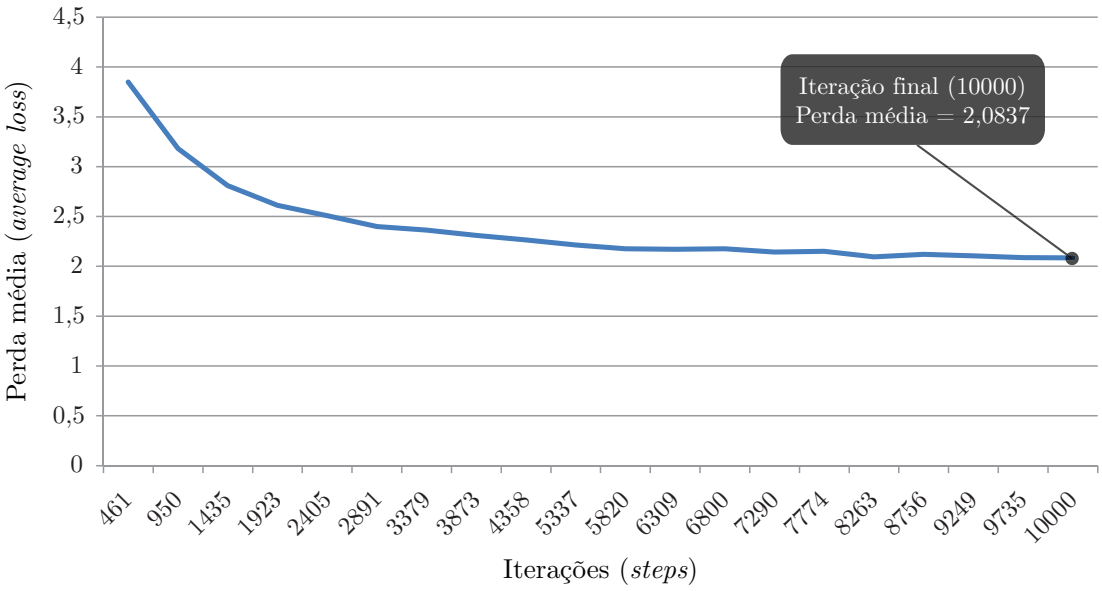
Figura 4.7: Gráfico de treinamento da rede YOLOv3-tiny utilizando o *dataset* limpo.



Fonte: Do autor.

Para o treinamento da rede MobileNetV2 o *dataset* limpo também foi aplicado, resultando no gráfico da Figura 4.8.

Figura 4.8: Gráfico de treinamento da rede MobileNetV2 utilizando o *dataset* limpo.



Fonte: Do autor.

Neste caso também utilizou-se os arquivos de pesos da iteração final de cada uma das redes, sendo que estes apresentaram os melhores resultados durante todo o processo de treinamento. Os valores resultantes do processo de treinamento referente as métricas de cada uma das CNN podem ser visualizados na Tabela 4.2.

Tabela 4.2: Métricas e valores resultantes dos processos de treinamento para as redes neurais convolucionais.

Rede neural convolucional	Métricas			
	AP	IoU	*Precision*	Recall
YOLOv3	57,88%	59,07%	0,78	0,49
YOLOv3-tiny	42,53%	42,81%	0,59	0,39
MobileNetV2	73,32%	62,28%	0,95	0,65

Fonte: Do autor.

Observando os resultados apresentados anteriormente é possível perceber que houve um leve ganho nos resultados das métricas e nas perdas de treinamento de todas as CNN, demonstrando, assim, que a limpeza do *dataset* foi benéfica. Todavia, as perdas de treinamento ainda permaneceram acima de 1, logo acredita-se que este é um *dataset* difícil, devido principalmente à ampla variedade de imagens existentes (diversas espécies de peixes em diferentes cenários), assim como uma possível escassez de imagens de treinamento e a capacidade das CNN em si utilizadas neste trabalho no âmbito da detecção de pequenos objetos - como mencionado por Rosebrock (2017), He et al. (2019) e Redmon (2016).

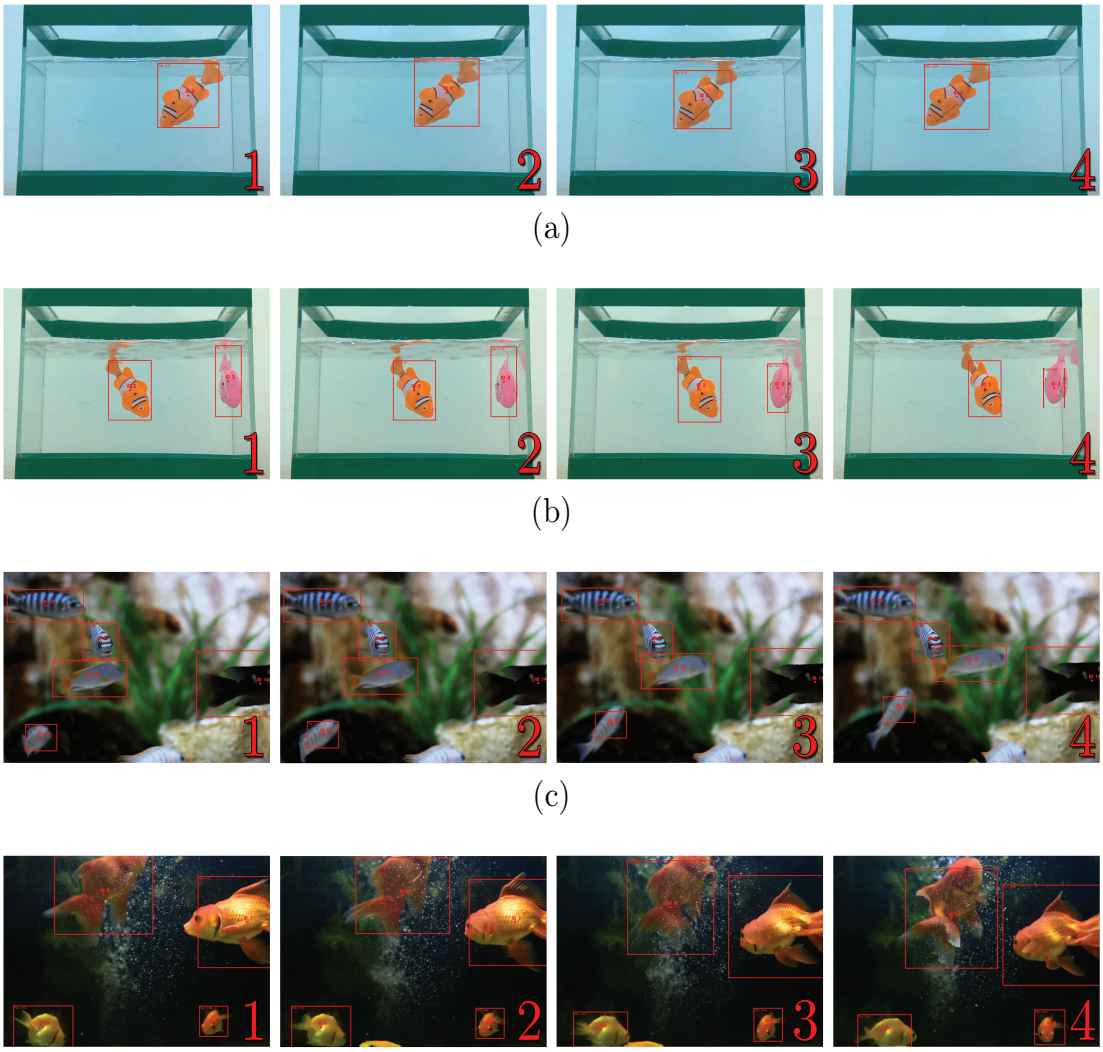
Apesar das perdas de treinamento serem relativamente elevadas, os resultados das métricas apresentados na Tabela 4.2 são considerados razoáveis para a tarefa de detecção de objetos. Logo, por meio dos arquivos de pesos e configurações, foram realizadas as integrações e utilizações das redes treinadas no *framework* desenvolvido, visando inspecionar e estudar a eficácia das CNN.

Objetivando avaliar os detectores, rastreadores, a capacidade de processamento do computador pessoal (CPU) e da placa NVIDIA Jetson Nano (GPU), assim como o desempenho do *framework* desenvolvido para a aplicação em si, quatro vídeos (entitulados de V1, V2, V3, V4) capturados em 30 *frames* por segundo com 30 segundos de duração, variando em três diferentes resoluções (640x480, 800x600 e 1280x720) e codificados no formato MP4 foram utilizados.

Os vídeos V1 e V2 foram gravados utilizando a câmera CSI acoplada na placa NVIDIA Jetson Nano, em conjunto de um aquário de pequeno porte e peixes robóticos, sendo o V1 uma simulação de teste para um único peixe robô e o V2 para dois peixes robôs. Já os vídeos V3 e V4 foram obtidos gratuitamente

no site Pixabay (2020), nos quais são imagens de vários peixes reais em aquários com enriquecimento ambiental. A Figura 4.9 apresenta o *framework* desenvolvido em execução, demonstrando os *frames* em sequência do processo de detecção e rastreamento dos peixes.

Figura 4.9: Execução do *framework* desenvolvido, demonstrando por meio de *frames* em sequência extraídos dos vídeos (a) V1, (b) V2, (c) V3 e (d) V4 os processos de detecção e rastreamento.

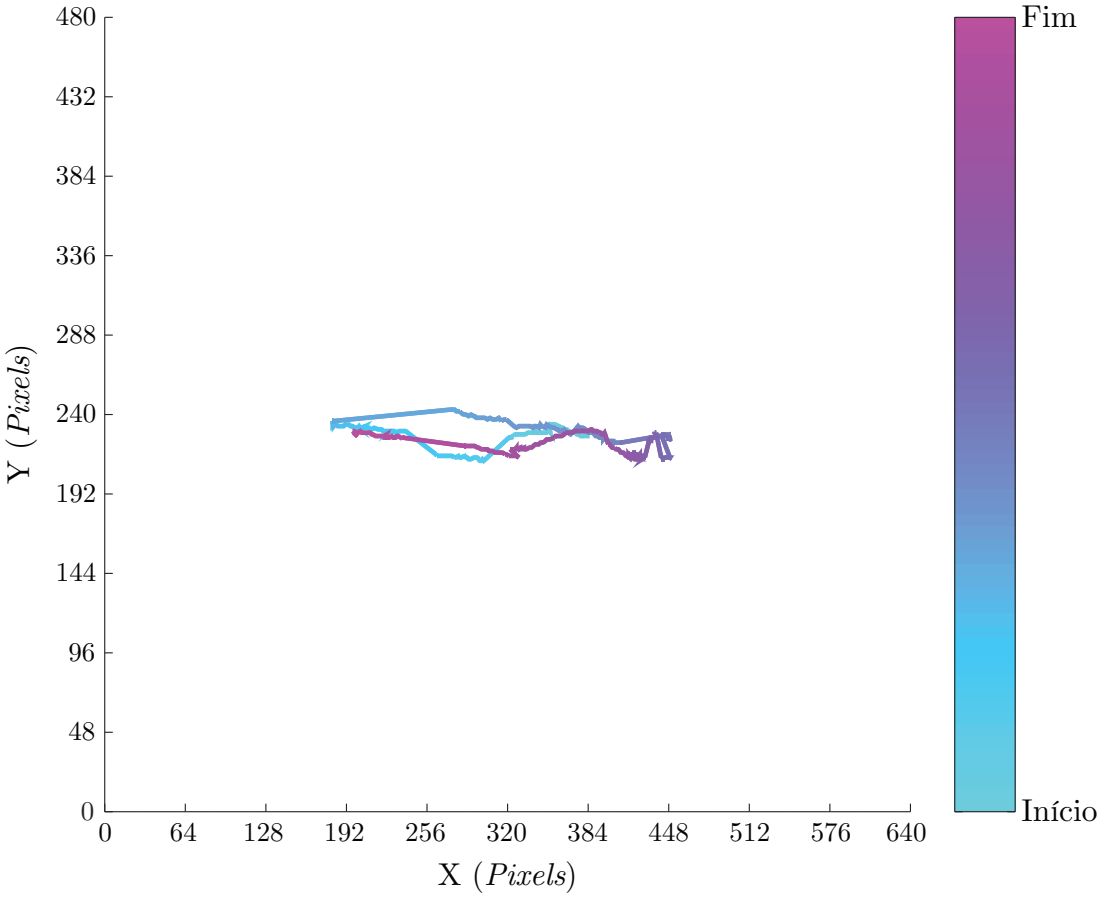


Fonte: Do autor.

Além disso, para cada vídeo, foi gerado um arquivo de saída de formato CSV contendo as respectivas identificações e coordenadas centrais X e Y (em pixels) de cada um dos peixes detectados e rastreados. Por meio destes arquivos é possível realizar as devidas manipulações e obter um gráfico bidimensional de movimentação dos peixes no aquário. A escolha do pós-processamento do arquivo CSV e a utilização de um software externo como o Octave é feita devido a flexibilidade na geração de gráficos, cálculos matemáticos e a necessidade do usuário em manipular adequadamente as informações de acordo com sua necessi-

dade (removendo possíveis dados imprecisos e ruidosos, por exemplo). A Figura 4.10 apresenta um exemplo de gráfico gerado pelo software Octave (EATON et al., 2017) a partir dos dados obtidos do vídeo V1 (resolução 640x480, detector YOLOv3 e rastreador CSRT), no qual é observado o movimento de um peixe no aquário e, aplicando uma escala de cores, é possível acompanhar a trajetória realizada pelo peixe.

Figura 4.10: Gráfico contendo as informações de movimento referentes ao vídeo V1 na resolução de 640x480. Utilizou-se como detector a CNN YOLOv3 e como rastreador o CSRT. Por meio da utilização de uma escala de cores é possível analisar a trajetória do peixe no aquário.



Fonte: Do autor.

O gráfico gerado anteriormente demonstra que houve apenas leves variações na região central do aquário e movimentos horizontais (em grande parte). A área de movimento do peixe é considerada relativamente pequena devido à resolução total do vídeo também incluir os entornos (aquário e parte do ambiente externo). Não foram notados movimentos em regiões superiores e inferiores pois o peixe robótico possui limitações, não sendo possível simular o complexo e natural movimentos dos peixes, sendo utilizado neste trabalho apenas com o enfoque no teste de funcionamento e eficácia do *framework* (assim como das CNN e rastreadores aplicados).

Referente à execução e desempenho do *framework* desenvolvido, foi realizado o monitoramento do consumo de memória pelo *framework* para as redes treinadas e do tempo de processamento (medidos em *frames* por segundo) pelo computador pessoal (CPU) e pela placa NVIDIA Jetson Nano (GPU). A Tabela 4.3 apresenta a quantidade de memória utilizada para cada dispositivo de acordo com a CNN aplicada.

Tabela 4.3: Consumo de memória pelo *framework*.

Dispositivo	Rede neural	Consumo de memória (GB)
Computador pessoal (CPU)	YOLOv3	0,75
	YOLOv3-tiny	0,34
	MobileNetV2	0,53
NVIDIA Jetson Nano (GPU)	YOLOv3	1,77
	YOLOv3-tiny	1,2
	MobileNetV2	1,75

Fonte: Do autor.

Analisando os resultados da tabela anterior, percebe-se que o consumo de memória pela placa foi superior ao computador pessoal. Isto ocorre devido à necessidade de carregamento dos pacotes de suporte para a utilização de uma GPU no processo de detecção. Além disso, há uma relação direta do consumo de memória com o arquivo de pesos das redes, o *framework* utilizado e o vídeo escolhido como entrada. Também foi observado que os rastreadores não causam impactos no consumo de memória, assim como a quantidade de peixes detectados em um vídeo.

O monitoramento do tempo de processamento pelo computador pessoal e pela placa refere-se à execução geral do *framework*, englobando todos os recursos implementados e descritos na seção de metodologia tal como os processos de detecção e rastreamento. A fim de avaliar a capacidade de execução dos dispositivos, foram aplicados os vídeos V1, V2, V3 e V4 em três resoluções distintas, variando os rastreadores e detectores. A Tabela 4.4 apresenta os dados de tempos de processamento referente à execução do *framework* no computador pessoal (CPU) medidos em FPS.

Para a placa NVIDIA Jetson Nano (GPU), é possível observar os tempos de processamento necessários de execução do *framework* na Tabela 4.5, dado em FPS.

Além disso, durante a execução do *framework*, foi realizado o monitoramento do consumo energético do computador pessoal (CPU) e da placa NVIDIA Jetson

Tabela 4.4: Tempos de processamento em FPS referente à execução do *framework* no computador pessoal (CPU).

		Rastreador																
		dlib				CSRT				KCF				MOSSE				
Detector	YOLOv3	21,05	20,33	10,55	9,81	16,66	16,2	4,91	4,07	20,16	19,37	5,54	3,54	25,4	25,41	20,88	20,08	**640x480**
		19,69	18,99	10,09	9,5	14,89	14,56	4,18	3,38	16,24	16,26	3,9	2,47	23,34	23,28	18,08	17,28	**800x600**
		17,65	17,17	9,39	9,12	12,89	12,45	3,16	2,49	12,1	11,83	1,85	1,08	19,82	19,82	13,59	12,73	**1280x720**
	YOLOv3-tiny	22,82	23,13	16,42	11,05	19,57	20,32	9,98	5,12	21,64	21,86	12,06	4,29	25,88	25,96	23,54	20,98	**640x480**
		21,26	21,66	15,45	10,67	17,79	18,71	8,6	4,33	18,55	19,49	8,35	3,18	23,81	23,91	20,93	17,9	**800x600**
		18,75	18,98	13,96	9,81	15,11	15,63	6,54	3,08	14,14	14,44	3,73	1,38	20,31	20,32	16,71	13,85	**1280x720**
	MobileNetV2	22,32	23,54	15,41	12,06	18,76	21,03	10,04	6,97	20,79	21,37	9,98	3,78	26,19	26,33	22,93	20,75	**640x480**
		20,93	21,39	14,72	11,58	17,21	17,87	8,77	5,8	18,41	18,08	6,21	2,48	24,13	24,16	20,21	17,92	**800x600**
		18,52	19,03	13,71	11,23	15,06	15,97	7,46	4,18	12,17	13,99	3,78	1,06	20,56	20,72	15,96	13,38	**1280x720**
		V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
		Vídeos																

Fonte: Do autor.

Tabela 4.5: Tempos de processamento em FPS referente à execução do *framework* na placa NVIDIA Jetson Nano (GPU).

		Rastreador																
		dlib				CSRT				KCF				MOSSE				
Detector	YOLOv3	14,93	14,57	6,46	5,97	11,49	11,53	2,67	2,26	13,5	13,55	3,49	2,47	18,13	18,37	13,67	12,59	**640x480**
		12,79	12,44	6,07	5,53	9,26	9,24	2,28	1,89	10,18	10,34	2,54	1,71	15,26	15,37	10,68	9,72	**800x600**
		9,55	9,47	5,02	4,54	6,79	6,83	1,7	1,34	6,77	6,84	1,06	0,81	10,6	10,7	6,73	6,02	**1280x720**
	YOLOv3-tiny	16,08	16,43	10,9	6,76	13,54	14,17	5,93	2,75	15,15	15,09	7,88	2,97	18,56	18,78	16,02	13,22	**640x480**
		13,98	14,4	9,74	6,24	11,59	12,3	5,13	2,37	12,34	12,72	5,56	2,08	15,7	15,94	13,05	10,44	**800x600**
		10,23	10,5	7,4	5,13	8,33	8,76	3,62	1,77	8,23	8,58	2,87	1,18	11,08	11,27	8,53	6,44	**1280x720**
	MobileNetV2	15,54	16,24	9,87	7,18	12,36	13,55	5,04	2,85	14,34	14,92	6,03	2,74	18,62	18,76	15,09	12,92	**640x480**
		13,57	14,12	8,78	6,52	10,62	11,57	4,14	2,39	11,52	11,9	3,85	1,73	15,65	15,95	11,96	10,12	**800x600**
		9,92	10,35	7,01	5,26	7,55	8,28	3,25	1,69	7,13	7,95	2,55	1	10,85	11,12	7,84	6,09	**1280x720**
		V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
		Vídeos																

Fonte: Do autor.

Nano (GPU). Por meio dos softwares PAPI (do inglês, *Performance Application Programming Interface*) (TERPSTRA et al., 2010) - aplicado no computador pessoal - e tegrastats (NVIDIA, 2020) - aplicado na placa NVIDIA Jetson Nano - foram obtidos dados instantâneos de potência consumida (em Watts). Posteriormente realizou-se uma média aritmética entre estes valores, a fim de comparar o consumo energético de cada dispositivo referente à execução do *framework*, alternando entre detectores, rastreadores, resoluções e vídeos. No apêndice A encontram-se as Tabelas A.1, A.2, A.3 e A.4, que apresentam os dados de consumo energético médio do computador pessoal (CPU) e da placa NVIDIA Jetson Nano (CPU, GPU e total - CPU e GPU combinados) respectivamente.

Outras variáveis também monitoradas em ambos os dispositivos são a utilização da CPU e GPU (no caso da placa NVIDIA Jetson Nano apenas). A utilização da CPU e GPU, dada em porcentagem, refere-se ao tempo despendido pelo processador para processar os recursos exigidos de uma determinada tarefa. Este valor pode ultrapassar os 100%, estando ligado ao número de núcleos (do inglês, *cores*) do processador utilizado, sendo no caso deste trabalho 8 núcleos para o computador pessoal e 4 para a placa NVIDIA Jetson Nano. Por meio do PID (do inglês, *Process IDentifier*) único do processo do *framework* de detecção e rastreamento de peixes, foi realizado o monitoramento da utilização da CPU e GPU para os 4 vídeos aplicados. A coleta dos dados (uso e consumo energético

da CPU e GPU e o consumo de memória) referentes à execução do *framework* foi realizada com uma taxa de amostragem de 30 Hz (0,03334 segundos), conforme a taxa de *frames* máxima dos vídeos utilizados (30 *frames* por segundo). No apêndice A as Tabelas A.5, A.6 e A.7 apresentam os dados de utilização da CPU do computador pessoal e da CPU e GPU da placa NVIDIA Jetson Nano respectivamente.

Analisando os resultados das tabelas citadas anteriormente, é possível admitir que o desempenho geral do *framework* é satisfatório. Nota-se que o computador pessoal (CPU) apresentou resultados de tempo de processamento superior à placa, contudo, *stutterings* (pequenos travamentos) foram percebidos no momento da detecção, algo não visto ao utilizar a GPU da placa. O processo de rastreamento foi realizado com maior rapidez e estabilidade no computador pessoal em comparação a placa, logo, percebe-se que os níveis superiores de FPS do computador pessoal são advindos do processo de rastreamento, devido à sua maior predominância de execução.

Percebe-se que o uso da GPU nos rastreadores considerados mais exigentes computacionalmente (CSRT e KCF) é menor. Isso deve-se ao fato que a GPU se mantêm inativa por um maior período de tempo. Apesar do dlib e do MOSSE apresentarem um uso de GPU maior, a GPU foi utilizada por menos tempo. É válido ressaltar que a GPU não é utilizada durante o processo de rastreamento, somente a CPU. Acredita-se que esses valores mais altos também são decorrentes do tempo de estabilização da GPU (retorno ao 0%), sendo que estes são processados mais rapidamente e, em um curto período de tempo, é realizado novamente o processo de detecção. Mesmo que em alguns momentos o consumo energético da CPU, GPU ou combinado (CPU e GPU) seja levemente inferior em alguns rastreadores, o tempo de processamento (FPS) é maior, consequentemente há um maior consumo de energia.

Referente a detecção, conforme os resultados de treinamento, métricas, consumo de memória, tempo de processamento e inspeções visuais, a rede que apresentou a melhor troca entre tempo de processamento, precisão e acurácia, apesar do seu elevado consumo de memória, foi a MobileNetV2. A YOLOv3 demonstrou precisão e acurácia próximos à MobileNetV2 mas, devido a arquitetura complexa da rede, o seu tempo de processamento é prejudicado. Referente a YOLOv3-tiny, o tempo de processamento apresentado foi similar a MobileNetV2, todavia, possui os piores resultados de precisão e acurácia entre as demais, dado neste caso à simplicidade da rede.

Além disso, no processo de detecção para todas as CNN, falhas foram constatadas. Em diversos momentos os peixes não foram detectados ou foram perdidos nas imagens, principalmente quando tentava-se detectar peixes de pequenas dimensões ou desfocados. Acredita-se que caso sejam otimizadas as perdas no processo de treinamento das CNN, estes problemas possam ser diminuídos ou até mesmo sanados. Uma possível troca ou elaboração de um *dataset* com um maior número de imagens e com imagens mais específicas (referentes à apenas uma espécie de peixe) podem gerar resultados de detecção melhores.

Analisando os resultados das Tabelas A.8, A.9, A.10, A.11 e A.12 presentes no apêndice A nota-se que a aplicação de diferentes detectores, rastreadores e a escolha do intervalo de realização da detecção impacta consideravelmente nos resultados das métricas. Como observado na Tabela 4.2 a MobileNetV2 apresentou melhores resultados decorrentes do processo de treinamento, desta forma percebe-se uma influência nos resultados das métricas de rastreamento, produzindo em sua maioria resultados superiores aos outros detectores utilizados. Sendo assim, acredita-se que a busca por aprimoramentos da capacidade do detector podem elevar a precisão e acurácia do processo.

A existência de altos valores de FP e FN implicaram no desempenho geral do processo, sendo observado seu impacto no resultado de várias métricas (MOTA, MOTP, *Precision* e Recall, por exemplo), assim como os valores de IDS.

Percebe-se que os intervalos menores de detecção (30 e 50 FPS) produziram resultados melhores, retornando valores maiores de precisão e acurácia. Porém, em determinados momentos, notou-se que um intervalo maior (80 FPS) apresentou resultados próximos ou ligeiramente superiores.

Referente aos rastreadores, o dlib apresentou uma melhor troca entre precisão, acurácia e tempo de processamento. Os rastreadores CSRT e KCF apresentaram um desempenho ligeiramente superior ao dlib em relação a precisão e acurácia, no entanto, são algoritmos que necessitam de um maior poder de processamento e, consequentemente, há um consumo energético maior.

O MOSSE, apesar de ser um rastreador veloz, exigir um menor poder de processamento e baixo consumo energético, produziu resultados inferiores de acurácia e precisão comparado aos demais rastreadores. Além disso, analisando os vídeos de rastreamento gerados por meio da inspeção visual, percebe-se que o MOSSE tem dificuldades em ajustar a caixa delimitadora as dimensões do objeto, impactando negativamente a precisão e acurácia. Por outro lado, o MOSSE apresentou uma capacidade superior aos demais rastreadores em rastrear os peixes que rea-

lizaram movimentos rápidos (disparos).

Para todos os rastreadores e vídeos aplicados foram constatados alguns momentos de falha, no qual não foi possível efetuar o rastreamento do peixe (permanecendo com a caixa delimitadora travada em outra posição na imagem) e houve trocas de caixas delimitadoras e ID's entre os peixes devido à oclusões.

Apesar das falhas mencionadas anteriormente, os resultados para as tarefas de detecção e rastreamento são aceitáveis de acordo com *benchmarks* de MOTChallenge (2020), Lin et al. (2014) e Everingham et al. (2010), porém, aprimoramentos para ambos os processos fazem-se necessários.

Conforme análises e resultados demonstrados anteriormente, o rastreador dlib e o detector MobileNetV2 são consideradas as opções mais equilibradas. Em diversos momentos estes obtiveram resultados superiores ou muito próximos dos melhores, portanto, acredita-se serem, de modo geral, as melhores opções de rastreador e detector para a aplicação.

5 Considerações Finais

No presente trabalho foi proposto um *framework* que faz o uso de técnicas de aprendizagem de máquina para efetuar a detecção e rastreamento de peixes, visando auxiliar pesquisadores no processo de análise comportamental para a aferição da qualidade da água e a presença de substâncias tóxicas. O *framework* foi estruturado em três etapas: entrada (interface com o usuário), processamento (detecção e rastreamento) e saída (resultados).

Testes iniciais foram realizados no *framework* para a detecção e rastreamento de objetos genéricos, visando apenas a validação e testes. Após constatado o pleno funcionamento, foi dada sequência nos demais processos para a utilização na detecção e rastreamento de peixes.

Para o sucesso no processo de detecção, realizado por meio de redes neurais convolucionais, foram necessárias diversas tentativas de treinamento até atingir resultados aceitáveis. De acordo com os resultados, o *dataset* OIDv4 referente à classe "peixe (*fish*)" mostrou-se volátil, sendo crucial a limpeza e organização dos dados para a otimização e extração de melhores resultados das CNN.

Apesar de todas as CNN treinadas conseguirem detectar os peixes, a MobileNetV2 se sobressaiu e produziu resultados de desempenho e precisão superiores a YOLOv3 e YOLOv3-tiny. Todavia, falhas se fizeram presentes no processo de detecção, sendo ainda necessário realizar aperfeiçoamentos nas redes treinadas.

Relativo aos rastreadores, o dlib é considerado a melhor opção entre o CSRT, KCF e MOSSE, possuindo bons resultados de desempenho e precisão. Assim como nos detectores, os rastreadores também apresentaram falhas, demonstrando problemas clássicos como oclusão e troca de IDs, devido a limitações intrínsecas a estes.

As funcionalidades desenvolvidas para o *framework*, tal como a integração do detector e rastreador foram realizadas sem maiores problemas, atuando conforme projetado. Portanto, apesar das limitações encontradas, é possível admitir que o *framework* pode ser utilizado para a detecção e rastreamento de peixes, contudo,

ainda é necessário conduzir mais experimentos e aperfeiçoamentos a fim de buscar melhores resultados.

O *framework* e o *dataset* desenvolvidos podem ser encontrados na página localizada no GitHub: `https://github.com/mtg1603/Fish-CNN-Tracking`.

5.1 Trabalhos futuros

Buscando a continuidade do trabalho, são listados a seguir possíveis contribuições que visam otimizar os processos presentes no *framework* e novas abordagens no âmbito do processo de detecção e rastreamento de peixes:

- Criação e aplicação nas CNN utilizadas neste trabalho de um *dataset* robusto e específico (voltado apenas a uma única espécie de peixe), composto por um grande número de imagens e verificado humanamente;
- Treinamento, implementação e testes no *framework* abordando a utilização de outras redes neurais convolucionais, assim como outros tipos de redes neurais (recorrentes, por exemplo);
- Desenvolvimento, treinamento, implementação e testes no *framework* de uma rede neural específica para a detecção de peixes, aplicando operações que buscam extrair o maior número de características dado o formato e a dimensão dos peixes;
- Análise, implementação e testes no *framework* de outros rastreadores;
- Realização de testes (*benchmarks*) do *framework* desenvolvido em outros dispositivos com variadas especificações;
- Aplicação de mais câmeras simultâneas (posicionadas em locais distintos para a captura), mesclando as imagens e obtendo uma maior quantidade de informação, visando aprimorar os resultados;
- Implementação de uma interface gráfica para o *framework*, tornando-o mais amigável, visualmente agradável e buscando facilitar sua utilização;
- Desenvolvimento de uma versão do *framework* voltada para dispositivos móveis;

Referências

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANE, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIEGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Disponível em: <http://www.tensorflow.org/>. Acesso em: 18 de outubro de 2019.

AGÊNCIA NACIONAL DE ÁGUAS. *Conjuntura dos Recursos Hídricos no Brasil*. 2019. Disponível em: <http://www.snirh.gov.br/portal/snirh/centrais-de-conteudos/conjuntura-dos-recursos-hidricos/conjuntura_informe_anual_2019-versao_web-0212-1.pdf>. Acesso em: 7 de maio de 2020.

AGÊNCIA NACIONAL DE ÁGUAS. 2015. Disponível em: <https://www.ana.gov.br/>. Acesso em: 26 de agosto de 2019.

AGGARWAL, C. C. *Neural Networks and Deep Learning*. [S.l.]: Springer, 2018. 497 p. ISBN 978-3-319-94462-3.

ALGETHAMI, N.; REDFERN, S. A robust tracking-by-detection algorithm using adaptive accumulated frame differencing and corner features. *Journal of Imaging*, v. 6, p. 25, 2020. https://doi.org/10.3390/jimaging6040025.

ALVES, M. T. R.; TERESA, F. B.; NABOUT, J. C. A global scientific literature of research on water quality indices: trends, biases and future directions. *Acta Limnologica Brasiliensia*, v. 26, p. 245–253, 2014. https://doi.org/10.1590/s2179-975x2014000300004.

AMERSHI, S.; BEGEL, A.; BIRD, C.; DELINE, R.; GALL, H.; KAMAR, E.; NAGAPPAN, N.; NUSHI, B.; ZIMMERMANN, T. Software engineering for machine learning: A case study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, p. 291–300, 2019. https://doi.org/10.1109/icse-seip.2019.00042.

ANDRADE, R. de O. *As máquinas que tudo veem*. 2019. Disponível em: <https://revistapesquisa.fapesp.br/2019/03/14/as-maquinas-que-tudo-veem/>. Acesso em: 12 de agosto de 2019.

AZEVEDO, C. S.; BARÇANTE, L. Enriquecimento ambiental em zoológicos: em busca do bem-estar animal. *Revista Brasileira de Zoociências*, Universidade Federal de Juiz de Fora, v. 19, n. 2, jun 2018.

BAIN, R.; CRONK, R.; WRIGHT, J.; YANG, H.; SLAYMAKER, T.; BARTRAM, J. Fecal contamination of drinking-water in low- and middle-income countries: A systematic review and meta-analysis. *PLoS Medicine*, Public Library of Science (PLoS), v. 11, n. 5, p. e1001644, may 2014.

BALLARD, D. H.; BROWN, C. M. *Computer vision*. [S.l.]: Prentice Hall, 1982.

BERNARDIN, K.; STIEFELHAGEN, R. Evaluating multiple object tracking performance: The clear mot metrics. *EURASIP Journal on Image and Video Processing*, v. 2008, p. 1–10, 2008. https://doi.org/10.1155/2008/246309.

BERNICO, M. *Deep learning quick reference : useful hacks for training and optimizing deep neural networks with TensorFlow and Keras*. [S.l.]: Packt Publishing, 2018. ISBN 9781788837996,1788837991.

BOCHINSKI, E.; EISELEIN, V.; SIKORA, T. High-speed tracking-by-detection without using image information. *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, p. 1–6, 2017. https://doi.org/10.1109/AVSS.2017.8078516.

BOLME, D.; BEVERIDGE, J. R.; DRAPER, B. A.; LUI, Y. M. Visual object tracking using adaptive correlation filters. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, p. 2544–2550, 2010. https://doi.org/10.1109/CVPR.2010.5539960.

BRADSKI, A. *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]*. 1. ed.. ed. [S.l.]: O'Reilly Media, 2008. Gary Bradski and Adrian Kaehler.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

BROWN, R. M.; MCCLELLAND, N. I.; DEININGER, R. A.; TOZER, R. G. A water quality index: Do we dare? *Water Sewage Works*, v. 117, n. 10, p. 339–343, 1970.

BUDUMA, N.; LOCASCIO, N. *Fundamentals of Deep Learning: Designing Next-generation Machine Intelligence Algorithms*. [S.l.]: O'Reilly Media, 2017.

CARLSON, R. E. A trophic state index for lakes1. *Limnology and Oceanography*, Wiley, v. 22, n. 2, p. 361–369, mar 1977.

CHAHYATI, D.; FANANY, M. I.; ARYMURTHY, A. M. Tracking people by detection using CNN features. *Procedia Computer Science*, v. 124, p. 167–172, 2017. https://doi.org/10.1016/j.procs.2017.12.143.

CHOLLET, F. *Deep Learning with Python*. [S.l.]: Manning, 2017. ISBN: 978-1617294433.

COMPANHIA AMBIENTAL DO ESTADO DE SÃO PAULO. 1998. Disponível em: <https://cetesb.sp.gov.br/>. Acesso em: 22 de agosto de 2019.

CONSELHO NACIONAL DO MEIO AMBIENTE. *Resolução n. 357*. 2005. Disponível em: <http://www2.mma.gov.br/port/conama/legiabre.cfm?codlegi=459>. Acesso em: 23 de agosto de 2019.

COSTA, D. de A.; ASSUMPÇÃO, R. dos S. F. V.; AZEVEDO, J. P. S. de; SANTOS, M. A. dos. Dos instrumentos de gestão de recursos hídricos - o enquadramento - como ferramenta para reabilitação de rios. *Saúde em Debate*, FapUNIFESP (SciELO), v. 43, n. spe3, p. 35–50, dec 2019.

DANELLJAN, M.; HäGER, G.; KHAN, F. S.; FELSBERG, M. Accurate scale estimation for robust visual tracking. *Proceedings of the British Machine Vision Conference*, 2014. http://dx.doi.org/10.5244/C.28.65.

DARKLABEL. *DarkLabel - Video/Image Labeling and Annotation Tool*. 2020. Disponível em: <https://github.com/darkpgmr/DarkLabel>. Acesso em: 16 de setembro de 2020.

DELLAMATRICE, P. M.; MONTEIRO, R. T. R. Principais aspectos da poluição de rios brasileiros por pesticidas. *Revista Brasileira de Engenharia Agrícola e Ambiental*, v. 18, p. 1296–1301, 2014. https://doi.org/10.1590/1807-1929/agriambi.v18n12p1296-1301.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: IEEE. *2009 IEEE conference on computer vision and pattern recognition*. [S.l.], 2009. p. 248–255.

DINIZ, D. G.; SIQUEIRA, L. S. de; HENRIQUE, E. P.; PEREIRA, P. D. C.; DINIZ, C. G.; ABREU, C. C. de; MAGALHÃES, N. G. de M.; SOARES, G. L. da S.; PAIXÃO, P. E. G.; MENESES, J. O.; COUTO, M. V. S. do; SOUSA, N. da C.; CUNHA, F. dos S.; DINIZ, C. W. P.; FUJIMOTO, R. Y. Environmental enrichment increases the number of telencephalic but not tectal cells of angelfish (pterophyllum scalare): an exploratory investigation using optical fractionator. *Environmental Biology of Fishes*, Springer Science and Business Media LLC, v. 103, n. 7, p. 847–857, jun 2020.

EATON, J. W.; BATEMAN, D.; HAUBERG, S.; WEHBRING, R. *GNU Octave version 4.2.1 manual: a high-level interactive language for numerical computations*. 2017. Disponível em: <https://www.gnu.org/software/octave/doc/v4.2.1/>. Acesso em: 25 de junho de 2020.

EMBRAPA. *SACAM*. 2009. Disponível em: <https://ainfo.cnptia.embrapa.br/digital/bitstream/item/81221/1/Proci-09.00411.pdf>. Acesso em: 19 de julho de 2019.

EVERINGHAM, M.; GOOL, L. V.; WILLIAMS, C. K.; WINN, J.; ZISSERMAN, A. The pascal visual object classes (voc) challenge. *International journal of computer vision*, v. 88, p. 303–338, 2010. https://doi.org/10.1007/s11263-009-0275-4.

FEICHTENHOFER, C.; PINZ, A.; ZISSERMAN, A. Detect to track and track to detect. *CoRR*, abs/1710.03958, 2018. https://arxiv.org/abs/1710.03958.

FREEPIK. 2020. Vetores, Fotos de arquivo e downloads PSD grátis. Disponível em: <https://br.freepik.com/>. Acesso em: 19 de junho de 2020.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. MIT Press, 2016. Disponível em: <http://www.deeplearningbook.org>. Acesso em: 1 de setembro de 2019.

GOOGLE, L. *Open Images Dataset V4*. 2018. Disponível em: <https://storage.googleapis.com/openimages/web/index.html>.

GOOGLE, L. *Google Cloud Documentação - Fluxo de trabalho de machine learning*. 2019. Disponível em: <https://cloud.google.com/ai-platform/docs/ml-solutions-overview>. Acesso em: 5 de setembro de 2019.

HAYKIN, S. S. *Neural networks and learning machines*. Third. [S.l.]: Pearson Education, 2009. ISBN: 978-0131293762.

HE; HUANG; WEI; LI; GUO. Tf-yolo: An improved incremental network for real-time object detection. *Applied Sciences*, v. 9, p. 3225, 2019. https://doi.org/10.3390/app9163225.

HEINDL, C. *py-motmetrics*. 2020. Disponível em: <https://github.com/cheind/py-motmetrics>. Acesso em: 15 de setembro de 2020.

HENRIQUES, J. F.; CASEIRO, R.; MARTINS, P.; BATISTA, J. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 37, p. 583–596, 2015. https://doi.org/10.1109/TPAMI.2014.2345390.

HERRERA, C. T.; LABRAM, J. G. A perovskite retinomorphic sensor. *Applied Physics Letters*, AIP Publishing, v. 117, n. 23, p. 233501, dec 2020.

HICKMAN, D.; JOHNSON, J.; VEMULAPALLI, T.; CRISLER, J.; SHEPHERD, R. Commonly used animal models. In: *Principles of Animal Research*. [S.l.]: Elsevier, 2017. p. 117–175.

HILDRETH, E. C.; ULLMAN, S. *The measurement of visual motion.* [S.l.], 1982.

HOLLEMANS, M. *Core ML Survival Guide*. [S.l.]: Leanpub, 2020.

HOLT, A.; HUANG, C.-Y. *Embedded Operating Systems: A Practical Approach*. [S.l.]: Springer, 2014. ISBN: 978-1447166023.

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

INSTITUTO AMBIENTAL DO PARANÁ. 2017. Disponível em: <http://www.iap.pr.gov.br/>. Acesso em: 26 de agosto de 2019.

IPIÑA, K. L. de; CEPEDA, H.; REQUEJO, C.; FERNANDEZ, E.; CALVO, P. M.; LAFUENTE, J. V. Machine learning methods for environmental-enrichment-related variations in behavioral responses of laboratory rats. In: *Understanding the Brain Function and Emotions*. [S.l.]: Springer International Publishing, 2019. p. 420–427.

KING, D. E. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, v. 10, p. 1755–1758, 2009. https://dl.acm.org/doi/10.5555/1577069.1755843.

KLETTE, R. *Concise Computer Vision - An Introduction into Theory and Algorithms*. [S.l.]: Springer, 2014. 1-413 p. (Undergraduate Topics in Computer Science).

KRASIN, I.; DUERIG, T.; ALLDRIN, N.; VEIT, A.; ABU-EL-HAIJA, S.; BELONGIE, S.; CAI, D.; FENG, Z.; FERRARI, V.; GOMES, V.; GUPTA, A.; NARAYANAN, D.; SUN, C.; CHECHIK, G.; MURPHY, K. Openimages: A public dataset for large-scale multi-label and multi-class image classification. 2016. Disponível em: <https://github.com/openimages>. Acesso em: 1 de outubro de 2019.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (Ed.). *Advances in Neural Information Processing Systems 25*. [S.l.]: Curran Associates, Inc., 2012. p. 1097–1105.

KUKLINA, I.; KOUBA, A.; KOZÁK, P. Real-time monitoring of water quality using fish and crayfish as bio-indicators: a review. *Environmental Monitoring and Assessment*, v. 185, p. 5043–5053, 2013. https://doi.org/10.1007/s10661-012-2924-2.

LAMPARELLI, M. C. *Graus de trofia em corpos d'água do estado de São Paulo: avaliação dos métodos de monitoramento*. Tese (Doutorado) — Universidade de São Paulo, 2004.

LEA, P. *Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security*. 1. ed. [S.l.]: Packt Publishing, 2018. ISBN 1788470591, 978-1788470599.

LEAL-TAIXE, L. Multiple object tracking with context awareness. *CoRR*, abs/1411.7935, 2016. https://arxiv.org/pdf/1411.7935.

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

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft coco: Common objects in context. *European conference on computer vision*, v. 8693, p. 740–755, 2014. https://doi.org/10.1007/978-3-319-10602-1_48.

LIU, S.; ZHANG, T.; CAO, X.; XU, C. Structural correlation filter for robust visual tracking. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. https://doi.org/10.1109/cvpr.2016.467.

LIU, W.; ANGUELOV, D.; ERHAN, D.; SZEGEDY, C.; REED, S.; FU, C.-Y.; BERG, A. C. Ssd: Single shot multibox detector. *Computer Vision ECCV 2016*, p. 21–37, 2016. https://doi.org/10.1007/978-3-319-46448-0_2.

LUKEŽIČ, A.; VOJÍŘ, T.; ZAJC, L. Č.; MATAS, J.; KRISTAN, M. Discriminative correlation filter tracker with channel and spatial reliability. *International Journal of Computer Vision*, v. 126, p. 671–688, 2018. https://doi.org/10.1007/s11263-017-1061-3.

LUO, W.; XING, J.; MILAN, A.; ZHANG, X.; LIU, W.; ZHAO, X.; KIM, T.-K. Multiple object tracking: A literature review. *CoRR*, abs/1409.7618, 2014. https://arxiv.org/abs/1710.03958.

MARR, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. [S.l.]: Henry Holt and Co., Inc., 1982. https://dl.acm.org/doi/10.5555/1095712.

MATHWORKS. 2019. Object Recognition - 3 things you need to know. Disponível em: <https://www.mathworks.com/solutions/image-video-processing/object-recognition.html>. Acesso em: 01 de março de 2020.

MCKINNEY Wes. Data Structures for Statistical Computing in Python. In: WALT Stéfan van der; MILLMAN Jarrod (Ed.). *Proceedings of the 9th Python in Science Conference*. [S.l.: s.n.], 2010. p. 56 – 61.

MENG, L.; HIRAYAMA, T.; OYANAGI, S. Underwater-drone with panoramic camera for automatic fish recognition based on deep learning. *IEEE Access*, v. 6, p. 17880–17886, 2018. https://doi.org/10.1109/ACCESS.2018.2820326.

MILAN, A.; LEAL-TAIXE, L.; REID, I.; ROTH, S.; SCHINDLER, K. Mot16: A benchmark for multi-object tracking. *CoRR*, abs/1603.00831, 2016. https://arxiv.org/abs/1603.00831.

MINISTÉRIO DO MEIO AMBIENTE. *Lei n. 9.433: Política Nacional de Recursos Hídricos*. Brasília, Secretaria de Recursos Hídricos, 1997. 72 p. Disponível em: <http://www.planalto.gov.br/ccivil_03/LEIS/L9433.htm>. Acesso em: 23 de agosto de 2019.

MITCHELL, T. M. *Machine Learning*. 1. ed. New York, NY, USA: McGraw-Hill, Inc., 1997. ISBN 0070428077, 9780070428072.

MOTCHALLENGE. *Multiple Object Tracking Benchmark*. 2020. Disponível em: <https://www.motchallenge.net/>. Acesso em: 12 de setembro de 2020.

NOLDUS. *EthoVision XT - Video Tracking Software*. 2019. Disponível em: <https://www.noldus.com/ethovision-xt>. Acesso em: 19 de julho de 2019.

NVIDIA. *Tegrastats Utility Documentation*. 2020. Disponível em: <https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/TegraLinuxDriverPackageDevelopmentGuide/AppendixTegraStats.html>. Acesso em: 7 de novembro de 2020.

NWQMC; USGS; EPA. *Water Quality Portal*. 1997. Disponível em: <https://www.waterqualitydata.us/>. Acesso em: 8 de maio de 2020.

OLIPHANT, T. E. *A guide to NumPy*. [S.l.]: Trelgol Publishing USA, 2006.

OZAKI, H.; ICHISE, H.; KITAURA, E.; YAGINUMA, Y.; YODA, M.; KUNO, K.; WATANABE, I. Immutable heavy metal pollution before and after change in industrial waste treatment procedure. *Scientific Reports*, v. 9, p. 1–12, 2019. https://doi.org/10.1038/s41598-019-40634-2.

PALMER, S. E. *Vision science: Photons to phenomenology*. [S.l.]: MIT press, 1999.

PARKER, J. R. *Algorithms for image processing and computer vision*. [S.l.]: John Wiley & Sons, 2010.

PARRA, L.; SENDRA, S.; GARCÍA, L.; LLORET, J. Design and deployment of low-cost sensors for monitoring the water quality and fish behavior in aquaculture tanks during the feeding process. *Sensors*, v. 18, p. 750, 2018. https://doi.org/10.3390/s18030750.

PATTERSON, J.; GIBSON, A. *Deep Learning: A Practitioner's Approach*. [S.l.]: O'Reilly, 2017. ISBN: 978-1-4919-1425-0.

PERMANENT, C. Environnement et nuisances. *Éditions législatives et administratives. Paris, França*, v. 1, 1973.

PIXABAY. 2020. Over 1.4 million royalty free stock photos and videos. Disponível em: <https://pixabay.com/>. Acesso em: 16 de junho de 2020.

P.U., I.; C.C., C.; F.C., I.; I.F., F.; C.A., O. A review of environmental effects of surface water pollution. *International Journal of Advanced Engineering Research and Science*, AI Publications, v. 4, n. 12, p. 128–137, 2017.

RECLAMATION, B. of. *Water Facts - Worldwide Water Supply*. 2019. Disponível em: <https://www.usbr.gov/mp/arwec/water-facts-ww-water-sup.html>. Acesso em: 8 de maio de 2020.

REDMON, J. *Darknet: Open Source Neural Networks in C*. 2016. Disponível em: <http://pjreddie.com/darknet/>. Acesso em: 5 de julho de 2020.

REDMON, J. Darknet: Neural networks for object detection. *Github repository*, 2020. Disponível em: <https://github.com/AlexeyAB/darknet/>. Acesso em: 18 de julho de 2020.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. https://arxiv.org/abs/1506.02640.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. https://arxiv.org/abs/1804.02767.

RESENDE, R.; SIEBEL, A.; BONAN, C. *Biotecnologia Aplicada à Saúde - Fundamentos e Aplicações*. [S.l.]: Blucher, 2015. 622 p. ISBN: 978-85-212-0896-9.

ROSEBROCK, A. *Deep Learning for Computer Vision with Python*. [S.l.]: PyImageSearch, 2017.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, American Psychological Association (APA), v. 65, n. 6, p. 386–408, 1958.

SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A.; CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Jun 2018.

SARAIVA, S. de O.; POMPEU, P. S. The effect of structural enrichment in hatchery tanks on the morphology of two neotropical fish species. *Neotropical Ichthyology*, FapUNIFESP (SciELO), v. 12, n. 4, p. 891–901, dec 2014.

SCHWARZENBACH, R. P.; EGLI, T.; HOFSTETTER, T. B.; GUNTEN, U. von; WEHRLI, B. Global water pollution and human health. *Annual Review of Environment and Resources*, Annual Reviews, v. 35, n. 1, p. 109–136, nov 2010.

SHKURTI, F.; CHANG, W.-D.; HENDERSON, P.; ISLAM, M. J.; HIGUERA, J. C. G.; LI, J.; MANDERSON, T.; XU, A.; DUDEK, G.; SATTAR, J. Underwater multi-robot convoying using visual tracking by detection. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, p. 4189–4196, 2017. https://doi.org/10.1109/IROS.2017.8206280.

SNIRH. *Sistema Nacional de Informações sobre Recursos Hídricos*. 1997. Disponível em: <http://www.snirh.gov.br/portal/snirh/centrais-de-conteudos/conjuntura-dos-recursos-hidricos/conjuntura_informe_anual_2019-versao_web-0212-1.pdf>. Acesso em: 7 de maio de 2020.

SONKA, M.; HLAVAC, V.; BOYLE, R. *Image processing, analysis, and machine vision*. [S.l.]: Cengage Learning, 2014. https://doi.org/10.1007/978-1-4899-3216-7.

STIEFELHAGEN, R.; BERNARDIN, K.; BOWERS, R.; GAROFOLO, J.; MOSTEFA, D.; SOUNDARARAJAN, P. The clear 2006 evaluation. *Multimodal Technologies for Perception of Humans*, v. 4122, p. 1–44, 2007. https://doi.org/10.1007/978-3-540-69568-4_1.

SUN, D.-W. *Computer vision technology for food quality evaluation*. [S.l.]: Academic Press, 2016.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: *Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015.

TEAM, T. pandas development. *pandas-dev/pandas: Pandas*. [S.l.]: Zenodo, Feb 2020.

TERPSTRA, D.; JAGODE, H.; YOU, H.; DONGARRA, J. Collecting performance data with papi-c. *Tools for High Performance Computing 2009*, p. 157–173, 2010. https://doi.org/10.1007/978-3-642-11261-4_11.

TOLEDO, A.; TALARICO, M.; CHINEZ, S.; AGUDO, D. A aplicação de modelos simplificados para avaliação do processo da eutrofização em lagos e reservatórios tropicais. 1983.

TOLEDO, A. J. Informe preliminar sobre os estudos para a obtenção de um índice para a avaliação do estado trófico de reservatórios de regiões quentes tropicais. *São Paulo: CETESB*, p. 12, 1990.

TOLLERVEY, N. H. *Programming with MicroPython: Embedded programming with microcontrollers and Python*. [S.l.]: O'Reilly Media, Inc., 2017. ISBN: 978-1491972731.

USEPA, U. S. E. P. A. Water quality criteria summary (poster). *Office of Science and Technology, Health and Ecological Criteria Division*, Ecological Risk Assessment Branch (WH-550-D), v. 1, may 1991.

VAPNIK, V.; GOLOWICH, S. E.; SMOLA, A. Support vector method for function approximation, regression estimation and signal processing. *Proceedings of the 9th International Conference on Neural Information Processing Systems*, p. 281–287, 1996. http://dl.acm.org/citation.cfm?id=2998981.2999021.

VITTORIO, A. *Toolkit to download and visualize single or multiple classes from the huge Open Images v4 dataset*. Github, 2018. Disponível em: <https://github.com/EscVM/OIDv4_ToolKit>. Acesso em: 7 de outubro de 2019.

WU, B.; NEVATIA, R. Tracking of multiple, partially occluded humans based on static body part detection. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 06)*, v. 1, p. 951–958, 2006. https://doi.org/10.1109/CVPR.2006.312.

XU, W.; MATZNER, S. Underwater fish detection using deep learning for water power applications. *CoRR*, abs/1811.01494, 2018. http://arxiv.org/abs/1811.01494.

XU, Z.; CHENG, X. E. Zebrafish tracking using convolutional neural networks. *Scientific Reports*, Springer Science and Business Media LLC, v. 7, n. 1, feb 2017.

ZAGATTO, P.; LORENZETTI, M.; LAMPARELLI, M.; SALVADOR, M.; JUNIOR, N.; BERTOLETTI, E. Aperfeiçoamento de um índice de qualidade de Água. *Acta Limnológica Brasiliensia*, v. 11, p. 111–126, 01 1999.

ZAGATTO, P. A.; LORENZETTI, M. L.; PEREZ, L. S. N.; MENEGON, N. M.; BURATINI, S. V. Proposal for a new water quality index. *SIL Proceedings, 1922-2010*, Informa UK Limited, v. 26, n. 5, p. 2449–2451, jun 1998.

ZEBRAFISHFILM.ORG. 2018. Disponível em: <https://www.zebrafishfilm.org/>. Acesso em: 17 de agosto de 2019.

ZEBRATRACK. *ZebraTrack - Overview*. 2017. Disponível em: <https://haesemeyer.github.io/ZebraTrack/>. Acesso em: 18 de julho de 2019.

ZHANG, X.; XIA, G.-S.; LU, Q.; SHEN, W.; ZHANG, L. Visual object tracking by correlation filters and online learning. *ISPRS Journal of Photogrammetry and Remote Sensing*, v. 140, p. 77–89, 2018.

ZUO, W.; WU, X.; LIN, L.; ZHANG, L.; YANG, M.-H. Learning support correlation filters for visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Institute of Electrical and Electronics Engineers (IEEE), v. 41, n. 5, p. 1158–1172, 2019.

Apêndices A

Este apêndice contêm 12 Tabelas com resultados gerais e na íntegra do consumo energético médio (CPU e GPU), a utilização de CPU e GPU e das métricas de rastreamento do MOTChallenge. Nessas Tabelas variam-se os parâmetros de vídeo, detector, rastreador e o período (*frame*) de realização da detecção para avaliar os possíveis impactos nos processos de detecção e rastreamento.

Tabela A.1: Consumo energético médio em Watts pelo computador pessoal (CPU) referente à execução do *framework*.

Detector		Rastreador																Resolução
		dlib				**CSRT**				**KCF**				**MOSSE**				
	YOLOv3	5,1817	5,2113	5,1377	4,9558	5,2626	5,3876	5,3369	5,1749	5,1170	5,1213	4,8142	4,5479	5,1846	5,3573	5,0584	4,8692	**640x480**
		5,1358	5,1822	5,0891	4,9135	5,3866	5,3533	5,3122	5,2040	5,0347	5,0331	4,6563	4,5203	5,2413	5,1373	5,0272	5,0680	**800x600**
		5,1257	5,1836	5,1636	5,0922	5,4261	5,3851	5,3835	5,3234	5,0251	4,9235	4,3156	4,1414	5,0900	5,1646	5,0617	4,9438	**1280x720**
	YOLOv3-tiny	2,8595	2,9018	3,1566	3,4997	3,6143	3,4925	4,7897	5,1796	2,9315	3,0130	3,6728	4,0052	2,7797	2,8123	2,8855	2,7212	**640x480**
		2,9345	3,0094	3,2989	3,6236	3,8314	3,7248	5,0256	5,2891	3,2287	3,1398	3,9797	4,0437	2,9086	2,9135	3,0126	2,9328	**800x600**
		3,1938	3,2577	3,6426	3,9379	4,2601	4,1470	5,3704	5,3844	3,7271	3,6099	4,1156	4,0174	3,1215	3,1499	3,3310	2,7452	**1280x720**
	MobileNetV2	2,4986	2,6549	3,0606	3,2710	3,3089	3,3046	4,8273	5,2080	2,7322	2,8187	3,7757	4,1446	2,5394	2,6068	2,7639	2,6021	**640x480**
		2,7701	2,8611	3,2679	3,5018	3,6175	3,6834	5,1095	5,3143	2,9395	3,1013	4,0000	4,0237	2,6474	2,6908	2,8393	2,8065	**800x600**
		3,0852	3,0376	3,5939	3,7832	4,1791	4,0898	5,3547	5,4168	3,6351	3,6978	3,9980	3,9209	2,9454	2,9288	3,3951	3,4195	**1280x720**
		V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
		Vídeos																

Fonte: Do autor.

Tabela A.2: Consumo energético médio em Watts da CPU da placa NVIDIA Jetson Nano referente à execução do *framework*.

Detector		Rastreador																Resolução
		dlib				**CSRT**				**KCF**				**MOSSE**				
	YOLOv3	0,997	1,013	1,127	1,104	1,287	1,367	1,649	1,616	1,052	1,044	1,202	1,209	0,960	0,958	1,047	1,058	**640x480**
		1,079	1,093	1,172	1,167	1,340	1,384	1,638	1,603	1,135	1,142	1,253	1,248	1,062	1,053	1,131	1,126	**800x600**
		1,156	1,164	1,219	1,209	1,370	1,404	1,607	1,568	1,216	1,209	1,259	1,143	1,153	1,155	1,211	1,214	**1280x720**
	YOLOv3-tiny	1,119	1,066	1,138	1,163	1,375	1,363	1,571	1,637	1,132	1,132	1,212	1,231	1,066	1,069	1,123	1,139	**640x480**
		1,187	1,186	1,224	1,225	1,414	1,398	1,568	1,610	1,205	1,205	1,252	1,254	1,178	1,172	1,215	1,204	**800x600**
		1,259	1,255	1,275	1,259	1,429	1,406	1,557	1,575	1,265	1,265	1,278	1,252	1,261	1,258	1,280	1,273	**1280x720**
	MobileNetV2	1,048	0,994	1,138	1,137	1,243	1,211	1,506	1,575	1,045	1,032	1,170	1,212	0,992	0,994	1,021	1,037	**640x480**
		1,082	1,121	1,141	1,135	1,285	1,274	1,527	1,549	1,114	1,117	1,226	1,244	1,080	1,071	1,100	1,121	**800x600**
		1,158	1,157	1,189	1,193	1,342	1,317	1,517	1,489	1,204	1,196	1,261	1,210	1,148	1,156	1,218	1,211	**1280x720**
		V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
		Vídeos																

Fonte: Do autor.

Tabela A.3: Consumo energético médio em Watts da GPU da placa NVIDIA Jetson Nano referente à execução do *framework*.

Detector		Rastreador																Resolução
		dlib				CSRT				KCF				MOSSE				
	YOLOv3	0,785	0,751	0,430	0,390	0,583	0,540	0,211	0,189	0,707	0,764	0,245	0,200	0,939	0,940	0,773	0,715	**640x480**
		0,771	0,732	0,419	0,383	0,499	0,482	0,191	0,169	0,524	0,549	0,169	0,143	0,840	0,860	0,667	0,609	**800x600**
		0,631	0,640	0,377	0,344	0,389	0,409	0,159	0,137	0,311	0,431	0,093	0,081	0,687	0,704	0,461	0,411	**1280x720**
	YOLOv3-tiny	0,278	0,266	0,202	0,120	0,183	0,188	0,098	0,068	0,246	0,222	0,115	0,069	0,306	0,315	0,284	0,251	**640x480**
		0,255	0,282	0,217	0,118	0,168	0,196	0,093	0,066	0,196	0,230	0,091	0,058	0,299	0,298	0,266	0,226	**800x600**
		0,261	0,256	0,166	0,112	0,156	0,180	0,085	0,062	0,148	0,171	0,066	0,051	0,269	0,280	0,205	0,128	**1280x720**
	MobileNetV2	0,411	0,400	0,320	0,241	0,301	0,332	0,165	0,124	0,371	0,375	0,172	0,114	0,453	0,455	0,389	0,357	**640x480**
		0,388	0,427	0,299	0,214	0,279	0,306	0,146	0,112	0,329	0,344	0,125	0,083	0,423	0,428	0,365	0,337	**800x600**
		0,365	0,378	0,254	0,201	0,251	0,270	0,129	0,095	0,229	0,252	0,091	0,062	0,380	0,391	0,293	0,225	**1280x720**
		V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
		Vídeos																

Fonte: Do autor.

Tabela A.4: Consumo energético médio em Watts combinado (CPU e GPU) da placa NVIDIA Jetson Nano referente à execução do *framework*.

Detector		Rastreador																Resolução
		dlib				CSRT				KCF				MOSSE				
	YOLOv3	1,782	1,764	1,557	1,494	1,870	1,907	1,860	1,805	1,759	1,808	1,447	1,409	1,899	1,898	1,820	1,773	**640x480**
		1,850	1,825	1,591	1,550	1,839	1,866	1,829	1,772	1,659	1,691	1,422	1,391	1,902	1,913	1,798	1,735	**800x600**
		1,787	1,804	1,596	1,553	1,759	1,813	1,766	1,705	1,527	1,640	1,352	1,224	1,840	1,859	1,672	1,625	**1280x720**
	YOLOv3-tiny	1,397	1,332	1,340	1,283	1,558	1,551	1,669	1,705	1,378	1,354	1,327	1,300	1,372	1,384	1,407	1,390	**640x480**
		1,442	1,468	1,441	1,343	1,582	1,594	1,661	1,676	1,401	1,435	1,343	1,312	1,477	1,470	1,481	1,430	**800x600**
		1,520	1,511	1,441	1,371	1,585	1,586	1,642	1,637	1,413	1,436	1,344	1,303	1,530	1,538	1,485	1,401	**1280x720**
	MobileNetV2	1,459	1,394	1,458	1,378	1,544	1,543	1,671	1,699	1,416	1,407	1,342	1,326	1,445	1,449	1,410	1,394	**640x480**
		1,470	1,548	1,440	1,349	1,564	1,580	1,673	1,661	1,443	1,461	1,351	1,327	1,503	1,499	1,465	1,458	**800x600**
		1,523	1,535	1,443	1,394	1,593	1,587	1,646	1,584	1,433	1,448	1,352	1,272	1,528	1,547	1,511	1,436	**1280x720**
		V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
		Vídeos																

Fonte: Do autor.

Tabela A.5: Utilização da CPU do computador pessoal pelo *framework*.

Detector		Rastreador																Resolução
		dlib				CSRT				KCF				MOSSE				
	YOLOv3	312,84%	307,23%	265,35%	264,38%	322,88%	320,73%	310,19%	307,53%	307,45%	301,75%	221,09%	206,78%	327,00%	324,60%	308,11%	305,25%	**640x480**
		307,65%	306,16%	263,96%	260,90%	312,22%	314,85%	307,49%	307,42%	288,91%	291,41%	201,08%	174,51%	319,42%	317,62%	296,72%	294,68%	**800x600**
		301,21%	300,21%	259,61%	257,62%	308,55%	310,02%	304,87%	307,16%	250,02%	262,93%	148,13%	146,11%	310,31%	309,99%	284,26%	283,27%	**1280x720**
	YOLOv3-tiny	97,42%	96,58%	98,45%	100,72%	146,07%	140,65%	202,54%	239,62%	96,26%	97,61%	99,84%	100,72%	95,12%	94,88%	96,69%	99,59%	**640x480**
		102,00%	101,41%	102,25%	103,94%	152,58%	144,50%	215,66%	246,51%	102,91%	101,82%	103,47%	104,46%	101,78%	101,36%	101,63%	105,33%	**800x600**
		110,47%	110,06%	110,66%	110,90%	165,03%	156,24%	232,93%	262,46%	108,79%	110,72%	104,92%	104,92%	111,76%	110,35%	113,07%	99,78%	**1280x720**
	MobileNetV2	71,45%	70,66%	80,49%	84,11%	124,25%	115,87%	206,67%	232,92%	74,32%	73,16%	87,56%	93,08%	68,03%	68,07%	74,23%	78,32%	**640x480**
		79,94%	80,25%	85,55%	88,11%	136,42%	133,76%	220,11%	245,85%	81,85%	81,95%	96,09%	98,69%	76,14%	75,27%	81,91%	84,88%	**800x600**
		91,19%	90,14%	94,73%	96,69%	152,01%	147,75%	240,12%	266,40%	94,98%	94,55%	100,49%	101,40%	89,57%	89,78%	96,88%	102,12%	**1280x720**
		V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
		Vídeos																

Fonte: Do autor.

Tabela A.6: Utilização da CPU da placa NVIDIA Jetson Nano pelo *framework*.

Detector		Rastreador																Resolução
		dlib				CSRT				KCF				MOSSE				
	YOLOv3	86,66%	86,97%	93,08%	94,30%	128,35%	136,98%	168,05%	165,82%	90,84%	90,67%	96,91%	97,99%	87,57%	88,33%	91,64%	93,51%	**640x480**
		93,03%	93,65%	96,70%	97,82%	131,07%	136,29%	166,11%	163,18%	95,59%	96,46%	101,54%	101,44%	92,15%	92,46%	96,56%	99,08%	**800x600**
		99,14%	98,99%	100,67%	101,65%	130,71%	133,33%	159,30%	157,66%	100,39%	100,67%	101,34%	101,69%	99,64%	99,14%	103,72%	108,45%	**1280x720**
	YOLOv3-tiny	89,14%	85,02%	89,98%	94,00%	127,19%	124,50%	154,02%	165,08%	90,35%	89,68%	96,32%	97,94%	87,27%	86,62%	89,24%	93,18%	**640x480**
		93,22%	92,85%	95,97%	98,41%	129,22%	124,46%	152,15%	161,20%	95,33%	94,70%	99,72%	101,08%	92,86%	92,05%	95,92%	98,55%	**800x600**
		99,99%	99,76%	101,92%	102,69%	127,93%	124,39%	148,05%	155,53%	100,90%	100,22%	101,52%	101,95%	99,57%	100,17%	104,81%	108,38%	**1280x720**
	MobileNetV2	83,58%	80,19%	89,05%	90,90%	113,82%	109,45%	147,75%	156,73%	83,13%	82,25%	92,84%	95,88%	80,09%	79,53%	82,12%	85,64%	**640x480**
		86,32%	86,85%	90,38%	93,06%	116,53%	114,30%	148,61%	152,39%	89,01%	88,19%	97,39%	99,50%	85,12%	83,88%	88,96%	92,25%	**800x600**
		92,90%	93,01%	95,50%	97,35%	119,31%	115,82%	144,39%	148,85%	95,88%	95,05%	99,91%	100,86%	92,50%	91,81%	98,94%	102,31%	**1280x720**
		V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
		Vídeos																

Fonte: Do autor.

Tabela A.7: Utilização da GPU da placa NVIDIA Jetson Nano pelo *framework*.

Detector	Rastreador														Resolução
	dlib				CSRT				KCF				MOSSE		
YOLOv3	16,14%	15,01%	9,56%	9,06%	13,00%	12,18%	5,77%	5,13%	14,47%	15,22%	6,03%	5,08%	20,18%	19,28%	15,42%	14,20%	**640x480**
	15,77%	14,70%	9,82%	9,37%	11,57%	11,37%	5,49%	4,83%	11,54%	12,19%	4,43%	3,82%	16,78%	17,35%	12,69%	11,80%	**800x600**
	13,18%	13,47%	9,48%	8,94%	9,84%	9,80%	4,12%	3,50%	7,34%	9,93%	2,12%	1,77%	13,40%	14,29%	10,74%	10,30%	**1280x720**
YOLOv3-tiny	5,01%	4,88%	3,97%	4,19%	5,16%	5,82%	4,36%	3,39%	5,13%	5,05%	3,03%	2,98%	6,57%	6,25%	4,80%	4,25%	**640x480**
	5,33%	5,52%	4,19%	4,57%	5,31%	5,43%	4,35%	3,15%	4,90%	4,93%	3,43%	2,39%	5,54%	6,26%	4,67%	3,72%	**800x600**
	4,82%	5,34%	5,28%	5,32%	5,28%	5,02%	3,78%	2,38%	4,57%	4,72%	2,27%	1,56%	4,74%	5,10%	4,81%	5,68%	**1280x720**
MobileNetV2	13,96%	13,18%	10,68%	9,47%	11,18%	11,93%	7,22%	5,69%	12,60%	12,53%	6,67%	4,66%	16,00%	16,00%	13,22%	11,77%	**640x480**
	12,89%	14,06%	9,79%	9,04%	10,77%	11,20%	6,54%	5,22%	11,29%	11,36%	5,27%	3,17%	14,44%	14,18%	11,53%	10,65%	**800x600**
	11,26%	11,52%	9,79%	8,64%	10,00%	10,17%	5,29%	3,95%	8,75%	9,56%	3,19%	1,93%	11,98%	12,25%	10,34%	9,32%	**1280x720**
	V1	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	**V1**	**V2**	**V3**	**V4**	
	Vídeos																

Fonte: Do autor.

Tabela A.8: Resultados das métricas de rastreamento do MOTChallenge referentes ao vídeo um.

Detector		IDF1	IDP	IDR	RCLL	PRCN	GT	MT	PT	ML	FP	FN	IDS	FM	MOTA	MOTP	IDT	IDA	IDM	FPS
	dlib																			
	YOLOv3	14,80%	22,80%	11,00%	38,00%	78,90%	1	0	1	0	92	559	4	5	27,40%	72,20%	0	4	0	**30**
		22,50%	43,20%	15,20%	31,20%	88,60%	1	0	1	0	36	621	2	2	26,90%	68,00%	0	2	0	**50**
		19,80%	47,50%	12,50%	20,00%	75,60%	1	0	0	1	58	722	1	3	13,40%	73,90%	0	1	0	**80**
		34,80%	82,60%	22,10%	22,10%	82,60%	1	0	1	0	42	703	0	5	17,40%	70,90%	0	0	0	**110**
	YOLOv3-tiny	26,40%	39,80%	19,70%	45,10%	91,10%	1	0	1	0	40	495	2	3	40,50%	65,70%	0	2	0	**30**
		21,30%	39,80%	14,50%	33,40%	91,50%	1	0	1	0	28	601	2	2	30,00%	64,20%	0	2	0	**50**
		28,00%	39,30%	21,70%	48,10%	87,00%	1	0	1	0	65	468	2	4	40,70%	62,60%	0	2	0	**80**
		23,90%	50,40%	15,60%	30,20%	97,10%	1	0	1	0	8	630	1	1	29,20%	63,60%	0	1	0	**110**
	MobileNetV2	29,20%	35,50%	24,70%	61,90%	88,90%	1	0	1	0	70	344	3	7	53,80%	73,60%	0	3	0	**30**
		18,70%	26,00%	14,60%	52,20%	92,90%	1	0	1	0	36	431	4	5	47,80%	69,40%	0	4	0	**50**
		26,90%	30,80%	23,80%	67,30%	87,00%	1	0	1	0	91	295	3	3	56,90%	69,90%	0	3	0	**80**
		33,80%	40,40%	29,00%	55,00%	76,40%	1	0	1	0	153	406	2	11	37,80%	68,30%	0	2	0	**110**
	CSRT																			
		IDF1	**IDP**	**IDR**	**RCLL**	**PRCN**	**GT**	**MT**	**PT**	**ML**	**FP**	**FN**	**IDS**	**FM**	**MOTA**	**MOTP**	**IDT**	**IDA**	**IDM**	
	YOLOv3	14,80%	22,80%	11,00%	37,60%	77,90%	1	0	1	0	96	563	4	6	26,50%	73,60%	0	4	0	**30**
		21,80%	42,00%	14,70%	30,70%	87,40%	1	0	1	0	40	625	2	2	26,10%	72,30%	0	2	0	**50**
		18,20%	43,70%	11,50%	18,80%	71,40%	1	0	0	1	68	732	1	4	11,20%	77,10%	0	1	0	**80**
		40,90%	97,10%	25,90%	25,90%	97,10%	1	0	1	0	7	668	0	2	25,20%	70,30%	0	0	0	**110**
	YOLOv3-tiny	25,80%	38,90%	19,30%	43,80%	88,40%	1	0	1	0	52	507	2	3	37,80%	66,20%	0	2	0	**30**
		20,80%	38,90%	14,20%	33,00%	90,60%	1	0	1	0	31	604	2	2	29,40%	64,80%	0	2	0	**50**
		26,30%	36,90%	20,40%	46,70%	84,40%	1	0	1	0	78	481	2	5	37,80%	64,00%	0	2	0	**80**
		23,50%	49,60%	15,40%	29,90%	96,40%	1	0	1	0	10	632	1	1	28,70%	66,20%	0	1	0	**110**
	MobileNetV2	30,30%	36,90%	25,70%	63,30%	90,90%	1	0	1	0	57	331	3	7	56,70%	74,10%	0	3	0	**30**
		18,30%	25,40%	14,30%	51,40%	91,50%	1	0	1	0	43	438	4	5	46,20%	72,10%	0	4	0	**50**
		27,00%	30,90%	23,90%	66,90%	86,40%	1	0	1	0	95	299	3	3	56,00%	73,00%	0	3	0	**80**
		40,20%	48,10%	34,60%	61,00%	84,70%	1	0	1	0	99	352	2	5	49,80%	70,20%	0	2	0	**110**
	KCF																			
		IDF1	**IDP**	**IDR**	**RCLL**	**PRCN**	**GT**	**MT**	**PT**	**ML**	**FP**	**FN**	**IDS**	**FM**	**MOTA**	**MOTP**	**IDT**	**IDA**	**IDM**	
	YOLOv3	14,80%	22,80%	11,00%	37,30%	77,20%	1	0	1	0	99	566	4	5	25,80%	73,60%	0	4	0	**30**
		21,50%	41,30%	14,50%	30,00%	85,50%	1	0	1	0	46	631	2	3	24,70%	71,60%	0	2	0	**50**
		18,20%	43,70%	11,50%	19,00%	71,80%	1	0	0	1	67	731	1	3	11,40%	75,90%	0	1	0	**80**
		41,30%	97,90%	26,20%	26,20%	97,90%	1	0	1	0	5	666	0	1	25,60%	72,70%	0	0	0	**110**
	YOLOv3-tiny	25,80%	38,90%	19,30%	43,10%	87,00%	1	0	1	0	58	513	2	4	36,50%	66,90%	0	2	0	**30**
		20,80%	38,90%	14,20%	32,70%	89,70%	1	0	1	0	34	607	2	2	28,70%	65,90%	0	2	0	**50**
		26,10%	36,70%	20,30%	46,20%	83,60%	1	0	1	0	82	485	2	2	36,90%	64,20%	0	2	0	**80**
		23,20%	48,90%	15,20%	29,70%	95,70%	1	0	1	0	12	634	1	1	28,30%	66,50%	0	1	0	**110**
	MobileNetV2	28,00%	34,10%	23,70%	62,90%	90,30%	1	0	1	0	61	335	3	6	55,80%	74,70%	0	3	0	**30**
		18,50%	25,60%	14,40%	51,10%	90,90%	1	0	1	0	46	441	4	4	45,60%	70,20%	0	4	0	**50**
		26,50%	30,40%	23,50%	66,00%	85,20%	1	0	1	0	103	307	3	3	54,20%	73,00%	0	3	0	**80**
		37,30%	44,50%	32,00%	57,90%	80,40%	1	0	1	0	127	380	2	7	43,60%	70,50%	0	2	0	**110**
	MOSSE																			
		IDF1	**IDP**	**IDR**	**RCLL**	**PRCN**	**GT**	**MT**	**PT**	**ML**	**FP**	**FN**	**IDS**	**FM**	**MOTA**	**MOTP**	**IDT**	**IDA**	**IDM**	
	YOLOv3	14,80%	22,80%	11,00%	37,30%	77,20%	1	0	1	0	99	566	4	5	25,80%	72,10%	0	4	0	**30**
		22,10%	42,60%	15,00%	30,30%	86,10%	1	0	1	0	44	629	2	2	25,20%	71,20%	0	2	0	**50**
		19,80%	47,50%	12,50%	18,30%	69,30%	1	0	0	1	73	737	1	1	10,10%	73,10%	0	1	0	**80**
		36,00%	85,50%	22,80%	22,80%	85,50%	1	0	1	0	35	696	0	4	19,00%	69,00%	0	0	0	**110**
	YOLOv3-tiny	26,10%	39,40%	19,50%	43,60%	87,90%	1	0	1	0	54	509	2	3	37,40%	66,10%	0	2	0	**30**
		21,10%	39,50%	14,40%	32,40%	88,80%	1	0	1	0	37	610	2	2	28,00%	65,40%	0	2	0	**50**
		27,60%	38,70%	21,40%	46,50%	84,00%	1	0	1	0	80	483	2	7	37,40%	63,70%	0	2	0	**80**
		24,00%	50,70%	15,70%	30,30%	97,50%	1	0	1	0	7	629	1	1	29,40%	65,80%	0	1	0	**110**
	MobileNetV2	28,10%	34,20%	23,80%	61,10%	87,70%	1	0	1	0	77	351	3	6	52,20%	73,40%	0	3	0	**30**
		18,60%	25,80%	14,50%	47,30%	84,20%	1	0	1	0	80	475	4	10	38,00%	68,30%	0	4	0	**50**
		26,60%	30,50%	23,60%	58,50%	75,60%	1	0	1	0	170	374	3	5	39,40%	70,90%	0	3	0	**80**
		29,30%	35,00%	25,20%	50,80%	70,60%	1	0	1	0	191	444	2	13	29,40%	69,30%	0	2	0	**110**

Fonte: Do autor.

Tabela A.9: Resultados das métricas de rastreamento do MOTChallenge referentes ao vídeo dois.

Detector		IDF1	IDP	IDR	RCLL	PRCN	GT	MT	PT	ML	FP	FN	IDS	FM	MOTA	MOTP	IDT	IDA	IDM	FPS
	dlib																			
	YOLOv3	23,50%	55,30%	14,90%	25,90%	96,10%	2	0	1	1	19	1339	2	3	24,70%	72,00%	0	2	0	**30**
		23,80%	89,20%	13,70%	13,70%	89,20%	2	0	0	2	30	1558	0	0	12,10%	68,10%	0	0	0	**50**
		26,30%	50,50%	17,80%	28,20%	80,20%	2	0	1	1	126	1296	2	7	21,20%	71,20%	0	2	0	**80**
		20,20%	59,60%	12,20%	17,50%	85,60%	2	0	1	1	53	1490	0	5	14,60%	70,80%	1	0	1	**110**
	YOLOv3-tiny	17,20%	86,90%	9,50%	9,50%	86,90%	2	0	0	2	26	1634	0	1	8,10%	71,20%	0	0	0	**30**
		24,90%	81,10%	14,70%	14,70%	81,10%	2	0	0	2	62	1540	0	4	11,30%	64,00%	0	0	0	**50**
		10,50%	84,90%	5,60%	6,50%	98,30%	2	0	0	2	2	1689	0	0	6,40%	63,80%	1	0	1	**80**
		11,40%	45,60%	6,50%	9,90%	69,10%	2	0	0	2	80	1627	0	12	5,50%	59,50%	1	0	1	**110**
	MobileNetV2	24,60%	54,20%	15,90%	27,70%	94,30%	2	0	1	1	30	1306	2	10	25,90%	74,90%	0	2	0	**30**
		12,50%	33,20%	7,70%	23,00%	99,00%	2	0	1	1	4	1391	3	4	22,60%	72,30%	0	3	0	**50**
		19,60%	90,00%	11,00%	12,20%	100,00%	2	0	1	1	0	1585	1	1	12,20%	76,30%	0	1	0	**80**
		14,00%	46,60%	8,30%	15,30%	86,60%	2	0	1	1	43	1529	2	3	12,80%	73,10%	0	2	0	**110**
	CSRT																			
	YOLOv3	23,30%	54,90%	14,80%	25,40%	94,40%	2	0	1	1	27	1347	2	4	23,80%	72,00%	0	2	0	**30**
		22,90%	86,00%	13,20%	13,20%	86,00%	2	0	0	2	39	1567	0	3	11,10%	65,60%	0	0	0	**50**
		30,20%	58,00%	20,40%	30,40%	86,30%	2	0	1	1	87	1257	2	9	25,50%	71,60%	0	2	0	**80**
		20,20%	59,60%	12,20%	19,30%	94,60%	2	0	1	1	20	1457	0	3	18,20%	70,20%	1	0	1	**110**
	YOLOv3-tiny	17,70%	89,40%	9,80%	9,80%	89,40%	2	0	0	2	21	1629	0	1	8,60%	73,20%	0	0	0	**30**
		25,10%	81,70%	14,80%	14,80%	81,70%	2	0	0	2	60	1538	0	4	11,50%	66,90%	0	0	0	**50**
		9,80%	79,00%	5,20%	5,20%	79,00%	2	0	0	2	25	1712	0	0	3,80%	69,40%	0	0	0	**80**
		16,20%	64,50%	9,20%	11,40%	79,20%	2	0	0	2	54	1601	0	3	8,40%	62,50%	1	0	1	**110**
	MobileNetV2	25,70%	56,60%	16,60%	28,20%	96,20%	2	0	1	1	20	1296	2	6	27,00%	74,40%	0	2	0	**30**
		12,50%	33,20%	7,70%	23,00%	99,30%	2	0	1	1	3	1390	3	3	22,70%	73,90%	0	3	0	**50**
		19,60%	90,00%	11,00%	12,20%	100,00%	2	0	1	1	0	1585	1	1	12,20%	76,10%	0	1	0	**80**
		14,00%	46,60%	8,30%	15,60%	87,80%	2	0	1	1	39	1525	2	4	13,30%	74,60%	0	2	0	**110**
	KCF																			
	YOLOv3	23,20%	54,70%	14,70%	24,90%	92,40%	2	0	1	1	37	1357	2	8	22,70%	72,10%	0	2	0	**30**
		23,80%	89,20%	13,70%	13,70%	89,20%	2	0	0	2	30	1558	0	0	12,10%	67,70%	0	0	0	**50**
		28,80%	55,30%	19,50%	30,70%	87,10%	2	0	1	1	82	1252	2	5	26,00%	70,90%	0	2	0	**80**
		20,20%	59,60%	12,20%	18,40%	90,00%	2	0	1	1	37	1474	0	13	16,30%	69,00%	1	0	1	**110**
	YOLOv3-tiny	17,40%	87,90%	9,60%	9,60%	87,90%	2	0	0	2	24	1632	0	2	8,30%	73,10%	0	0	0	**30**
		26,10%	85,10%	15,40%	15,40%	85,10%	2	0	0	2	49	1527	0	2	12,70%	66,80%	0	0	0	**50**
		9,70%	78,20%	5,10%	5,10%	78,20%	2	0	0	2	26	1713	0	0	3,70%	68,90%	0	0	0	**80**
		16,20%	64,50%	9,20%	11,00%	76,80%	2	0	0	2	60	1607	0	3	7,70%	66,70%	1	0	1	**110**
	MobileNetV2	23,50%	51,90%	15,20%	26,60%	90,80%	2	0	1	1	49	1325	2	2	23,80%	75,20%	0	2	0	**30**
		12,50%	33,20%	7,70%	22,80%	98,10%	2	0	1	1	8	1395	3	6	22,10%	72,70%	0	3	0	**50**
		19,60%	90,00%	11,00%	12,20%	100,00%	2	0	1	1	0	1585	1	1	12,20%	75,80%	0	1	0	**80**
		14,00%	46,60%	8,30%	15,00%	84,70%	2	0	1	1	49	1535	2	4	12,20%	72,00%	0	2	0	**110**
	MOSSE																			
	YOLOv3	23,30%	54,90%	14,80%	24,70%	91,80%	2	0	1	1	40	1360	2	7	22,40%	68,30%	0	2	0	**30**
		22,10%	82,70%	12,70%	12,70%	82,70%	2	0	0	2	48	1576	0	7	10,10%	67,70%	0	0	0	**50**
		26,00%	49,80%	17,60%	27,00%	76,70%	2	0	1	1	148	1318	2	6	18,70%	69,80%	0	2	0	**80**
		20,20%	59,60%	12,20%	17,90%	87,50%	2	0	1	1	46	1483	0	8	15,30%	65,10%	1	0	1	**110**
	YOLOv3-tiny	15,20%	76,80%	8,40%	8,40%	76,80%	2	0	0	2	46	1654	0	0	5,90%	75,20%	0	0	0	**30**
		22,80%	74,10%	13,50%	13,50%	74,10%	2	0	0	2	85	1563	0	4	8,70%	66,50%	0	0	0	**50**
		10,10%	81,50%	5,40%	5,40%	81,50%	2	0	0	2	22	1709	0	0	4,20%	66,40%	0	0	0	**80**
		14,80%	59,10%	8,50%	10,30%	71,80%	2	0	0	2	73	1620	0	4	6,30%	58,10%	1	0	1	**110**
	MobileNetV2	22,80%	50,20%	14,70%	26,40%	90,00%	2	0	1	1	53	1329	2	3	23,40%	73,10%	0	2	0	**30**
		12,50%	33,20%	7,70%	18,30%	79,00%	2	0	1	1	88	1475	3	5	13,30%	68,60%	0	3	0	**50**
		19,00%	87,30%	10,70%	11,90%	97,30%	2	0	1	1	6	1591	1	1	11,50%	72,30%	0	1	0	**80**
		15,40%	51,20%	9,10%	12,00%	67,80%	2	0	1	1	103	1589	2	2	6,20%	72,80%	1	2	1	**110**

Fonte: Do autor.

Tabela A.10: Resultados das métricas de rastreamento do MOTChallenge referentes ao vídeo três.

Detector		IDF1	IDP	IDR	RCLL	PRCN	GT	MT	PT	ML	FP	FN	IDS	FM	MOTA	MOTP	IDT	IDA	IDM	FPS
	dlib																			
	YOLOv3	37,50%	60,80%	27,10%	29,70%	66,80%	22	0	6	16	1130	5371	8	37	14,90%	71,60%	2	7	1	**30**
		32,30%	54,20%	23,00%	27,70%	65,30%	22	0	5	17	1125	5529	6	24	12,90%	72,10%	1	5	0	**50**
		29,90%	53,50%	20,80%	23,70%	60,90%	22	0	5	17	1160	5836	2	13	8,50%	70,80%	1	2	1	**80**
		24,90%	47,30%	16,90%	19,20%	53,70%	22	0	3	19	1269	6174	4	25	2,60%	70,60%	3	4	3	**110**
	YOLOv3-tiny	14,30%	59,40%	8,10%	8,50%	62,00%	22	0	1	21	398	6995	3	6	3,30%	71,20%	0	3	0	**30**
		12,40%	56,00%	7,00%	8,00%	63,90%	22	0	2	20	344	7037	2	4	3,40%	71,00%	0	2	0	**50**
		12,60%	62,90%	7,00%	7,00%	62,90%	22	0	1	21	316	7109	0	4	2,90%	69,80%	0	0	0	**80**
		10,80%	54,30%	6,00%	6,00%	54,30%	22	0	1	21	385	7187	0	1	1,00%	70,20%	0	0	0	**110**
	MobileNetV2	28,80%	61,80%	18,70%	19,50%	64,40%	22	0	5	17	825	6154	6	13	8,60%	74,30%	0	6	0	**30**
		23,10%	59,30%	14,30%	15,80%	65,30%	22	0	4	18	641	6440	2	7	7,40%	74,70%	0	2	0	**50**
		22,30%	56,40%	13,90%	14,70%	59,50%	22	0	5	17	764	6522	2	13	4,70%	72,00%	0	2	0	**80**
		20,70%	47,70%	13,20%	14,10%	51,00%	22	0	3	19	1035	6566	3	10	0,50%	73,00%	3	3	3	**110**
	CSRT																			
	YOLOv3	38,30%	61,70%	27,80%	29,50%	65,60%	22	0	6	16	1186	5386	7	38	13,90%	71,50%	2	6	1	**30**
		30,90%	51,90%	22,00%	27,20%	64,10%	22	0	4	18	1163	5568	7	22	11,90%	72,00%	1	6	0	**50**
		31,90%	57,00%	22,10%	24,10%	62,10%	22	1	3	18	1124	5800	2	19	9,40%	71,50%	1	1	0	**80**
		26,20%	49,70%	17,80%	20,30%	56,60%	22	1	2	19	1190	6095	4	17	4,70%	69,70%	1	4	1	**110**
	YOLOv3-tiny	14,40%	59,80%	8,20%	8,60%	62,60%	22	0	1	21	392	6989	3	7	3,40%	70,70%	0	3	0	**30**
		12,10%	54,50%	6,80%	8,00%	64,50%	22	0	2	20	338	7031	2	6	3,60%	68,40%	0	2	0	**50**
		12,60%	63,00%	7,00%	7,00%	63,00%	22	0	1	21	315	7108	0	3	2,90%	69,60%	0	0	0	**80**
		12,20%	61,20%	6,70%	6,80%	61,80%	22	0	1	21	322	7124	1	1	2,60%	73,80%	0	1	0	**110**
	MobileNetV2	28,70%	62,50%	18,60%	19,40%	65,00%	22	0	5	17	796	6164	6	15	8,90%	74,90%	1	6	1	**30**
		22,40%	57,50%	13,90%	15,40%	63,60%	22	0	4	18	672	6471	2	8	6,50%	75,50%	0	2	0	**50**
		21,70%	54,80%	13,50%	14,00%	56,80%	22	0	5	17	815	6573	2	14	3,30%	73,40%	0	2	0	**80**
		22,40%	51,70%	14,30%	15,20%	55,00%	22	0	3	19	952	6483	2	9	2,70%	72,40%	2	2	2	**110**
	KCF																			
	YOLOv3	39,50%	64,50%	28,50%	30,30%	68,70%	22	1	5	16	1055	5325	6	34	16,50%	72,60%	2	5	1	**30**
		27,80%	45,80%	19,90%	27,30%	62,80%	22	0	4	18	1234	5558	7	19	11,10%	72,50%	1	6	0	**50**
		34,30%	61,30%	23,80%	24,00%	61,80%	22	0	4	18	1135	5811	1	12	9,10%	71,00%	1	1	1	**80**
		25,80%	49,40%	17,40%	20,10%	56,90%	22	0	3	19	1164	6109	3	27	4,80%	70,10%	3	3	3	**110**
	YOLOv3-tiny	13,30%	55,30%	7,60%	8,00%	58,20%	22	0	1	21	438	7035	3	4	2,20%	72,30%	1	3	1	**30**
		13,00%	58,70%	7,30%	9,10%	73,30%	22	0	2	20	254	6946	3	7	5,80%	69,10%	1	3	1	**50**
		10,50%	52,20%	5,80%	6,10%	54,60%	22	0	1	21	387	7180	1	2	1,00%	70,30%	1	1	1	**80**
		10,70%	53,90%	5,90%	6,00%	54,30%	22	0	1	21	385	7187	1	6	0,90%	69,40%	0	1	0	**110**
	MobileNetV2	27,90%	61,40%	18,00%	19,20%	65,40%	22	0	4	18	777	6175	6	14	9,00%	76,10%	2	6	2	**30**
		21,90%	56,40%	13,60%	15,10%	62,50%	22	0	4	18	692	6491	2	6	6,00%	76,80%	0	2	0	**50**
		22,90%	57,80%	14,30%	14,70%	59,40%	22	0	4	18	766	6523	2	7	4,60%	73,40%	1	2	1	**80**
		22,30%	52,20%	14,20%	14,60%	53,90%	22	0	3	19	957	6528	1	14	2,10%	72,10%	2	1	2	**110**
	MOSSE																			
	YOLOv3	39,60%	65,30%	28,40%	30,50%	70,20%	22	1	5	16	990	5310	6	29	17,50%	72,40%	0	6	0	**30**
		29,90%	49,70%	21,30%	27,40%	63,90%	22	0	4	18	1186	5550	7	25	11,80%	71,00%	1	6	0	**50**
		34,10%	60,90%	23,70%	24,10%	62,00%	22	0	3	19	1128	5803	4	14	9,30%	69,80%	2	2	1	**80**
		26,40%	50,50%	17,80%	19,30%	54,60%	22	0	4	18	1225	6170	2	20	3,20%	70,40%	1	2	1	**110**
	YOLOv3-tiny	14,20%	59,00%	8,10%	9,10%	66,20%	22	0	2	20	355	6951	2	6	4,40%	69,40%	0	2	0	**30**
		12,70%	57,20%	7,10%	8,60%	69,20%	22	0	2	20	294	6986	3	7	4,70%	66,50%	1	3	1	**50**
		11,90%	59,50%	6,60%	6,60%	59,50%	22	0	1	21	345	7138	0	4	2,10%	67,20%	0	0	0	**80**
		10,20%	51,20%	5,70%	5,70%	51,70%	22	0	1	21	407	7209	1	4	0,40%	69,00%	0	1	0	**110**
	MobileNetV2	30,20%	64,90%	19,60%	21,00%	69,30%	22	0	7	15	710	6039	6	9	11,60%	73,90%	0	6	0	**30**
		23,30%	60,00%	14,50%	16,80%	69,40%	22	0	4	18	565	6364	4	5	9,30%	73,40%	1	4	1	**50**
		24,40%	61,70%	15,20%	15,30%	62,00%	22	0	4	18	717	6474	1	6	5,90%	72,30%	0	1	0	**80**
		23,00%	54,00%	14,70%	15,30%	56,60%	22	0	4	18	901	6472	1	11	3,50%	70,90%	2	1	2	**110**

Fonte: Do autor.

Tabela A.11: Resultados das métricas de rastreamento do MOTChallenge referentes ao vídeo quatro.

Detector		IDF1	IDP	IDR	RCLL	PRCN	GT	MT	PT	ML	FP	FN	IDS	FM	MOTA	MOTP	IDT	IDA	IDM	FPS
	dlib																			
	YOLOv3	41,20%	47,50%	36,40%	58,30%	76,10%	13	1	7	5	849	1937	12	67	39,80%	71,70%	9	7	4	**30**
		50,20%	60,70%	42,80%	51,30%	72,70%	13	1	8	4	893	2264	7	50	31,90%	70,10%	1	7	1	**50**
		38,00%	47,40%	31,70%	43,20%	64,40%	13	0	7	6	1108	2639	8	49	19,20%	67,30%	4	6	2	**80**
		30,20%	39,10%	24,60%	35,30%	55,90%	13	0	7	6	1294	3008	10	50	7,20%	68,30%	6	6	2	**110**
	YOLOv3-tiny	28,10%	44,10%	20,60%	34,20%	73,30%	13	0	8	5	579	3057	11	33	21,50%	68,60%	5	9	3	**30**
		27,60%	47,20%	19,50%	27,70%	67,20%	13	0	6	7	629	3359	8	35	14,00%	66,70%	4	6	2	**50**
		21,20%	33,40%	15,50%	24,40%	52,60%	13	0	4	9	1023	3513	9	50	2,20%	63,90%	2	9	2	**80**
		24,20%	45,60%	16,50%	20,30%	56,20%	13	0	5	8	736	3701	4	45	4,40%	63,70%	2	3	1	**110**
	MobileNetV2	42,00%	52,00%	35,10%	50,90%	75,30%	13	1	7	5	775	2283	6	37	34,10%	72,70%	3	5	2	**30**
		41,50%	55,40%	33,10%	40,10%	67,00%	13	1	6	6	916	2785	5	48	20,20%	69,60%	2	5	2	**50**
		29,70%	37,90%	24,50%	37,80%	58,60%	13	1	3	9	1242	2890	3	59	11,00%	68,20%	2	3	2	**80**
		34,40%	48,00%	26,80%	31,70%	56,80%	13	0	5	8	1121	3174	2	44	7,50%	67,60%	2	2	2	**110**
	CSRT																			
		IDF1	**IDP**	**IDR**	**RCLL**	**PRCN**	**GT**	**MT**	**PT**	**ML**	**FP**	**FN**	**IDS**	**FM**	**MOTA**	**MOTP**	**IDT**	**IDA**	**IDM**	
	YOLOv3	42,80%	49,20%	37,90%	57,80%	75,10%	13	1	7	5	892	1960	10	48	38,40%	71,70%	8	6	4	**30**
		48,50%	58,90%	41,20%	51,20%	73,30%	13	1	8	4	866	2265	9	39	32,40%	70,30%	3	7	1	**50**
		30,80%	38,30%	25,70%	40,50%	60,30%	13	0	7	6	1239	2765	6	50	13,70%	67,80%	3	6	3	**80**
		35,50%	46,30%	28,80%	39,50%	63,40%	13	0	7	6	1058	2812	4	53	16,60%	66,90%	3	2	1	**110**
	YOLOv3-tiny	28,60%	45,20%	20,90%	33,90%	73,30%	13	0	7	6	574	3072	10	44	21,30%	68,50%	4	8	3	**30**
		27,40%	46,90%	19,30%	28,50%	69,00%	13	0	6	7	594	3324	8	44	15,50%	67,30%	4	6	2	**50**
		20,80%	32,60%	15,20%	25,10%	53,70%	13	0	3	10	1002	3482	9	48	3,30%	64,10%	3	8	2	**80**
		23,00%	43,30%	15,60%	19,80%	54,70%	13	0	5	8	761	3727	5	46	3,30%	65,40%	1	4	0	**110**
	MobileNetV2	45,60%	56,30%	38,30%	50,30%	74,00%	13	1	8	4	820	2310	8	35	32,50%	72,10%	3	6	2	**30**
		38,10%	51,00%	30,50%	40,40%	67,70%	13	1	6	6	898	2767	7	47	21,00%	68,80%	4	5	2	**50**
		29,60%	37,90%	24,30%	38,80%	60,40%	13	1	5	7	1179	2844	3	54	13,30%	66,70%	2	3	2	**80**
		37,70%	53,10%	29,20%	32,20%	58,70%	13	0	4	9	1055	3148	2	48	9,50%	68,20%	3	1	2	**110**
	KCF																			
		IDF1	**IDP**	**IDR**	**RCLL**	**PRCN**	**GT**	**MT**	**PT**	**ML**	**FP**	**FN**	**IDS**	**FM**	**MOTA**	**MOTP**	**IDT**	**IDA**	**IDM**	
	YOLOv3	42,60%	48,60%	37,90%	60,40%	77,60%	13	1	8	4	810	1838	13	38	42,70%	73,80%	9	8	4	**30**
		47,50%	57,70%	40,30%	52,40%	74,90%	13	1	8	4	814	2213	10	24	34,60%	72,30%	4	7	1	**50**
		34,40%	42,50%	28,90%	44,80%	65,90%	13	1	5	7	1077	2563	8	35	21,50%	69,20%	3	6	1	**80**
		37,90%	49,00%	30,90%	41,90%	66,50%	13	0	8	5	981	2698	6	43	20,70%	68,80%	3	5	2	**110**
	YOLOv3-tiny	30,10%	47,40%	22,00%	35,00%	75,40%	13	0	7	6	531	3019	9	27	23,40%	70,20%	3	9	3	**30**
		28,50%	48,70%	20,10%	28,60%	69,40%	13	0	6	7	586	3316	7	29	15,90%	69,00%	3	6	2	**50**
		21,30%	33,50%	15,60%	27,40%	58,90%	13	0	4	9	887	3374	10	26	8,10%	66,10%	2	9	1	**80**
		24,40%	45,10%	16,70%	21,80%	58,90%	13	0	5	8	707	3632	6	28	6,50%	66,80%	2	4	0	**110**
	MobileNetV2	47,20%	58,70%	39,50%	50,30%	74,70%	13	1	7	5	791	2309	6	22	33,10%	74,90%	3	5	2	**30**
		45,30%	60,60%	36,20%	43,50%	72,70%	13	1	6	6	757	2626	5	16	27,10%	71,20%	1	5	1	**50**
		44,10%	56,10%	36,30%	42,90%	66,40%	13	1	4	8	1011	2652	3	36	21,10%	68,90%	2	3	2	**80**
		44,80%	63,10%	34,70%	38,30%	69,70%	13	1	4	8	773	2866	1	48	21,70%	67,10%	2	1	2	**110**
	MOSSE																			
		IDF1	**IDP**	**IDR**	**RCLL**	**PRCN**	**GT**	**MT**	**PT**	**ML**	**FP**	**FN**	**IDS**	**FM**	**MOTA**	**MOTP**	**IDT**	**IDA**	**IDM**	
	YOLOv3	35,40%	40,30%	31,60%	53,80%	68,50%	13	0	9	4	1147	2147	13	58	28,80%	71,60%	7	9	4	**30**
		27,80%	33,60%	23,70%	40,40%	57,20%	13	0	6	7	1407	2769	12	50	9,90%	70,50%	3	10	1	**50**
		23,80%	29,00%	20,20%	28,90%	41,50%	13	0	5	8	1898	3302	13	25	-12,20%	68,10%	4	10	1	**80**
		21,80%	28,00%	17,80%	26,30%	41,40%	13	0	8	5	1729	3422	10	28	-11,10%	68,00%	5	7	2	**110**
	YOLOv3-tiny	24,60%	38,80%	18,00%	30,80%	66,20%	13	0	7	6	729	3217	12	39	14,80%	68,50%	5	9	3	**30**
		15,40%	25,70%	11,00%	19,00%	44,40%	13	0	4	9	1109	3761	12	35	-5,10%	66,50%	5	9	2	**50**
		16,00%	25,60%	11,70%	18,40%	40,30%	13	0	3	10	1268	3790	8	24	-9,00%	64,00%	0	8	0	**80**
		15,60%	29,30%	10,60%	13,40%	37,00%	13	0	3	10	1058	4024	6	16	-9,50%	66,40%	2	5	1	**110**
	MobileNetV2	33,50%	40,40%	28,50%	44,60%	63,20%	13	0	8	5	1205	2575	14	41	18,30%	71,50%	4	10	2	**30**
		26,70%	35,40%	21,50%	33,50%	55,30%	13	0	6	7	1259	3090	11	40	6,20%	69,80%	1	9	1	**50**
		18,40%	22,60%	15,50%	26,30%	38,50%	13	0	6	7	1955	3424	10	29	-16,00%	68,60%	1	10	1	**80**
		11,80%	16,50%	9,20%	17,10%	30,60%	13	0	3	10	1795	3853	9	33	-21,80%	66,80%	5	7	3	**110**

Fonte: Do autor.

Tabela A.12: Resultados gerais das métricas de rastreamento do MOTChallenge referente aos quatro vídeos utilizados para o período de detecção de trinta *frames*.

Detector \ Métrica	IDF1	IDP	IDR	RCLL	PRCN	GT	MT	PT	ML	FP	FN	IDS	FM	MOTA	MOTP	IDT	IDA	IDM	Rastreador \ Métrica
YOLOv3	29,25%	46,60%	22,35%	37,98%	79,48%	38	1	15	22	2090	9206	26	112	26,70%	71,88%	11	20	5	
YOLOv3-tiny	21,50%	57,55%	14,48%	24,33%	78,33%	38	0	10	28	1043	12181	16	43	18,35%	69,18%	5	14	3	**dlib**
MobileNetV2	31,15%	50,88%	23,60%	40,00%	80,73%	38	1	14	23	1700	10087	17	67	30,60%	73,88%	**3**	16	**2**	
YOLOv3	29,80%	47,15%	22,88%	37,58%	78,25%	38	1	15	22	2201	9256	23	96	25,65%	72,20%	10	18	5	
YOLOv3-tiny	21,63%	**58,33%**	14,55%	24,03%	78,43%	38	0	9	29	**1039**	12197	15	55	17,78%	69,65%	4	**13**	3	**CSRT**
MobileNetV2	**32,58%**	53,08%	**24,80%**	**40,30%**	**81,53%**	38	1	15	22	1693	10101	19	63	**31,28%**	73,88%	4	17	3	
YOLOv3	30,03%	47,65%	23,03%	38,23%	78,98%	38	**2**	15	**21**	2001	**9086**	25	85	26,93%	73,03%	11	19	5	
YOLOv3-tiny	21,65%	57,38%	14,63%	23,93%	77,13%	38	0	9	29	1051	12199	**14**	**37**	17,60%	70,63%	4	14	4	**KCF**
MobileNetV2	31,65%	51,53%	24,10%	39,75%	80,30%	38	1	13	24	1678	10144	17	44	30,43%	**75,23%**	5	16	4	
YOLOv3	28,28%	45,83%	21,45%	36,58%	76,93%	38	1	16	**21**	2276	9383	25	99	23,63%	71,10%	7	21	4	
YOLOv3-tiny	20,03%	53,50%	13,50%	22,98%	74,28%	38	0	10	28	1184	12331	16	48	15,63%	69,80%	5	**13**	3	**MOSSE**
MobileNetV2	28,65%	47,43%	21,65%	38,28%	77,55%	38	0	**17**	**21**	2045	10294	25	59	26,38%	72,98%	4	21	**2**	

Fonte: Do autor.