BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

DETECÇÃO DE MICRORGANISMOS AMBIENTAIS EM IMAGENS MICROSCÓPICAS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

CONRADO COSTA NUNES

Rio Verde, GO

2025

INSTITUTO FEDERAL GOIANO - CAMPUS RIO VERDE
BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

DETECÇÃO DE MICRORGANISMOS AMBIENTAIS EM IMAGENS MICROSCÓPICAS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

CONRADO COSTA NUNES

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Goiano - Campus Rio Verde, como requisito parcial para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Prof. Douglas Cedrim Oliveira

Rio Verde, GO

Fevereiro, 2025

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema Integrado de Bibliotecas do IF Goiano - SIBi

Costa Nunes, Conrado
C838d Detecção de microrganismos ambientais em imagens microscópicas utilizando redes neurais convolucionais / Conrado Costa Nunes. Rio Verde 2025.

52f. il.

Orientador: Prof. Dr. Douglas Cedrim Oliveira.
Tcc (Bacharel) - Instituto Federal Goiano, curso de 0219201 - Bacharelado em Ciência da Computação - Integral - Rio Verde (Campus Rio Verde).
1. Microorganismos ambientais. 2. Detecção de objetos. 3. Aumento de dados. 4. Yolo. 5. EMDS5. I. Título.

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

TERMO DE AUTORIZAÇÃO PARA DISPONIBILIZAR PRODUÇÃO TÉCNICA NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Repositório Institucional do IF Goiano - RIIF Goiano Sistema Integrado de Bibliotecas

- Profissional de Educação do IF Goiano -

Com base no disposto na Lei Federal nº 9.610/98, e manual sobre a Produção Técnica, publicado pela DAV/CAPES/MEC*, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano, a disponibilizar gratuitamente o documento no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada eletronicamente abaixo, em formato digital para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

Identificação da Produção Técnica – DAV/CAPES

[ ] Editoria [ ] Material Didático

[ ] Curso de Formação Profissional [ ] Projetos de Extensão à Comunidade

[ ] Relatório Técnico Conclusivo [ ] Atividade Técnica/Tecnológica

[ ] Disseminação do Conhecimento Técnico/Tecnológico [ ] Produto Bibliográfico

[X] Outras Produções Técnicas - Tipo: TCC (Graduação)

Nome Completo do Autor/a: Conrado Costa Nunes

Matrícula: 2018102201910094

Título do Trabalho: Detecção de microrganismos ambientais em imagens microscópicas utilizando redes neurais convolucionais

Restrições de Acesso ao Documento

Documento confidencial: [X] Não [ ] Sim

Justifique: _____

Informe a data que poderá ser disponibilizado no RIIF Goiano: 18 / 02 / 2025

O documento está sujeito a registro de patente? [ ] Sim [X] Não

O documento pode vir a ser publicado como livro e/ou artigo? [ ] Sim [X] Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O/A referido/a docente e/ou autor/a declara que:

1 - o documento é seu trabalho original, detém os direitos autorais da produção técnica e não infringe os direitos de qualquer outra pessoa ou entidade;

2 - obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autor/a, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;

3 - cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Rio Verde, 18 de fevereiro de 2025.

(Assinado Eletronicamente)

Conrado Costa Nunes (Autor)

(Assinado Eletronicamente)

Douglas Cedrim Oliveira (Orientador)

1058004

(Assinatura do Docente, Autor e/ou Detentor dos Direitos Autorais)

Documento assinado eletronicamente por:

- **Douglas Cedrim Oliveira, PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 18/02/2025 10:19:27.
- **Conrado Costa Nunes, 2018102201910094 - Discente**, em 18/02/2025 10:20:10.

Este documento foi emitido pelo SUAP em 18/02/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifgoiano.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 677328
Código de Autenticação: 58a136720b

INSTITUTO FEDERAL GOIANO
Campus Rio Verde
Rodovia Sul Goiana, Km 01, Zona Rural, 01, Zona Rural, RIO VERDE / GO, CEP 75901-970
(64) 3624-1000

Regulamento de Trabalho de Conclusão de Curso (TCC) – IF Goiano - Campus Rio Verde

ANEXO V - ATA DE DEFESA DE TRABALHO DE CURSO

Aos sete dias do mês de fevereiro de dois mil e vinte e cinco às quatorze horas, reuniu-se a Banca Examinadora composta por: Prof. Dr. Douglas Cedrim Oliveira (orientador), Profa. Dra. Heyde Francielle do Carmo França (membro interno) e Profa. Ma. Andrea Barboza Proto Sardi (membro interno), para examinar o Trabalho de Conclusão de Curso (TCC) intitulado "Detecção de microrganismos ambientais em imagens microscópicas utilizando redes neurais convolucionais" de Conrado Costa Nunes, estudante do curso de bacharelado em Ciência da Computação do IF Goiano – Campus Rio Verde, sob Matrícula nº 2018102201910094. A palavra foi concedida ao estudante para a apresentação oral do TC, em seguida houve arguição do candidato pelos membros da Banca Examinadora. Após tal etapa, a Banca Examinadora decidiu pela APROVAÇÃO do estudante. Ao final da sessão pública de defesa foi lavrada a presente ata, que segue assinada pelos membros da Banca Examinadora.

Rio Verde, 07 de fevereiro de 2025.

(Assinado eletronicamente)

Douglas Cedrim Oliveira

Orientador(a)

(Assinado eletronicamente)

Heyde Francielle do Carmo França

Membro da Banca Examinadora

(Assinado eletronicamente)

Andrea Barboza Proto Sardi

Membro da Banca Examinadora

Observação:

Documento assinado eletronicamente por:

- **Douglas Cedrim Oliveira**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 07/02/2025 19:27:34.
- **Heyde Francielle do Carmo Franca**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 07/02/2025 22:02:09.
- **Andrea Barboza Proto Sardi**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 10/02/2025 12:26:14.

Este documento foi emitido pelo SUAP em 07/02/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifgoiano.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 673298
Código de Autenticação: ed0a202c43

Dedico esse trabalho a minha família e amigos, que me deram todo o suporte e apoio para concluir essa longa jornada.

AGRADECIMENTOS

Em primeiro lugar, a Deus por me dar a honra e a oportunidade de ter chegado até aqui, aos meus pais, que dedicaram suas vidas para cuidar de mim e me ajudar a me tornar quem sou hoje, por sempre me apoiarem e me incentivarem nas minhas escolhas.

Aos professores, pelos seus ensinamentos. Em especial, ao professor Douglas Cedrim Oliveira, por ter sido meu orientador com dedicação e amizade. E a todos os meus amigos que sempre se disponibilizaram a me ajudar.

RESUMO

NUNES, Conrado Costa. **DETECÇÃO DE MICRORGANISMOS AMBIENTAIS EM IMAGENS MICROSCÓPICAS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS.** Fevereiro, 2025. 52 f. Monografia – (Curso de Bacharel em Ciência da Computação), Instituto Federal Goiano - Campus Rio Verde. Rio Verde, GO.

O estudo de microrganismos ambientais é fundamental para caracterizar a biodiversidade encontrada em lugares diferentes, como por exemplo solo e rios. O isolamento de microrganismos e identificação de sua espécie é uma área bastante desafiadora, e que tem sido cada vez mais multidisciplinar. Uma tarefa comum é utilizar características encontradas na forma do microrganismo (morfologia) para efetuar a identificação e classificação da espécie. Essa tarefa pode ser feita manualmente ou utilizando técnicas de processamento digital de imagens e visão computacional.
Este trabalho propõe a utilização de uma rede neural convolucional para detecção de objetos com o objetivo de identificar microrganismos ambientais em imagens microscópicas, através da utilização de uma técnica de aprendizado de máquina profundo e uma base de dados pública de imagens de microrganismos ambientais.
Uma vez que a quantidade de imagens é importante para obter bons resultados nesse processo, foram realizados diversos pré-processamentos nas imagens com o objetivo de aumentar o número de imagens da base de dados original (*data augmnentation*). Com isso, foi realizado o treinamento do modelo com ambos os conjuntos, original e aumentado, e ao fim do treinamento foram realizadas validações e testes com as métricas de avaliação indicando que é possível obter bons resultados de detecção de microrganismos ambientais com a metodologia proposta.

Palavras-chave: Microorganismos ambientais; Detecção de objetos; Aumento de dados; Yolo; EMDS5

ABSTRACT

NUNES, Conrado Costa. **DETECÇÃO DE MICRORGANISMOS AMBIENTAIS EM IMAGENS MICROSCÓPICAS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS.** Fevereiro, 2025. 52 f. Trabalho de Conclusão de Curso – Bacharel em Ciência da Computação, Instituto Federal Goiano - Campus Rio Verde. Rio Verde, GO, Fevereiro, 2025.

The study of environmental microorganisms is essential to characterize the biodiversity found in different places, such as soil and rivers. Isolating microorganisms and identifying their species is a very challenging area, and one that has become increasingly multidisciplinary. A common task is to use characteristics found in the shape of the microorganism (morphology) to identify and classify the species. This task can be done manually or using digital image processing and computer vision techniques.
This work proposes the use of a convolutional neural network for object detection with the aim of identifying environmental microorganisms in microscopic images, through the use of a deep machine learning technique and a public database of images of environmental microorganisms.
Since the number of images is important to obtain good results in this process, several pre-processings were performed on the images with the aim of increasing the number of images in the original database (data augmentation). With this, the model was trained with both sets, original and augmented, and at the end of the training, validations and tests were carried out with the evaluation metrics, indicating that it is possible to obtain good results in detecting environmental microorganisms with the proposed methodology.

Keywords: Environmental microorganisms; object detection; data augmentation; Yolo; EMDS5

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AP	Average Precision
CNN	Redes neurais convolucionais (do inglês *Convolutional Neural Networks*)
DA	Data Augmentation
EMDS	*Environmental Microorganism Dataset*
GT	*Ground Truth*
GTM	*Ground Truth Multiple-object image segmentation*
GTS	*Ground Truth Single-object image segmentation*
IA	Inteligência Artificial
IoU	Interseção sobre união (do inglês *Intersection over Union*
MA	Microrganismos ambientais
mAP	Mean Average Precision
MLP	Multilayer percepctron
RCNN	Redes neurais convolucionais baseadas em região (do inglês *Region Based Convolutional Neural Networks*)
RoI	Regiões de interesse (do inglês, *Region of Interest*)
SVM	Support Vector Machine
VANT	Veículo aéreo não tripulado
VP	Verdadeiro positivo
FN	Falso negativo
FP	Falso positivo
VN	Verdadeiro negativo
VOC	Visual Object Classes Challenge
YOLO	You Only Look Once

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

O estudo de Microrganismos Ambientais (MAs) tem se tornado cada vez mais necessário com o passar dos anos. Li et al. (2021) explicam que os MAs fazem parte do meio ambiente sendo que alguns são benéficos e outros podem trazer malefícios, que podem afetar a nossa saúde física, ou seja, são patogênicos. Cita como exemplo a Rotífera que é um MA comum e que se encontra em grandes quantidades em lagos, lagoas, rios e demais corpos de águas salobras. Dessa forma, estudar esses microrganismos pode ter impacto nas nossas vidas, sendo tema recorrente de pesquisa em saúde pública.

A Sociedade Americana de Microbiologia, sediada nos Estados Unidos, destaca a importância de estudar os MAs, dando um exemplo significativo de um incidente ocorrido em 1976, envolvendo mais de 4.000 veteranos da Segunda Guerra Mundial que participaram de uma viagem e após o retorno, muitos desses veteranos começaram a manifestar sintomas semelhantes, como: dores no peito, febre, congestão pulmonar e pneumonia (AMERICAN ACADEMY FOR MICROBIOLOGY, 2004). Ao longo dos anos, milhares de americanos desenvolveram uma variedade de sintomas semelhantes. Todos esses casos tinham algo em comum: eram causados por agentes patogênicos ambientais. Portanto, a investigação da incidência e epidemiologia dessas doenças ambientais torna-se crucial para compreender os organismos e os mecanismos que perpetuam essas enfermidades, isso permite avaliar adequadamente a extensão da ameaça representada por esses agentes e com esses resultados coletados, torna-se possível avaliar os riscos para o desenvolvimento de respostas que auxiliem no combate desses patógenos no meio ambiente.

Li et al. (2021) aborda o desafio enfrentado pelos pesquisadores ao coletarem microrganismos ambientais (MAs), pois estão em ambientes externos sujeitos às mudanças frequentes de temperatura e salinidade. Devido à sensibilidade desses microrganismos a essas condições variáveis, a qualidade dos MAs observados muitas vezes é comprometida, dificultando sua coleta. Como resultado, os pesquisadores enfrentam dificuldades na construção de conjuntos de dados de imagens desses microrganismos. Embora existam conjuntos de dados de MAs disponíveis atualmente, muitos deles não são públicos, o que dificulta o desenvolvimento de novas pesquisas, uma vez que uma quantidade menor de pesquisadores terão acesso às bases de dados dessas imagens, e devido ao fato que montar novos conjuntos de dados de imagens demanda um tempo e complexidade consideráveis para realizar a coleta e aquisição das imagens (LI et al., 2021).

A série de conjunto de dados de imagens *Environmental Microorganism Dataset* (EMDS) possui as suas primeiras quatro versões (EMDS-1 a EMDS-4) restritas ao uso privado. Porém, a partir da quinta versão, EMDS-5, todas as imagens das 21 classes que foram utilizadas no EMDS-4 foram tornadas públicas (LI et al., 2021), o que possibilita análises de MAs por outros grupos de pesquisa.

A microscopia óptica tem sido uma das principais técnicas para efetuar a aquisição de imagens de microrganismos, ajudando a caracterizar e compreender o mundo microbiano (JECKEL; DRESCHER, 2021). Uma vantagem chave da microscopia em relação a outras técnicas de caracterização de micróbios consiste na possibilidade de adquirir imagens de células vivas com alta resolução espacial. Além disso, a partir do uso de proteínas fluorescentes foi possível então rotular especificamente componentes particulares de células e também estudar funções celulares usando microscopia. Depois da etapa de aquisição (coleta) de imagens, a extração das propriedades quantitativas (características) se torna uma etapa extremamente importante. As análises dessas imagens dependiam muita das vezes da quantificação manual. Porém, essa análise manual apresentava algumas dificuldades quanto à precisão, pois era limitada, e considerada impraticável, quando havia centena ou milhares de imagens para serem analisadas.

Para auxiliar nesta análise técnicas de processamento digital de imagens e visão computacional são comumente aplicadas, permitindo automatizar e processar rapidamente grandes quantidades de imagens. Mais especificamente, técnicas de segmentação de imagens e extração de características de imagens são bastante comuns para esse propósito de localização e classificação de espécies de microrganismos ambientais (GINORIS et al., 2007).

Ao longo do tempo diversos trabalhos passaram a utilizar técnicas de inteligência artificial (IA) e aprendizado de máquina para obter melhores na localização e classificação. Em alguns casos a capacidade dos algoritmos aproxima ou ultrapassa a taxa de acerto humana nesta tarefa (ZHANG et al., 2023). Essas características extraídas das imagens geralmente passavam por algoritmos de treinamento supervisionado, como por exemplo: redes neurais multicamadas, *k-nearest neighbors* e *support vector machines (SVM).*

É possível ver que nos últimos anos a utilização destas tecnologias se desenvolveu muito rápido, com isso alcançando desempenhos excepcionais em diversas áreas de análise e processamento de imagens, sendo assim podendo realizar trabalhos com alto nível de dificuldade e que levam muito tempo para serem realizados. Mais especificamente trabalhos que utilizam aprendizado de máquina profundo (*deep learning*), onde as características são extraídas diretamente de acordo com a arquitetura da rede, mostram-se com um grande potencial na análise de imagens de microrganismos (ZHANG et al., 2023).

Deste forma, o objetivo principal deste trabalho consiste na utilização de conjuntos de dados públicos de imagens de microrganismos ambientais combinado a técnicas de aprendizado de máquina profundo para detectar microrganismos presentes na imagem, ou seja, a sua localização na imagem bem como sua classificação. Adicionalmente, pretende-se investigar se o aumento dessa base de dados pode contribuir para melhores resultados na detecção.

O trabalho está estruturado da seguinte forma: No **Capítulo 2** será falado da fundamentação teórica, ou seja, serão apresentadas as definições referentes aos conceitos e

elementos utilizados no trabalho. No **Capítulo 3** será falado sobre os trabalhos relacionados utilizados para o desenvolvimento deste trabalho. No **Capítulo 4** será falado sobre os materiais e métodos que foram utilizados para execução do trabalho. No **Capítulo 5** serão mostrados os resultados obtidos e discussões. No **Capítulo 6** será apresentada a conclusão obtida após análise dos resultados e apresentados algumas discussões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Visão computacional

Também conhecida como Análise de Imagens por alguns autores, a área de Visão Computacional é tida como uma subárea da Computação Gráfica. Gomes e Velho (2004) em seu livro indicam que a subárea pode ser definida de acordo com a natureza dos dados de entrada e saída. No caso de visão computacional a entrada são imagens e a saída são dados extraídos dessas imagens, podendo ser dados geométricos, físicos ou de uma natureza mais subjetiva, como reconhecer uma pessoa, um objeto, um animal, por exemplo.

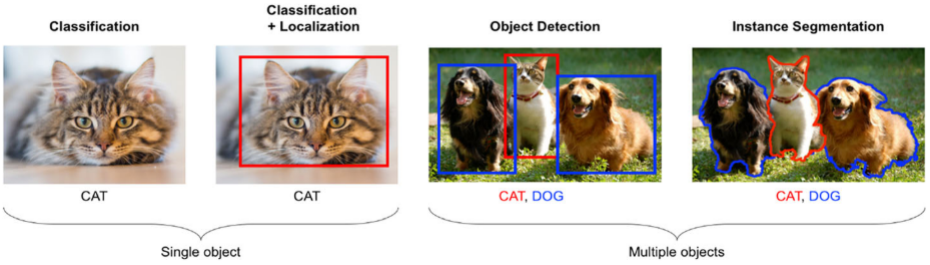
Em Schaeffer-Filho et al. (2022) a visão computacional é descrita como uma área interdisciplinar que reúne elementos de inteligência artificial, óptica e processamento de imagens, ou seja, é um ramo que tem como objetivo fazer com que máquinas obtenham a capacidade de extrair e interpretar informações de forma significativa a partir de imagens ou vídeos, com isso, realizando diversas tarefas como reconhecimento fácil, detecção de imagens, entre outras.

Schaeffer-Filho et al. (2022) ainda citam que boa parte dos grandes avanços nos últimos anos de áreas da visão computacional foram devido à utilização de aprendizado profundo (*deep learning*), por conta da capacidade de conseguir aprender automaticamente padrões complexos utilizando um grande conjunto de dados. Com essa sinergia entre o aprendizado e visão computacional acabou ampliando as possibilidade de aplicações, bem como a diminuição de erros em aplicações que já eram comuns como por exemplo a detecção de objetos, a qual foi utilizada neste trabalho e será tratado com mais detalhes nas demais seções.

Liu et al. (2022) pontua como objetivo da detecção de objetos: identificar a localização, o tamanho, a relação espacial e classes de objetos específicos presentes em imagens ou vídeos. Cita que a detecção é um domínio cruzado entre a visão computacional, o processamento digital de imagens e a percepção humana.Possui uma grande diversidade de aplicações, como: contagem automática de células, reconhecimento facial, detecção de defeitos industriais, detecção de aviação (Veículo aéreo não tripulado - VANT), detecção de tráfego de veículos e detecção e contagem de pedestres.

Com essas definições fica nítido que a quantidade de aplicações nessa área é bastante vasta. Na Figura 1 são ilustrados alguns exemplos de aplicações. Seja considerando um único objeto: classificação e localização de objetos; ou considerando vários objetos: detecção e segmentação de cada um dos objetos.

Figura 1 – Exemplos de classificação, localização, detecção de objetos e segmentação.



Fonte: Extraída de Diwan, Anirudh e Tembhurne (2023).

2.1.1 Classificação de objetos

Classificação de objetos tem como objetivo denominar um único objeto presente em uma imagem, ou seja, dado uma determinada imagem como entrada de dado e espera como retorno de saída uma determinada classe que represente aquele objeto.

Analisando a Figura 1 é possível notar um exemplo de classificação na primeira imagem, onde, dado uma imagem como entrada traz como saída a classe e neste caso o retorno foi a classe ”CAT”. Para que seja possível determinar se a classificação esta correta é necessário avaliar esta classificação e para avaliar este desempenho são utilizados diversas métricas, que serão explicadas algumas que foram utilizadas neste trabalho na Seção 2.3.

2.1.2 Localização de objetos

Diferente da classificação de objetos a localização de objetos tem como objetivo encontrar um único objeto presente em uma imagem com o auxilio de uma caixa delimitadora (*bounding box*), ou seja, dado uma imagem como entrada de dado é esperado como retorno de saída uma caixa delimitadora em volta do objeto.

Analisando a Figura 1 é possível notar um exemplo de localização na segunda imagem, onde, dado uma imagem como entrada traz como saída uma caixa delimitadora em torno do objeto.

2.1.3 Detecção de objetos

Detecção de objetos engloba tanto a classificação quanto a localização, porém, a detecção de objetos classifica e localiza diversos objetos presente em uma imagem, ou seja, dado uma imagem como entrada de dado a detecção de objetos traz como saída a denominação da classe daquele objeto e traz também como saída uma caixa delimitadora em volta dos objetos.

Analisando a Figura 1 é possível notar um exemplo de detecção na terceira imagem, onde, dado uma imagem como entrada traz como saída uma caixa delimitadora (bounding box) em torno do objeto e a classe que a arquitetura previu que seria pertencente a aquele

objeto.

E assim como na classificação de objetos, também é utilizado diversas métricas para avaliar o desempenho da detecção de objetos e será tratado sobre as métricas utilizadas para este trabalho na Seção 2.4.

2.1.4 Segmentação de instâncias

Segmentar uma imagem corresponde a dividir a imagem em regiões diversas de interesse (GONZALEZ; WOODS, 2009). Um exemplo a partir da Figura 1 à esquerda seria dividir a imagem em duas regiões: uma delimitando o gato e outra delimitando o resto da imagem, ou seja, o *background*.

Diwan, Anirudh e Tembhurne (2023) discute a segmentação de instâncias de forma que ela se assemelha à detecção de objetos, envolvendo tanto a classificação quanto a localização de múltiplos objetos, como pode ser visto na Figura 1 à direita. Porém, ela difere da detecção de forma que não delimitará uma caixa (*bounding box*) mas sim um contorno mais próximo do objeto.

Neste trabalho, não faremos a segmentação de instâncias, mas usaremos uma base de dados que já disponibiliza imagens segmentadas (LI et al., 2021).

2.2 Aprendizado profundo para detecção de objetos

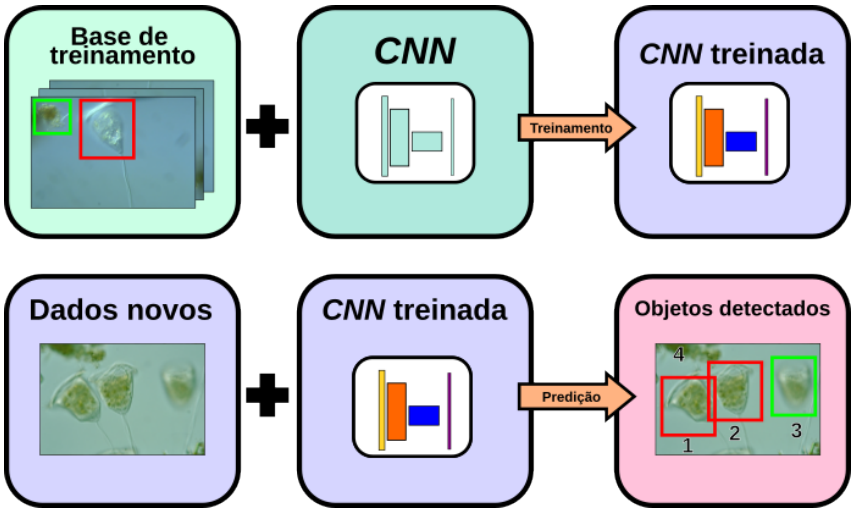
Awan (2024) define a aprendizagem profunda como um aprendizado de máquina que possui como objetivo ensinar os computadores a executar tarefas aprendendo a partir de exemplos assim como os seres humanos. Em vez de passar para o treinamento explicitamente as características de um objeto, são passadas diversas imagens similares para que o computador aprenda através das características presentes nas imagens.

Na detecção de imagens não são passadas como entrada somente as imagens para que o modelo aprenda pelas suas características, são passados também os dados da caixa delimitadora (*bounding box*) do(s) objeto(s). Após essa etapa de treinamento são passadas novas imagens para o modelo já treinado para que assim ele realize as predições.

O modelo pode detectar de forma correta ou não, sendo que também existe a possibilidade de o modelo não conseguir realizar a detecção. Esse processo está representado na Figura 2, onde podemos ver, após a predição, três casos diferentes: 1) um objeto foi detectado e corretamente classificado (casos 1 e 2); o objeto foi detectado porém com classificação errada (caso 3, *bounding box* em verde mas era para estar em vermelho); 3) modelo não conseguiu realizar a detecção do objeto (caso 4).

Jeckel e Drescher (2021) citam que quando contornado a situação das limitações das abordagens tradicionais de análise de imagens, os métodos de aprendizagem profunda, como por exemplos as redes neurais convolucionais (*convolutional neural networks - CNN*), melhoram muito a precisão da segmentação, não somente em imagens microscópicas,

Figura 2 – Exemplo de utilização de aprendizado de máquina profundo (*deep learning*) para detecção de objetos.



Fonte: Adaptada de Jeckel e Drescher (2021).

mas também quando se trata de outras imagens em geral. Com uma gama de dados para treinamento e muitos recursos computacionais disponíveis, pode-se alcançar uma segmentação de instâncias altamente precisa com uma diversa e alta variedade de tipos de células. Um exemplo deste processo de segmentação pode ser vista na Figura 1, no contexto de animais. As aplicações das CNNs dentro da análise biológica de imagens são bastante diversas, desde tarefas de classificação à restauração e segmentação de imagens.

Jeckel e Drescher (2021) citam também que em relação ao desempenho das CNNs, ele está intimamente ligado ao design da sua arquitetura. Porém, também depende da qualidade e da quantidade de imagens que estão disponíveis para o treinamento. Para as tarefas de segmentação de objetos, os dados do treinamento devem consistir em um par de uma determinada imagem e sua segmentação correspondente. Porém muita das vezes não são disponibilizadas as segmentações correspondentes das imagens e assim acaba sendo necessário realizar de forma manual ou utilizar um algoritmo de segmentação que seja preciso.

Neste trabalho essa tarefa de utilizar dados adicionais necessários para a detecção (*bounding box*) não foi necessária ser feita de forma manual. Ainda assim foi necessário realizar alguns pré-processamentos, como poderá ser visto no Capítulo 4.

Liu et al. (2022) ainda cita que no últimos anos com o surgimento de atualizações e de placas gráficas (GPUs) de alto desempenho a melhoria e a aparição de diversos conjuntos de dados de grande escala (como DOTA, ImageNet, COCO) e o crescente aumento de redes neurais com redes baseadas em algoritmos convolucionais de detecção de objeto que utilizam técnicas de aprendizagem profunda tornaram-se a corrente principal da tecnologia contemporânea de detecção de objetos e que estes atuais algoritmos de detecção de objetos podem ser divididos em: *algoritmos de um estágio* ou *algoritmos de dois estágios*.

Diwan, Anirudh e Tembhurne (2023) também citam que algoritmos implementados em dois estágios comumente fazem a extração de características e em seguida fazem a classificação e/ou a localização, onde em sua primeira etapa irá gerar as regiões de interesse (RoI) com o auxilio de rede de proposta de região (RPN) e a sua segunda etapa será responsável pela previsão dos objetos e caixas delimitadoras (*bounding box*) para as regiões desejadas. Ou seja, quando se trata de dois estágios as arquiteturas separam para que cada parte do modelo realizem um papel especifico. Park (2021) também fala de detecção em dois estágios, onde em seu trabalho explica que este ramo de dois estágios é derivado da RCNN, e que apesar de serem conhecidos por serem lentos, são bem poderosos e que recentemente com os avanços em compartilhar recursos fez com que melhorasse os detectores de dois estágios para que tenham o seu custo computacional próximo aos dos detectores de estágio único.

Já quando se trata de algoritmos de estágio único, Diwan, Anirudh e Tembhurne (2023) explicam que são arquiteturas mais simples projetadas para realizar a detecção de objetos em um único estágio, ou seja, gerar a regiões de interesse e realizar a previsão dos objetos já adicionando a caixa delimitadora (*bounding box*) no objeto, assim considerando todas as propostas de região levando em conta todos os tamanhos espaciais de uma imagem ao mesmo tempo. Com isso, diferente dos detectores de dois estágios, os detectores de estágio único possuem um custo computacional menor e são mais rápidos.

Diwan, Anirudh e Tembhurne (2023) citam alguns exemplos de arquiteturas que utilizam dois estágios e arquiteturas que utilizam estágio único. As que utilizam dois estágios eles citam alguns modelos mais populares que utilizam redes neurais convolucionais baseadas em região (*Region Based Convolutional Neural Networks - RCNN*), já de estágio único eles citam alguns que utilizam o Single Shot Detector (Detector de disparo único - SSD) como a You Only Look Once (YOLO) e o próprio Single Shot Detector (SSD). Como outros exemplos de algoritmos de estágio único podem ser citados: *Space Pyramid Pooling-Net* (SPP), *Fast-RCNN*, *Faster-RCNN*, *Mask-RCNN*, também incluindo a família de algoritmos YOLO como: YOLOv1, YOLOv2, YOLOv3, YOLOv4, YOLOv5, etc. Liu et al. (2022) indica que apesar de os algoritmos de dois estágios conseguirem atingir uma alta precisão, quando se trata de cenários diários de detecção de objetos o seu tempo de detecção torna difícil de satisfazer o requisito de ser em tempo real.

Liu et al. (2022) também cita em seu artigo que a YOLOv5, que foi utilizada neste trabalho, é um algoritmo com elevada confiabilidade e estabilidade, fácil de implantar e de treinar e que é também um dos algoritmos de detecção de estágio único que possui atualmente a maior precisão. Na Figura 3 é possível ver os resultados obtidos na detecção de rostos, onde foi utilizado o dataset *Wider Face*, que foi construído de forma a possuir nas imagens de rostos uma grande variabilidade de escala, postura, ângulo, iluminação e possíveis oclusões.

Figura 3 – Exemplo de detecção de rostos utilizando a YOLOv5, exibindo as respectivas caixas delimitadoras em vermelho.

Fonte: Adaptada de Liu et al. (2022).

2.2.1 Detecção em dois estágios

Citado por Diwan, Anirudh e Tembhurne (2023) como sendo um modelo de arquitetura de dois estágios, eles explicam que o RCNN utiliza um algoritmo de busca seletiva para que extraiam as propostas de regiões, ou seja, o RCNN é um algoritmo de detecção de objetos de rede neural convolucional que se baseada em regiões. Ainda é explicado que no artigo que a sua arquitetura é separa em três módulos, sendo que no primeiro é quando se gera as propostas de região com a utilização de um algoritmo de busca seletiva, no segundo cada uma das propostas de região são passadas pela a arquitetura contendo cinco camadas convolucionais e em seguida por duas camadas densas com isso gerando um vetor de características com tamanho de 4096 e por fim no terceiro modulo é onde possui os classificadores lineares independentes e pré-treinados para cada classe, assim passando o vetor de características pelos classificadores lineares para obter as pontuações de classe para que no final a supressão não máxima seja aplicada em todas as pontuações para obter o melhor ajuste

2.2.2 Detecção em estágio único

Citado por Diwan, Anirudh e Tembhurne (2023) como sendo um dos modelos de arquitetura estágio único, eles explicam que o autores da Yolo reformularam o problema

de detecção de objetos como um problema de regressão ao invés de um problema de classificação, assim criando a Yolo. Explicam em seu artigo que na Yolo uma rede neural convolucional prevê as caixas delimitadoras (bounding box) e as probabilidades de classe para todos os objetos que estão presentes em uma determinada imagem, ou seja, a CNN funciona de maneira que propague os recursos de baixo nível com eficiência das camadas convolucionais iniciais para as camadas convolucionais posteriores em uma CNN profunda.

A Yolo é uma arquitetura de identificação e classificação de objetos em tempo real, diferentemente de outras redes neurais nela é aplicada apenas uma rede neural a toda imagem e ela divide está imagem em regiões e faz a previsão de caixas que delimitam e probabiliza cada região.

Com isso Jiang et al. (2022) cita que a Yolo se tornou um algoritmo viral e muito utilizado e é famosa devido sua característica de detecção e uma estrutura simples, assim conseguindo gerar diretamente a posição e categoria da caixa delimitadora por meio de redes neurais.

Redmon et al. (2016) falam em seu artigo da detecção unificada da Yolo, de seu design de rede e do treinamento realizado em seu trabalho. Para a detecção em estágio único, a rede utiliza recursos da imagem para que possa prever diversas caixas delimitadoras (*bounding boxes*), de todas as classes de forma simultânea, e manter apenas as caixas que sejam representativas para cada classe. Com isso a rede irá conseguir analisar toda a imagem e todos os objetos presentes nela, permitindo um treinamento completo e uma velocidade elevada em tempo real mantendo assim uma alta precisão.

Para isso é realizada a divisão da imagem de entrada em uma grade de resolução SxS, caso ocorra de a grade estar no centro do objeto essa grade fica então responsável por detectar aquele objeto. Para que isso ocorra eles explicam que cada uma das células é responsável por prever B caixas delimitadoras e cada caixa gera uma pontuação de confiança que reflete o quão confiante ou preciso aquele modelo esta de que contem um objeto nela e o quão precisa se acredita que a caixa esta prevendo o objeto. Essa confiança é calculada como:

$$\Pr(\text{obj}) \cdot \text{IOU}^{\text{truth pred}}, \tag{1}$$

onde *Pr* é a probabilidade associada a conter o objeto *obj* na célula e a interseção sobre união (IOU) é calculada entre a caixa prevista e o resultado esperado. Além disso, caso não exista nenhum objeto naquela célula as pontuações de confianças devem ser zero. Esse processo é ilustrado na Figura 4.

Em relação a caixa delimitadora, Redmon et al. (2016) explica que é necessário determinar 5 variáveis: (**x**, **y**) representando o centro da caixa em relação ao imagem; **w** (*width*) e **h** (*height*) que são a altura e a largura previstas em relação a toda a imagem; e por último a previsão da confiança que é a interseção sobre união (IOU) entre a caixa que será prevista e qualquer outra caixa que espera-se encontrar. Cada uma das células também irá realizar a previsão da probabilidade de classe condicional C, ou seja, de quais

classes podem pertencer aquela célula e a probabilidade estará condicionada à célula da grade que contenha um objeto.

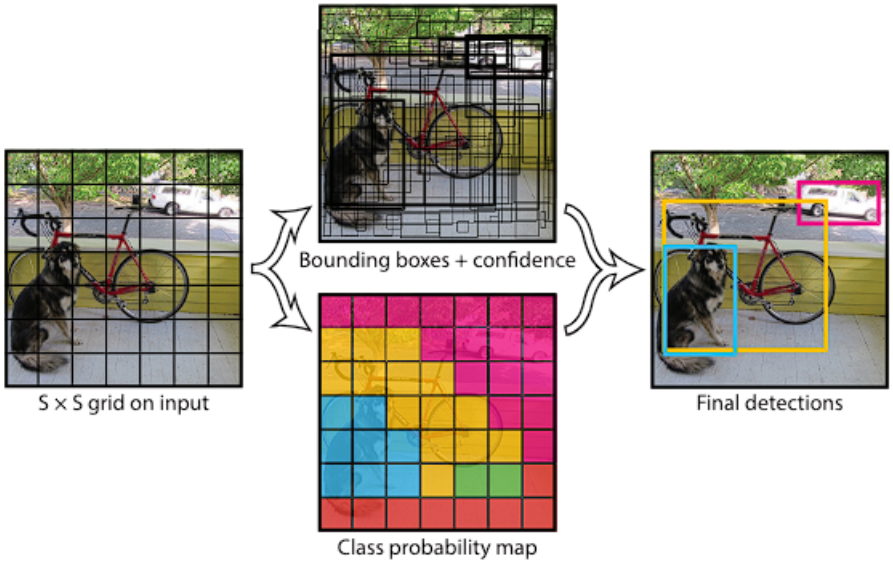


Figura 4 – Imagem retirada do trabalho de Redmon et al. (2016).

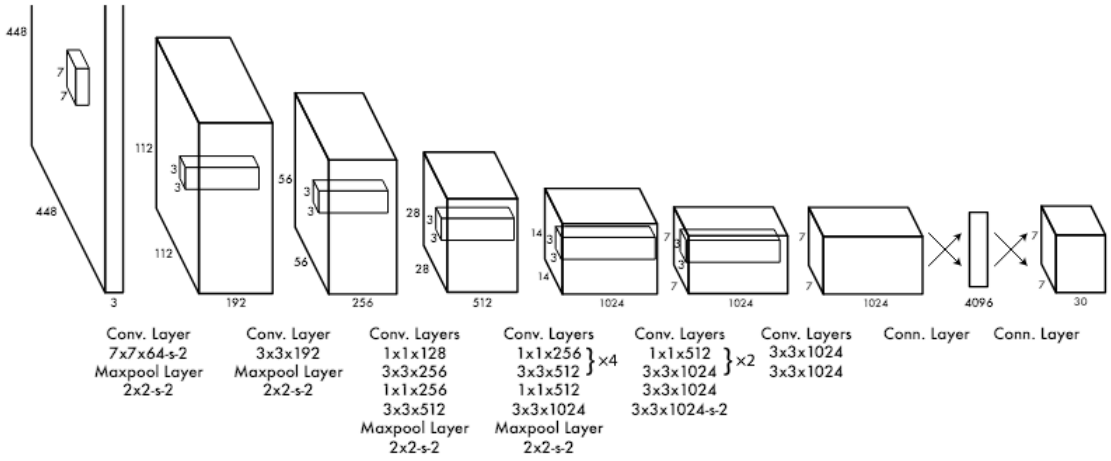
A arquitetura da rede neural convolucional que foi utilizada para desenvolver o trabalho de Redmon et al. (2016) implementou o modelo com apenas uma rede neural convolucional e fez uma avaliação utilizando o conjunto de dados de detecção de PASCAL VOC, onde as suas camadas de convoluções iniciais da rede tem como objetivo extrair recursos da imagem enquanto as camadas totalmente conectadas tem como objetivos prever as probabilidades e as coordenadas de saída da rede.

Redmon et al. (2016) fala em seu artigo que se inspirou no modelo da GoogleNet de classificação de imagens para criar sua arquitetura, a rede criada possui 24 camadas convolucionais que são seguidas por 2 camadas totalmente conectadas, com isso, ao invés de utilizar os módulos iniciais, iguais o da GoogleNet, foi utilizado camada de redução de 1x1 que são seguidas por camadas convolucionais de 3 x 3.

A Figura 5 representa a rede que foi implementada. Esta rede possui como dimensões da sua grade de entrada (grid) 7x7, a partir dessa dimensão de grade a rede irá seguir os demais passos apresentados na Figura 4 que apresenta uma grade genérica SxS e assim gerar as bounding boxes, confiança e o mapa de probabilidade de classe para que por fim realize a detecção final do objeto presente na imagem.

Essa primeira arquitetura da YOLO vem sendo aprimorada a cada versão nova disponibilizada dessa família de algoritmos, em geral apresentando melhorias em relação à cada versão anterior, sendo a YOLOv4 (SOLAWETZ, 2024) a última que utiliza como base a arquitetura original da Darknet. Em Solawetz (2020a) é possível ver uma comparação entre as versões YOLOv4 e YOLOv5. Já em Solawetz (2020b), podem ser vistos mais detalhes sobre a YOLOv5, a versão que foi a utilizada nesse trabalho.

Figura 5 – Arquitetura da rede neural convolucional utilizada no trabalho de Redmon et al. (2016).



Fonte: Extraída de Redmon et al. (2016)

2.3 Métricas para classificação

Em um processo de classificação é de suma importância medir o desempenho do modelo para que assim seja possível avaliar se o modelo esta evoluindo e consequentemente conseguindo realizar as classificações de forma correta e precisa e para isso são utilizados algumas métricas que auxiliam nesta análise e serão discutidos sobre elas com mais detalhes nas seções seguintes.

2.3.1 Matriz de Confusão

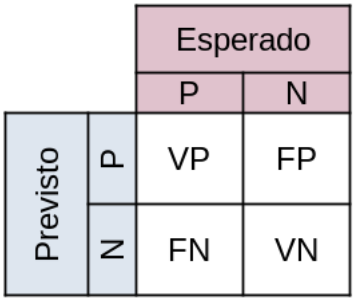
Matriz de confusão ou matriz de erro é uma tabela que possui como objetivo possibilitar a observação da performance do algoritmo de classificação, assim tendo os valores reais e os valores previsto, onde as linhas representam a classe do objeto que se espera e as colunas representam uma previsão para aquele objeto na imagem, ou vice-versa, podendo variar de acordo com quem estiver montando a matriz de confusão (MAGNO, 2023).

A matriz de confusão é ilustrada na Figura 6, e é definida como:

- Verdadeiro Positivo (VP): Previu como valor positivo e era valor positivo;
- Falso Negativo (FN): Previu como valor negativo e se esperava o valor positivo;
- Falso Positivo (FP): Previu como positivo mas se esperava valor negativo;
- Verdadeiro Negativo (VN): Previu como valor negativo e realmente era negativo.

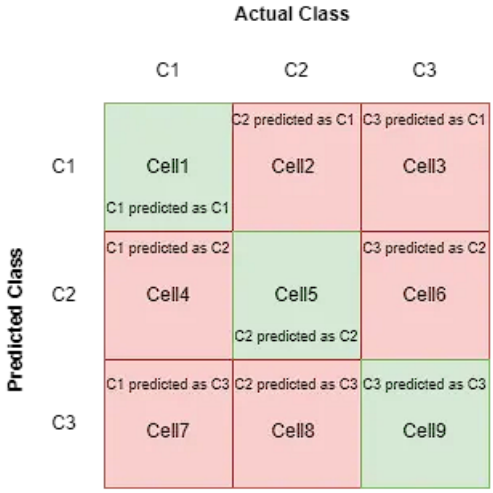
Já se tratando de classificação multiclasse Rajesh (2024) explica que diferente da classificação binária ela não é representada especificamente como positivas ou negativas, ou seja, não se calcula os valores de VP, VN, FP e FN de forma direta e ao invés disso realiza o cálculo por classe. Observando a Figura 7 como exemplo os cálculos são realizados da seguinte forma para C1:

Figura 6 – Matriz de confusão.



Fonte: Autoria própria.

Figura 7 – Matriz de confusão - MultiClasses.



Fonte: Retirada do trabalho de Rajesh (2024).

- VP é encontrada na Cell1, onde previu corretamente.
- VN é somado as cell: 5, 6, 8 e 9, onde foi previsto C2 ou C3 como C2 ou C3.
- FP é somado as cell: 2 e 3, onde foi previsto C2 ou C3 como C1.
- FN é somado as cell: 4 e 7, onde foi previsto C1 como C2 ou C3

2.3.2 Acurácia

A acurácia é uma métrica utilizada para definir o quão próximo um resultado experimental esta do valor real esperado, ou seja, quando maior for a acurácia melhor será o resultado (TCHILIAN, 2022; GOOGLE, 2024). Ela é medida como sendo

$$\text{acc} = \frac{\text{classificações corretas}}{\text{total de classificações}} = \frac{VP + VN}{VP + VN + FP + FN} \tag{2}$$

2.4 Métricas para detecção

Em um processo de detecção, é de suma importância medir o desempenho do modelo para verificar sua evolução e sua capacidade de realizar as detecções de forma

correta e precisa. Para isso, são utilizadas algumas métricas que auxiliam nessa análise, as quais serão discutidas em maior detalhe nas seções seguintes.

2.4.1 Precisão

Essa métrica indica o quão precisa a arquitetura é para detectar os objetos (ULTRALYTICS, 2023), indicando quantas das detecções foram corretas, ou seja, quantifica uma proporção dos verdadeiros positivos entre as demais previsões positivas e assim avaliando o quão capaz a arquitetura foi de evitar os falsos positivos (GOOGLE, 2024). Formalmente é escrita como:

$$\text{prec} = \frac{\text{positivos classificados corretamente}}{\text{tudo classificado como positivo}} = \frac{VP}{VP + FP} \tag{3}$$

2.4.2 *Recall*

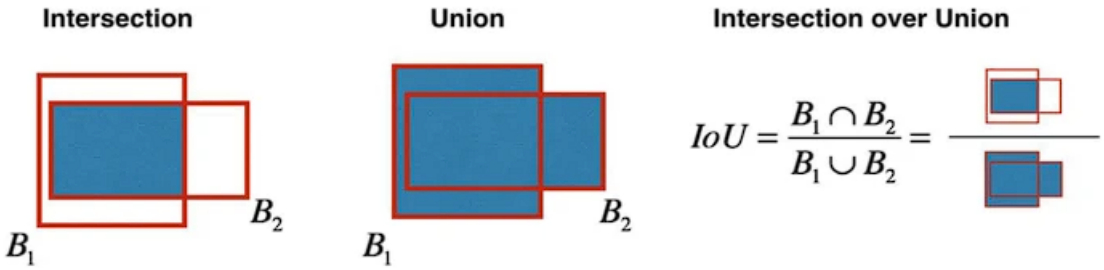
Essa métrica indica o quão capaz a arquitetura é de identificar as instâncias de objetos presentes nas imagens (ULTRALYTICS, 2023), ou seja, o *recall* calcula uma proporção entre verdadeiros positivos e todos os positivos reais e assim calculando o quão capaz o modelo foi de detectar todas as instâncias de uma classe. Formalmente:

$$\text{recall} = \frac{\text{corretamente classificados que são positivos}}{\text{todos os positivos}} = \frac{VP}{VP + FN} \tag{4}$$

2.4.3 *Intersection over Union - IoU*

Segundo Ultralytics (2023) interseção sobre união (*Intersection over Union - IoU*) é uma medida que estima a sobreposição entre a *bounding box* prevista e uma *bounding box* verdadeira, assim desempenhando uma papel crucial para avaliar a precisão da localização de objeto.

Figura 8 – Exemplo ilustrando a métrica de intersecção sobre união.



Fonte: Retirada do trabalho de Diwan, Anirudh e Tembhurne (2023).

2.4.4 Média da precisão média (mAP)

Ultralytics (2023) explica em seu site que precisão média média(mAP) é uma extensão da Precisão Média (AP) que calcula uma área sob a curva de recuperação média e assim obtendo um valor que engloba a precisão do modelo e seu desempenho de recuperação.

mAP_0.5 e mAP_0.95

Já a precisão média média(mAP), segundo a Ultralytics (2023) amplia este conceito da AP, o mAP calcula valores médios da AP em diversas classes de objetos. Assim a mAP_0.5 é a precisão média calculada dentro de um limite de interseção sobre a união (Intersection over Union - IoU) de 0.5, o que significa que é uma medida da precisão dos modelos que são considerados como detecções fáceis.

Já a mAP_0.5:0.95 é calculada em vários limites da IoU, contendo uma variação de 0.5 até 0.95 e assim apresentando um medida de precisão abrangente em relação ao desempenho do modelo em diversos níveis de dificuldade de detecção.

Durante o treinamento a YOLOv5 nos gera um arquivo csv contendo as métricas de treinamento e de validação de cada época que foi executado e trazendo o resultado de cada um, as métricas são referente à: Box_loss, Obj_loss, Cls_loss, precision, recall, mAP_0.5 e mAP_0.5:0.95.

2.5 Medidas de perda

As métricas de perda têm como objetivo mostrar o quanto o modelo errou, ou seja, indicar o quanto o modelo aprendeu ao longo das épocas. Com base nessas métricas, é possível avaliar a melhoria ou degradação do desempenho do modelo na detecção, tanto na localização do(s) objeto(s) adicionando a caixa delimitadora (*bounding box*) quanto na classificação dos objetos presentes na imagem. (REDMON et al., 2016).

2.5.1 Caixa delimitadora

A métrica da caixa delimitadora (*box loss*) é a perda de regressão da caixa delimitadora, ou seja, é a capacidade da YOLOv5 de detectar o centro dos objetos da imagem e o quão preciso será a montagem da *bounding box* que foram previstas estão cobrindo o objeto (REDMON et al., 2016).

Para o cálculo da box_loss a YOLOv5 utiliza o erro quadrático médio que é definido como uma métrica que tem como objetivo calcular a média da diferença entre o valor que foi previsto e o valor real (JUNIOR, 2021).

2.5.2 Presença do objeto

Essa métrica (*Obj loss*) é a confiança da presença do objeto, ou seja, é uma função de perda em relação a medida da probabilidade de um determinado objeto estar presente na região de interesse que foi proposta para o algoritmo (REDMON et al., 2016).

2.5.3 Classificação

Essa métrica (*Cls loss*) é uma função de perda de classificação, ou seja, é o calculo de desempenho do algoritmo em classificar os objetos com a classe correta (REDMON et al., 2016).

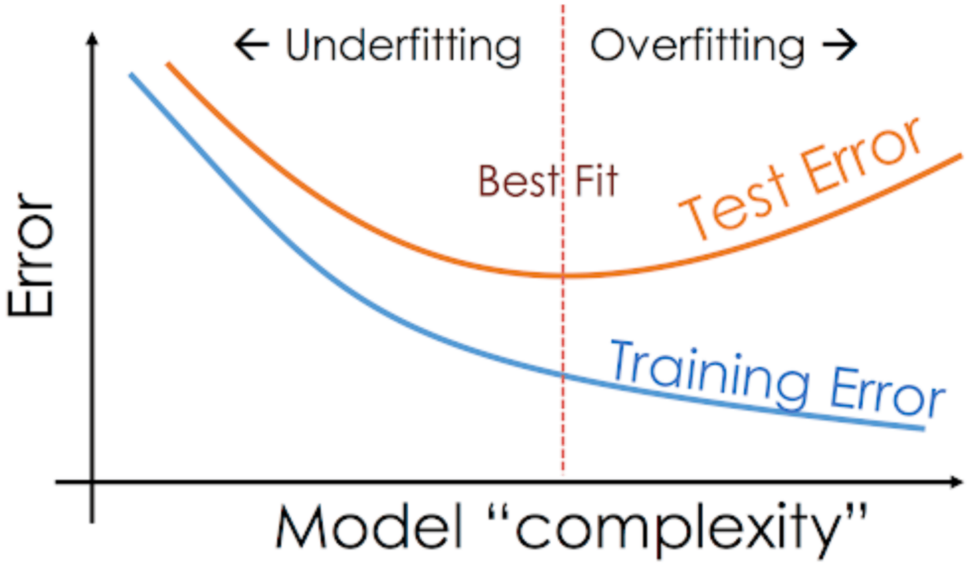
Para calcular esta perda a YOLOv5 utiliza o cálculo de entropia cruzada que pode ser empregada como uma métrica de erro quando as saídas de uma rede neural são concebidas de forma a representar hipóteses independentes (ACADEMY, 2022). Por exemplo, onde cada nó representa um conceito distinto e as ativações desses nós são interpretadas como a probabilidade ou confiança de cada uma das hipóteses ser verdadeira. Assim, com uma saída em forma de vetor representando uma distribuição de probabilidade, a medida de erro (entropia cruzada) indica a distância entre o que a rede neural acredita que a distribuição deve ser e o que deveria ser na realidade. Em outras palavras, expressa a diferença entre o que a rede neural prevê e o resultado esperado.

2.5.4 *Overfitting*

O *overfitting* ocorre quando um modelo apresenta complexidade excessiva, levando-o a memorizar detalhadamente os dados utilizados no treinamento. Como resultado, o desempenho durante o treinamento é elevado, porém, ao ser testado em novos conjuntos de dados durante a validação, seu desempenho se torna insatisfatório. Isso acontece porque o modelo se ajusta de forma demasiadamente específica aos dados de treinamento, em vez de aprender padrões generalizáveis, o que compromete sua capacidade de fazer previsões precisas em novos cenários. (SAXENA, 2023)

É possível analisar este comportamento na Figura 9, que é apresentado o gráfico contendo a evolução do modelo no treinamento (*Training Error*) e na validação (*Test Error*) sendo possível observar que ao chegar a um certo ponto o treinamento continua caindo enquanto o de validação volta a subir seu valor novamente, demostrando que o modelo volta a errar com o conjunto de dados diferente do utilizado no seu treinamento.

Figura 9 – Exemplo de gráfico demonstrando como identificar o *Overfitting*.



Fonte: Extraído de Saxena (2023).

3 TRABALHOS RELACIONADOS

3.1 Microorganismos ambientais

Ao contrário do que ocorre com bases de dados como a Wide Face, que foi utilizada para detecção de rostos e descrita em Liu et al. (2022), imagens de microrganismo ambientais são bem mais difíceis de serem obtidas, tornando-se difícil encontrar uma base com esses dados que possua uma grande variabilidade que seja pública.

3.1.1 Abordagens tradicionais para classificação

Ginoris et al. (2007) em seu trabalho analisam diferentes espécies de protozoários e metazoários, em um total de 22 espécies diferentes. Utilizam vinte e oito características (*features*) calculados manualmente para as imagens de cada espécie (ex: área, perímetro). Para efetuar a classificação usam duas estratégias, uma usando redes neurais peceptron multicamadas (*Multilayer percepctron - MLP*) e outra usando árvores de decisão. Não analisam os resultados usando *precision* e *recall*, mas uma medida de porcentagem de reconhecimento, obtendo resultados acima de 90%. Porém, sua base de dados não é disponibilizada publicamente. Algumas das espécies usadas são encontradas em outros datasets públicos, como o Li et al. (2021) e Zhao et al. (2022).

Zhao et al. (2022) também utilizam de abordagens tradicionais onde utilizaram da base de dados EMDS-6 para avaliar alguns modelos tradicionais de classificação de aprendizagem de maquina porem para classificar feições geográficas, onde traz diferentes modelos do Support Vector Machine (SVM) como SVM:Linear, SVM:Polinomial, SVM: RBF, entre outros e trazendo como resultados para estes modelos a precisão de: 51,67, 27,86 e 28,81, respectivamente, além disso utilizando o modelo RF apresentando maior precisão sendo: 98,33.

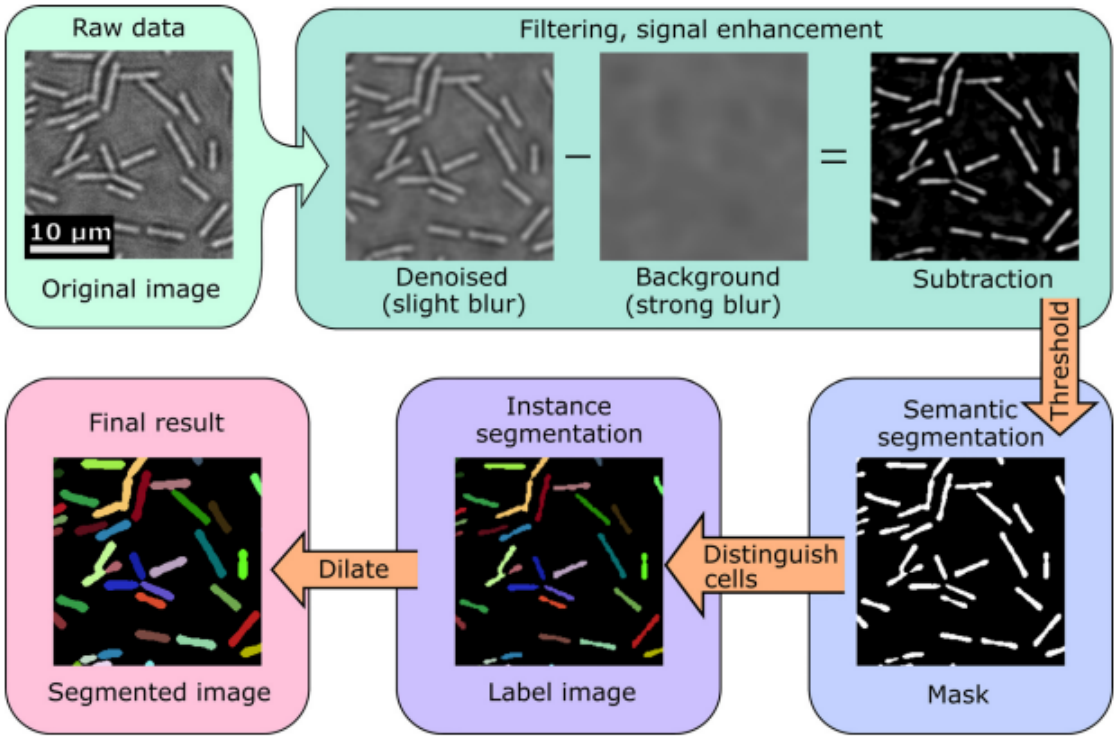
3.1.2 Abordagens tradicionais para detecção

Se tratando de abordagens tradicionais Jeckel e Drescher (2021) citam em seu trabalho que geralmente as imagens são processadas com a aplicação de vários filtros diferentes um do outro para que se possa obter: uma remoção do fundo, um aprimoramento do seu nível para o ruído e uma ênfase de estruturas que se deseja, entretanto, os tamanhos de seu *kernel* acaba sendo definido de forma automática levando em consideração os tamanhos das células (ou das colunas) que serão analisadas, já quanto a escolha e a ordem dos filtros para esse processo são levadas pela experiência da pessoa que irá desenvolver o código. Deve-se lembrar também que caso a qualidade ou a estrutura das imagens sejam alteradas, os paramentos e os filtros escolhidos muitas das vezes pode ser que sejam necessários à sua modificação para se adaptar as alterações. Para a obtenção de uma

segmentação semântica que é capaz de separar os objetos de interesse do fundo, é aplicado então um limite as imagens que foram filtrados previamente e assim tendo como resultado uma imagem binária que se denomina "máscara", que possui então apenas valores de 1 e 0, tanto para o plano quanto para o plano de fundo. E a partir disso essas operações morfológicas são capazes de serem aplicadas à uma imagem binária para que assim a imagem máscara possa representar uma segmentação semântica mais precisa e eficaz. Essas operações morfológicas realizadas podem ser, por exemplo, uma remoção de pequenos objetos, estreitamento de objetos ou um preenchimento de objetos com orifícios.

Na Figura 10 é mostrado um exemplo do processamento típico feito em uma imagem para obter uma segmentação adequada nesse contexto.

Figura 10 – Fluxo de trabalho de segmentação típico em análise de imagem tradicional.



Fonte: Extraído de Jeckel e Drescher (2021)

3.1.3 Aprendizado profundo para classificação

Em Zhao et al. (2022) são apresentados resultados de classificação de diferentes modelos de aprendizagem profunda como por exemplo ResNet34, ResNet50, GoogleNet, entre outras, onde é apresentado diferentes métricas para avaliar o desempenho de diferentes modelos de classificadores com o EMDS-6.

Analisando duas das diferentes métricas apresentadas, a F1-Score a Acurácia, Zhao et al. (2022) apresenta como melhor resultado para F1-Score e Acurácia o modelo Xception que tem como resultado, respectivamente, 42.40% e 44.29%.

Zhao et al. (2022) utilizaram em seu artigo para testar a possibilidade de utilização do dataset EMDS-6 para a detecção de objetos as arquiteturas de dois estágios Faster RCNN e a Mask RCNN. Segundo eles as RCNN mais rápidas entregam excelentes resultados de desempenho em várias áreas de detecção de objetos, já o Mask RCNN é melhorado na estrutura original do Faster RCNN, com isso, possuindo um esqueleto melhor e utilizando o algoritmo AlignPooling o Mask RCNN demonstrou resultados melhores de detecção de objetos em relação ao Faster RCNN.

Zhao et al. (2022) trás em seu artigo uma analise das arquiteturas Faster RCNN e a Mask RCNN analisando os resultadas AP e mAP de detecção dos objetos. É apresentado no artigo uma tabela com os valores de AP e mAP que mostra que as arquiteturas demonstraram terem resultados significativamente diferentes com base em seus valores de AP. Com isso, analisando a tabela é possível observar que o Faster RCNN apresentou melhores resultados ao classificar a classe Actinophrys, já o Mask RCNN apresentou melhores resultados ao classificar a classe Arcella. Assim analisando o resultado de mAP Zhao et al. (2022) concluíram que o Faster RCNN acabou sendo melhor que o Mask RCNN na detecção de objetos. Por consequência demonstrando que o dataset EMDS versão 6 consegue ser bastante efetivo quando aplicado na detecção de objetos

Uma dificuldade comum em trabalhos que classificam ou detectam microrganismos ambientais é que as imagens obtidas no processo, em geral, são restritas ao grupo de pesquisa que as gerou, são bases de dados privadas. Na Tabela 1, os trabalhos descritos nessa revisão bibliográfica são descritos quanto ao tipo de acesso à base: se pública ou se privada.

Tabela 1 – Algumas bases de dados de microrganismos ambientais e sua respectiva disponibilidade.

Referência	**Nome da base**	**Tipo de acesso**
Ginoris et al. (2007)	-	Privado
Li et al. (2021)	EMDS5	Público
Zhao et al. (2022)	EMDS6	Público

Fonte: Autoria própria.

Na Tabela 2 são detalhadas as espécies e quantitativos de imagens das bases públicas descritas na Tabela 1.

Tabela 2 – Bases de dados públicas EMDS-5 e EMDS-6. Descritivo das classes disponíveis e número de imagens por classe.

Classe da imagem	EMDS5	EMDS6
Actinophrys	20	40
Arcella	20	40
Aspidisca	20	40
Colpoda	20	40
Epistylis	20	40
Paramecium	20	40
Vorticella	20	40
Stentor	20	40
Spirostomum	20	40
Stylonychia	20	40
Codosiga	20	40
Euglypha	20	40
Rotifera	20	40
Keratella	20	40
Synchaeta	20	40
Noctiluca	20	40
Ceratium	20	40
Gymnodinium	20	40
Gonyaulax	20	40
Euglena	20	40
Phacus	20	40

Fonte: Adaptada de Li et al. (2021) e Zhao et al. (2022).

4 MATERIAIS E MÉTODOS

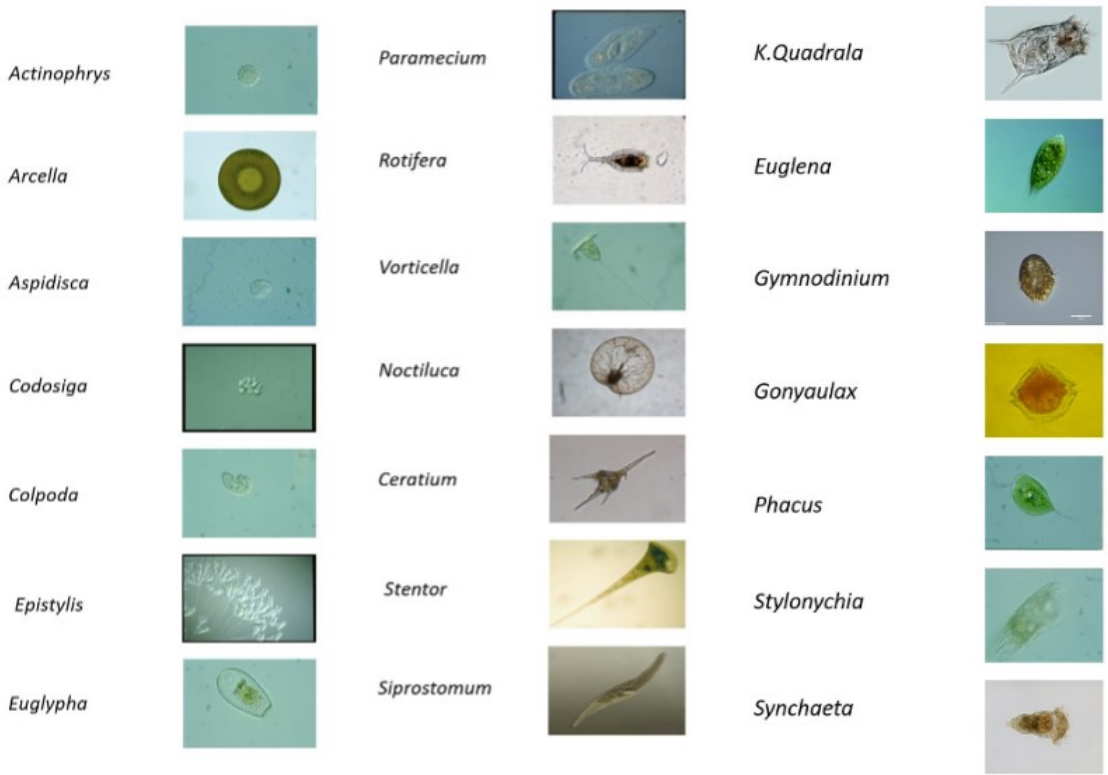
Neste capítulo são descritos: a base de dados que foi utilizada no desenvolvimento do trabalho, junto com o pré-processamento e os métodos que foram utilizados pra treinamento e detecção.

4.1 Base de dados

A base de dados utilizada nesse trabalho é a quinta versão do *Environmental Microorganism Data Set* (EMDSv5) (LI et al., 2021). É um dataset público que contém imagens microscópicas de microrganismos ambientais, possuindo um total de 21 diferentes espécies (classes), contando com 20 imagens de cada, totalizando 420 imagens, descritas na Tabela 2 e demonstrada na Figura 11.

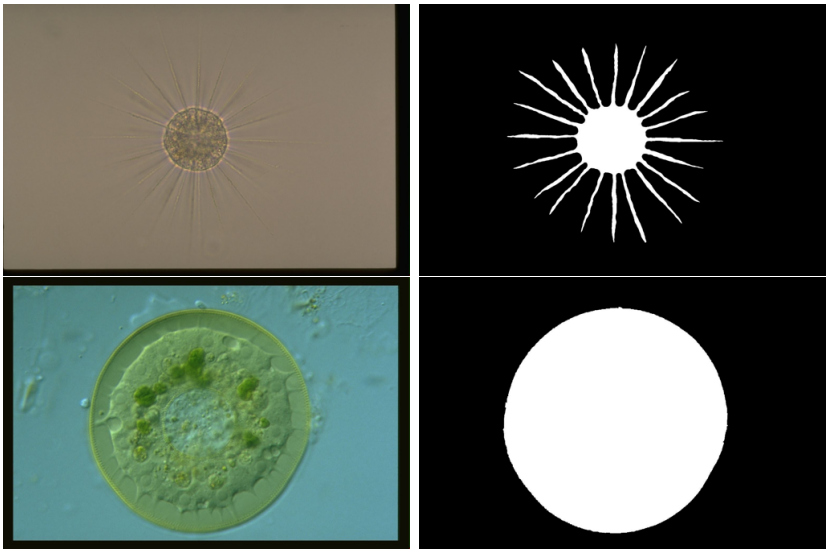
Além disso, essa base de dados também disponibiliza uma segmentação das espécies ideal (*Ground Truth - GT*), feita por especialistas. Essa segmentação é dividida em outros dois conjuntos de imagens GT: para objetos únicos (EMDS5-GTS) e para múltiplos objetos (EMDS5-GTM). Esses dois conjuntos são formados por imagens em tons de cinza, indicando a segmentação da espécie na imagem como ilustrado na Figura 12.

Figura 11 – Classes da base de dados EMDS-5.



Fonte: Adaptada de Li et al. (2021)

Figura 12 – EMDS5 original e sua correspondente GTS.



Fonte: Li et al. (2021)

4.2 YOLOv5

Jocher et al. (2022) em seu trabalho mostra que esta versão 5 da YOLO é mais rápida e precisa, tendo sido desenvolvida com o objetivo principal de ser mais simples de treinar, validar e implantar.

4.2.1 Estrutura da YOLO

Para a utilização da YOLO é exigido que tenha uma estrutura montada especificamente dentro de suas diretrizes para que possa ser utilizada de forma eficaz.

Estrutura de diretórios

A estrutura original ao baixar o dataset EMDSv5 de (LI et al., 2021) é disponibilizado as pastas contendo todas a imagens originais em uma única pasta e assim também ocorre com as demais GTS e GTM. Com isso para a Versão 1 (V1) foi criado duas pastas: uma chamada *images* e a outra *labels*.

Na *images* foi criado outras três pastas: train, val e test, onde nelas foram separadas 16 imagens de cada classe para a train, 2 imagens de cada classe para a val e 2 imagens de cada classe para a test.

Já a pasta *labels* também foi criado as três pastas assim como na *images*, porém ela foi utilizada para armazenar os arquivos txt correspondente de cada imagem e salvas de acordo com cada pasta em que a imagem estava salva em *images*. Os arquivos txt foram salvos com o nome de acordo com sua imagem correspondente.

A versão 2 (V2), que é a versão contendo o dataset original juntamente com as imagens do data augmentation segue o mesmo modelo da V1, porém nesta versão as

images geradas pelo data augmentation possuem uma nomenclatura diferente, onde foi adicionado 'da' ao final do nome das imagens e consequentemente as *labels* que foram geradas a partir destas imagens que possuem o 'da' também foram geradas com o 'da' no final do nome de cada arquivo respeitando a nomenclatura da imagem.

Labels

Ultralytics (2024) mostra em seu site que as *labels* são arquivos .txt no qual terá um arquivo para para imagem. No arquivo estará presente os dados de cada objeto da imagem que irá gerar um arquivo '.txt', onde cada linha do arquivo corresponde a um objeto na imagem.

As *labels* contém como informação do objeto da imagem a seguinte sequência de dados: classe da imagem de acordo com o arquivo .yaml (que começam em 0), os centros x e y, largura (width) e altura (heigth). Porém esses dados devem estar normalizados, ou seja, seus valores devem estar em um intervalo de 0 a 1, para isso basta dividir o centro de x e a largura (width) do objeto pela largura da imagem e dividir o centro y e a altura (height) do objeto pela altura da imagem.

Arquivo .yaml V1

Para o treinamento da YOLO com um conjunto de dados personalizados é preciso montar um arquivo .yaml, onde, nele estará o caminho do dataset (path) e os caminhos das pastas de treinamento (train), teste (test) e validação (val), neste arquivo também irá conter o número de classes que a base de dados possui (nc) e o nome de todas as classes sendo numeradas a partir de 0 ate a ultima classe. Segue modelo .yaml criado para este trabalho:

Algoritmo 1: Exemplo de uso de um arquivo de treinamento para a YoloV5.

```
path: ../datasets/EMDS5_v1
train: images/train
val: images/val
test: images/test
nc: 21
names:
  0: Actinophrys
  1: Arcella
  2: Aspidisca
  3: Codosiga
  4: Colpoda
  5: Epistylis
  6: Euglypha
  7: Parameciumi
  8: Rotifera
  9: Vorticella
  10: Noctiluca
  11: Ceratium
  12: Stentor
  13: Siprostomum
  14: Keratella Quadrala
  15: Euglena
  16: Gymnodinium
  17: Gonyaulax
  18: Phacus
  19: Stylongchia
  20: Synchaeta
```

Fonte: Autor.

4.2.2 Montagem do bounding box

Para a criação da bounding box não foi utilizado outras aplicações, ou seja, foi utilizado a própria imagem GTS para sua montagem. Como as imagens GTS são em tons de cinza foi necessário inicialmente transforma-lá em uma imagens preto e branco fazendo com que seus pixels fossem ou brancos (255) ou preto (0), para isso utilizamos o Algoritmo 2.

Algoritmo 2: Binarização da imagem.

```
for lin in range(height):
    for col in range(width):
        if img[lin, col] > 126:
            matrix[lin, col] = 255
        else:
            matrix[lin, col] = 0
```

Fonte: Autor.

Após transformar a imagem em tons de cinza para uma em preto e branco foi obtido as suas dimensões utilizando:

```
height = img.shape[0]
width = img.shape[1]
```

para encontrar a altura e a largura.

Com a imagem preto e branco e suas dimensões encontradas, foi utilizado um laço de repetição (for) para que percorresse toda a imagem e assim toda vez que o pixel percorrido fosse braco eram feito um comparativo com as variáveis X máximo e mínimo e Y máximo e mínimo, onde os valores máximos foram iniciados em 0 e os mínimos com um valor muito grande. Já com estes dados encontrados e armazenados, foi preciso apenas cria uma imagem a partir da matriz binária, inicializa um objeto para desenhar na imagem e desenhar a *bounding box* na imagem.

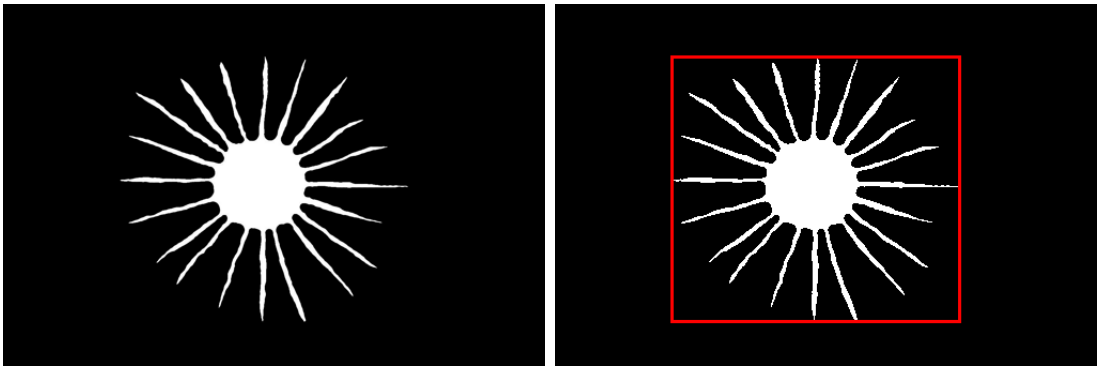
Tendo os dados cartesiano X máximo e mínimo e Y máximo e mínimo é possível calcular a altura, largura e o centro X e Y da bounding box para que possa ser informado nas labels. Para encontrar a largura e altura da bounding box foi feito uma subtração entre o X máximo e mínimo para a largura (w) e entre o Y máximo e mínimo para altura (h). Já para calcular o centro da bounding box foi utilizado o resultado de w_caixa e h_caixa, no qual para encontrar o centro X foi realizado uma soma entre o X mínimo com a metade de w_caixa e para o centro Y entre o Y mínimo com a metade de h_caixa. O Algoritmo 3 detalha em código esse procedimento, e na Figura 13 é mostrado um exemplo do resultado ao final do processo.

Algoritmo 3: Cálculo do bounding box.

```
Xmax = 0
Xmin = sys.maxsize
Ymax = 0
Ymin = sys.maxsize

for lin in range(height):
    for col in range(width):
        if matrix[lin, col] == 255:
            if lin < Ymin: Ymin = lin
            elif lin > Ymax: Ymax = lin
            if col < Xmin: Xmin = col
            elif col > Xmax: Xmax = col

data = Image.fromarray(matrix)
ret = ImageDraw.Draw(data)
ret.rectangle([(Xmin, Ymin), (Xmax, Ymax)], fill=None, outline="
    red")

w_caixa = Xmax - Xmin
h_caixa = Ymax - Ymin

x_centro = Xmin + (w_caixa/2)
y_centro = Ymin + (h_caixa/2)
```

Fonte: Autoria própria.

Figura 13 – EMDS5 GTS original e com a *bounding box*.



Fonte: Autoria própria.

4.2.3 Normalização dos dados da *bounding box*

Com os dados da bounding box obtidos é preciso realizar a normalização para atender as exigências da Yolov5 que para adicionar os dados de w_caixa (largura da bounding box), h_caixa (altura da bounding box), x_centro (centro x da bounding box), y_centro (centro y da bounding box) nas labels deve estar no intervalo de 0 e 1. Para isso foi realizado uma divisão de cada dado pelo seu dado correspondente da imagem toda e assim utilizando o Algoritmo 4 foi feita a normalização.

Algoritmo 4: Normalização do *bounding box*.

```
1 w_caixa = w_caixa / width
2 h_caixa = h_caixa / height
3 x_centro = x_centro / width
4 y_centro = y_centro / height
```

Fonte: Autoria própria.

4.2.4 Rodando a Yolo

Para rodar a YOLO vasta realizar o clone do seu diretório padrão utilizando o comando no terminal dentro de um pasta que não possui caracteres especiais

```
git clone https://github.com/ultralytics/yolov5
```

Em seguida acessar a pasta que será criada com o comando

```
cd yolov5
```

E em seguida fazer a instalação dos requerimentos da arquitetura utilizando

```
pip install -r requirements.txt
```

Após concluir a instalação dos requerimentos, selecione um modelo da YOLO que deseja utilizar e treine utilizando o seguinte comando no terminal:

```
python train.py --img 640 --batch 16 --epochs 3 --data coco128.yaml
--weights yolov5s.pt
```

Este modelo utiliza o dataset COCO128 especificando o conjunto de dados, o tamanho do lote, tamanho da imagem. E é possível visualizar os resultados ao concluir na pasta train que fica dentro da pasta runs, nela terá os resultados de treinamento separadamente por pastas.

4.2.5 Rodando a YOLO com o dataset EMDS5

Para rodar a YOLO utilizando o dataset escolhido, o EMDS5, foi utilizado os mesmos passos e comandos dito anteriormente, porém ao invés de utilizar os dados padrão, foi utilizado os arquivos de texto ".yaml" que foi definido os dados de cada imagem para realizar a execução da arquitetura. Então para isso foi adicionada a pasta do dataset que continha as imagens e as labels (arquivos de texto referente a cada imagem) na pasta dataset no mesmo nível do dataset padrão baixado após executar o comando de clone e foi adicionado o arquivo de texto ".yaml" na pasta "data"dentro da pasta "yolov5"que foi criada no mesmo nível da pasta do dataset.

Após concluir esta organização foi realizado a execução do comando para o treino, porém desta vez passando o ".yaml" que foi criado com a localização do dataset e demais dados.

```
python train.py --img 640 --batch 16 --epochs 300 --data emds5.yaml
--weights yolov5s.pt
```

4.3 Data Augmentation

Além da utilização do dataset EMDS5 original foram efetuadas operações para *data augmentation*. Esta estratégia é um meio de aumentar de forma artificial o conjunto de dados, criando então copias modificadas do conjunto original EMDS5. Foram utilizados 10 operações de *data augmentation*, no qual alguns fazem mudanças apenas nas imagens sem mudá-las geograficamente e outros que realizam mudanças maiores.

4.3.1 Data Augmentation TensorFlow

TensorFlow (2022) disponibiliza alguns modelos de data augmentation que podem ser utilizados para aumento de dados, neste trabalho foram utilizados cinco modelos dos vários que são disponibilizados, estes cinco modelos alteram apenas a qualidade da imagem assim não sendo necessário fazer a transformações em suas imagens GTS correspondentes, segue os modelos utilizados:

- (**DA01**): Ajusta o brilho das imagens de forma aleatória.
  Foi utilizada a função *tf.image.stateless_random_brightness(img, 0.2, (1, 2))*
- (**DA02**): Ajusta o contraste das imagens.
  Foi utilizada a função *tf.image.stateless_random_contrast(img, 0.2, 0.5, (1, 2))*
- (**DA03**): Ajusta a matriz das imagens RGB de forma determinística.
  Foi utilizada a função *tf.image.stateless_random_hue(img, 0.2, (1,2))*
- (**DA04**): Introduz ruídos a imagem e assim alterando a qualidade da imagem.
  Foi utilizada a função *tf.image.stateless_random_jpeg_quality(img, 30, 50, (1,2))*
- (**DA05**): Altera a saturação da imagem.

Foi utilizada a função *tf.image.stateless_random_saturation(img, 0.1, 2.0, (1,2))*

4.3.2 Data Augmentation do python

A própria biblioteca do python disponibiliza alguns métodos para tratar uma imagem e neste trabalho foram utilizado 3 modelos sendo 1 deles utilizados mais de uma vez totalizando assim 5 métodos utilizados, sendo eles:

- (**DA06**): Rotaciona a imagem 90 graus
  Foi utilizada a função *.rotate(90)*
- (**DA07**): rotaciona a imagem 180 graus
  Foi utilizada a função *.rotate(180)*
- (**DA08**): rotaciona a imagem 270 graus
  Foi utilizada a função *.rotate(270)*
- (**DA09**): Vira a imagem horizontalmente (da esquerda para direita)
  Foi utilizada a função *.transpose(im.FLIP_LEFT_RIGHT)*
- (**DA10**): Vira a imagem verticalmente (de cima para baixo)
  Foi utilizada a função *.transpose(im.FLIP_TOP_BOTTOM)*

Estes métodos utilizados mexem na imagem de forma geográfica, ou seja alterem a localização do objeto na imagem e até mesmo o formato da própria imagem, sendo assim necessário passar também estas transformações nas imagens GTS, pois como houve mudanças geométricas é necessário calcular o bounding box destas imagens GTS que foram geradas por estas transformações.

4.3.3 Exemplos com aumento de dados

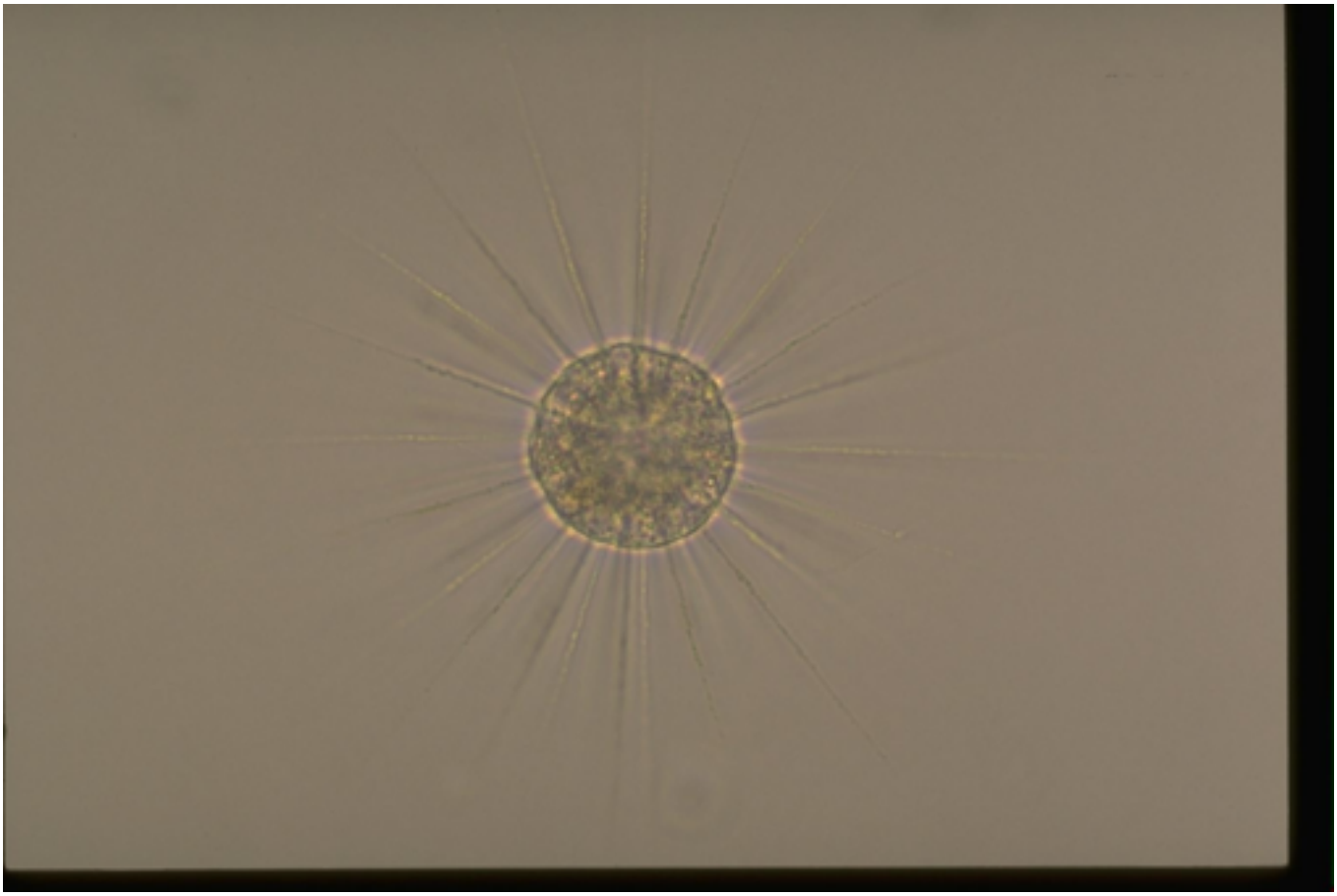
As Figuras 14a e 14b representam imagens originais das classes Keratella quadrala e Actinophrys, respectivamente. Elas são utilizadas aqui para demonstrar o efeito do aumento de dados.

As Figuras 15 e 16 representam as transformações de *data augmentation* que foram realizadas na imagens (DA01, . . . , DA10), onde é possível observar de forma mais clara as transformações realizadas.

As Figuras 17 e 18 também representam as 10 transformações de *data augmentation* utilizados nas imagens. Para essa classe Actinophrys, que é um modelo simétrico para as transformações utilizadas, não houve muita mudança na imagem e assim tendo dados praticamente repetidos, apresentando diferenças apenas no brilho das imagens. Isso pode ser uma explicação do problema de *overfitting* identificado durante o treinamento, que será discutido no Capítulo 5.

Figura 14 – EMDS5 original.

(a) Classe Keratella Quadrala.



(b) Classe Actinophrys

Fonte: Li et al. (2021).

4.4 EMDS5 Versão 2 (V2)

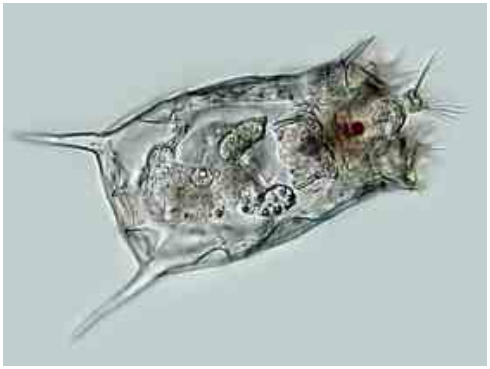
A versão 2 criada a partir do EMDS5 original é composta pela imagens do dataset original juntamente com as imagens geradas com o data augmentation totalizando assim 4620 imagens, onde, possui 3696 imagens na pasta train, 462 na pasta val e 462 nas pasta test. Com isso cada classe contendo 176 imagens ao todo.

Figura 15 – Imagens geradas pelo data augmentation - Tensor flow. As operações ilustradas aqui são, respectivamente: a) DA01, b) DA02, c) DA03, d) DA04, e) DA05.

(a)

(b)

(c)

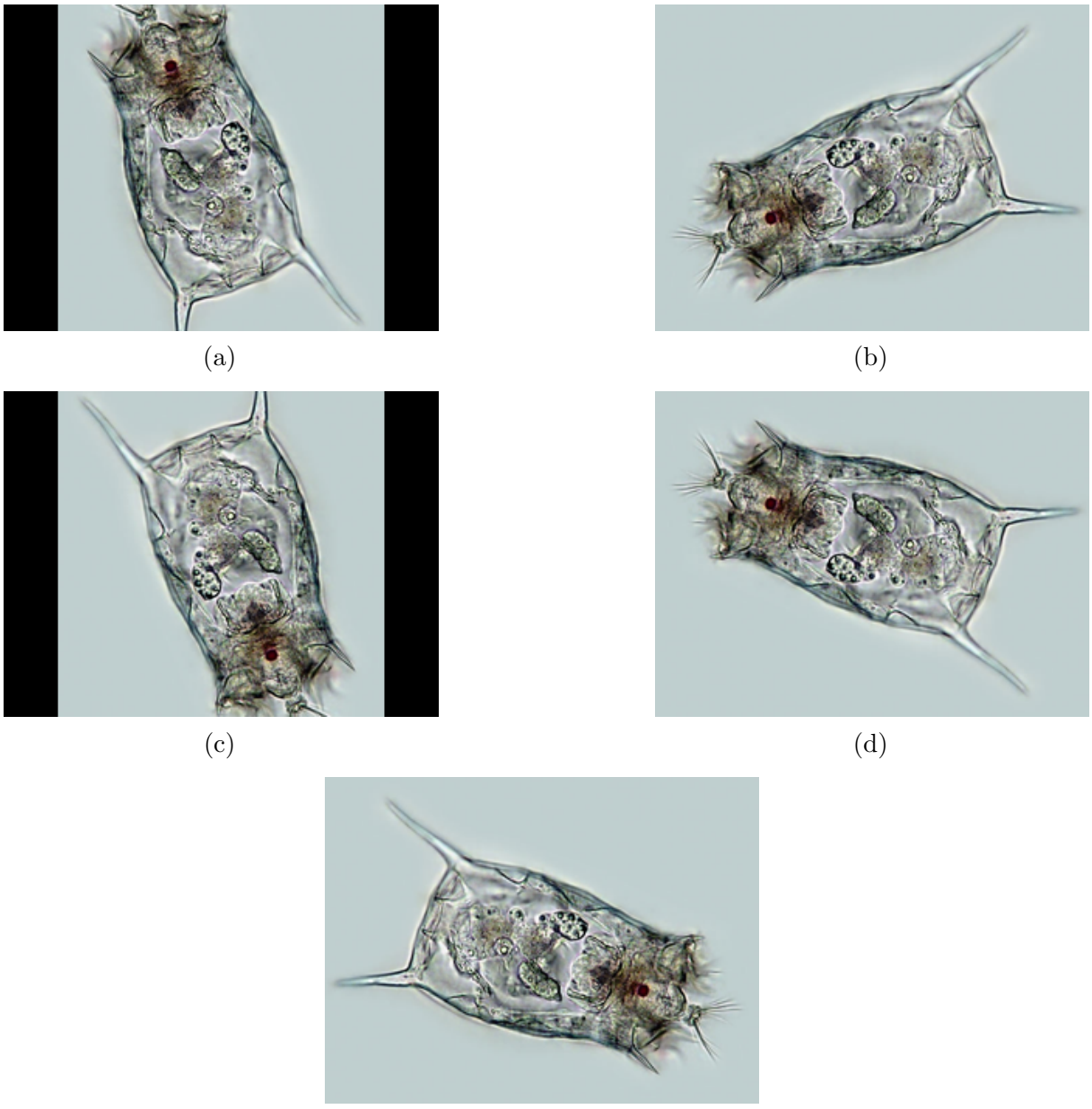


(d)

(e)

Fonte: Autoria própria.

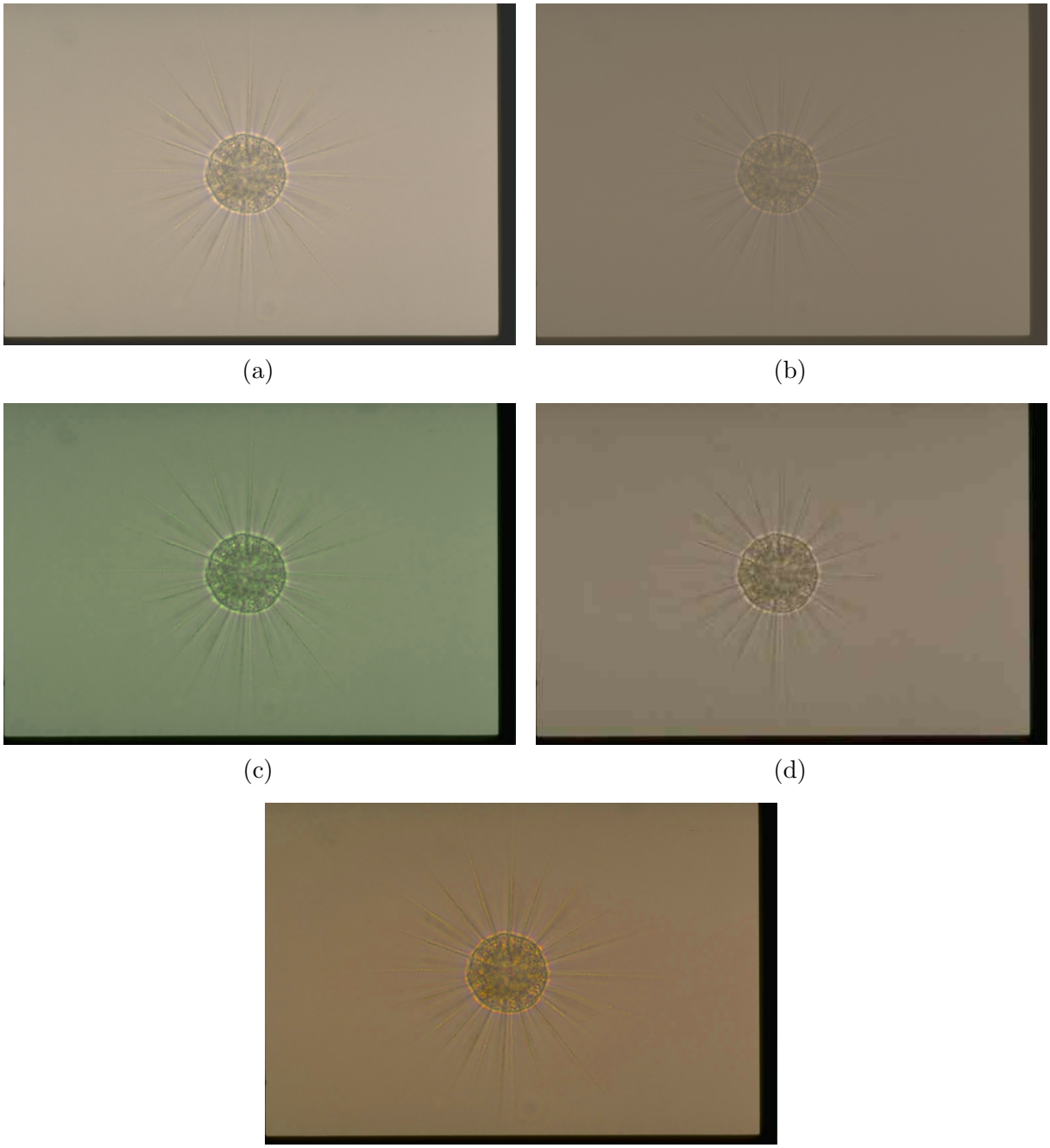
Figura 16 – Imagens geradas pelo data augmentation. As operações ilustradas aqui são, respectivamente: a) DA06, b) DA07, c) DA08, d)DA09, e)DA10.



(a) (b) (c) (d) (e)

Fonte: Autoria própria.

Figura 17 – Imagens geradas pelo data augmentation - Tensor flow. As operações ilustradas aqui são, respectivamente: a) DA01, b) DA02, c) DA03, d) DA04, e) DA05.



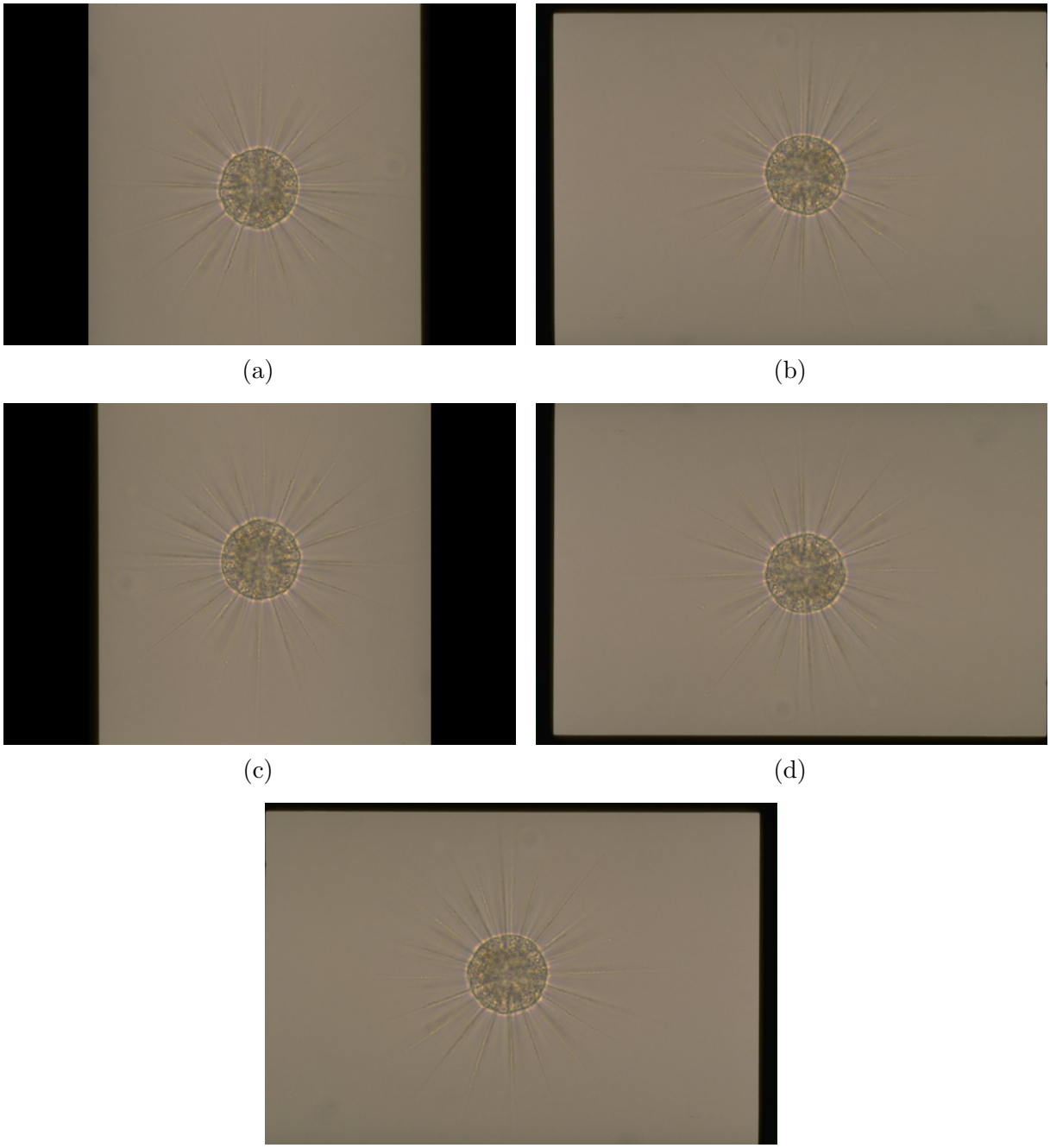
(a) (b)

(c) (d)

(e)

Fonte: Autoria própria.

Figura 18 – Imagens geradas pelo data augmentation. As operações ilustradas aqui são, respectivamente: a) DA06, b) DA07, c) DA08, d)DA09, e)DA10.



(a) (b)

(c) (d)

(e)

Fonte: Autoria própria.

5 RESULTADOS E DISCUSSÕES

Os resultados que foram obtidos utilizando as diferentes versões da YoloV5 treinados com o dataset EMDS5, tanto na sua versão original (V1) quanto com a versão com *data augmentation* (V2). Houve uma variação de desempenho em cada versão quando observado a partir das métricas obtidas após as 300 épocas, parâmetro utilizado como critério de parada do treinamento. Nas próximas seções será apresentado estes resultados através de gráficos e tabelas, demonstrando assim um comparativo e o desempenho dos diferentes tamanhos da YoloV5 com ambos os *datasets*. Para ter esses resultados de treinamento, validação e teste, foi realizado a seguinte separação das imagens: 80% para treino, 10% para validação e 10% para testes, isso tanto no EMDS5v1 quanto no EMDS5v2.

A máquina utilizada para os experimentos nesse trabalho possui configuração base: Intel Core i7-9700T, 16GB DDR4-SDRAM 2666 MHz e 256 GB SSD.

5.1 Resultados do YoloV5

Utilizando o EMDS5 na versão 1 não foi possível rodar a YOLOv5 no seu tamanho *XLarge* (YOLOv5x) e utilizando o EMDS5 na versão 2 não possível rodar nos tamanhos: *Large* (YOLOv5L) e *XLarge* (YOLOv5x) devido a maquina não suportar o peso destes tamanhos de rede.

Tabela 3 – Melhores resultados das métricas de cada modelo (em negrito) no treinamento após as 300 épocas com EMDS5.

Versão	**Modelo**	**Par.**($\times 10^6$)	**mAP_0.5**	**mAP_0.5:0.95**	**Precisão**	**Recall**
V1	YoloV5n	1.8	0.93205	0.70456	0.90377	0.90476
V1	YoloV5s	7.1	0.91634	0.70404	0.93292	**0.92235**
V1	YoloV5m	20.9	0.89685	0.73514	**0.94680**	0.86585
V1	YoloV5l	46.2	**0.94370**	**0.74247**	0.94582	0.90476
V1	YoloV5x	86.4	-	-	-	-
V2	YoloV5n	1.8	**0.84607**	0.62994	0.83918	0.85239
V2	YoloV5s	7.1	0.83933	0.66088	0.86911	**0.86937**
V2	YoloV5m	20.9	0.84366	**0.67052**	**0.87192**	0.86055
V2	YoloV5l	46.2	-	-	-	-
V2	YoloV5x	86.4	-	-	-	-

Fonte: Autoria própria.

Além das métricas apresentadas na Tabela 3, onde foram apresentados os melhores valores referentes a cada métrica, a YoloV5 também nos retorna as métricas de perdas, ou seja, o *loss*, tanto de treinamento quanto de validação do modelo e assim sendo possível analisar quanto aos erros e perdas, com esses dados foram criados gráficos para avaliar essa perca ao longo das épocas e para avaliar as métricas apresentadas.

O total de épocas de treinamento foi definido inicialmente como 300 épocas. Porém, nem todos os experimentos concluíram todas as 300 épocas. Isso acontece quando a função de perda não varia mais durante uma determinada quantidade de iterações.

Na versão 1 todos os experimentos concluíram as 300 épocas. Já na versão 2 houve uma variação um pouco maior, onde a quantidade de épocas foi: no modelo YoloV5n foi de 293; no YoloV5s foi de 263; e no YoloV5m foi de 200.

A seguir nas Tabelas 4, 5 e 6 são apresentados a média e a variância das ultimas 10 épocas de cada métrica e seus respectivos modelos da YoloV5 tanto na versão 1 da EMDS5 quanto na versão 2 da EMDS5, mostrando a estabilização dos valores.

Tabela 4 – Média e Variância (entre parênteses) das métricas de cada modelo no treinamento nas 10 ultimas épocas.

Versão	**Modelo**	**mAP_0.5**	**mAP_0.5:0.95**	**Precisão**	**Recall**
V1	YoloV5n	0,897 ($2{,}2 \times 10^{-5}$)	0,679 ($2{,}7 \times 10^{-5}$)	0,797 ($1{,}1 \times 10^{-3}$)	0,852 ($1{,}1 \times 10^{-3}$)
V1	YoloV5s	0,893 ($8{,}9 \times 10^{-6}$)	0,685 ($2{,}0 \times 10^{-5}$)	0,871 ($3{,}0 \times 10^{-4}$)	0,800 ($1{,}1 \times 10^{-4}$)
V1	YoloV5m	0,865 ($2{,}7 \times 10^{-5}$)	0,725 ($2{,}2 \times 10^{-5}$)	0,860 ($5{,}4 \times 10^{-4}$)	0,807 ($2{,}0 \times 10^{-5}$)
V1	YoloV5l	0,897 ($1{,}8 \times 10^{-4}$)	0,728 ($8{,}3 \times 10^{-5}$)	0,867 ($2{,}6 \times 10^{-3}$)	0,796 ($5{,}3 \times 10^{-4}$)
V2	YoloV5n	0,801 ($2{,}1 \times 10^{-7}$)	0,609 ($6{,}8 \times 10^{-7}$)	0,820 ($4{,}7 \times 10^{-4}$)	0,805 ($2{,}8 \times 10^{-5}$)
V2	YoloV5s	0,824 ($4{,}7 \times 10^{-7}$)	0,655 ($1{,}6 \times 10^{-6}$)	0,802 ($4{,}0 \times 10^{-6}$)	0,860 ($6{,}2 \times 10^{-8}$)
V2	YoloV5m	0,821 ($1{,}5 \times 10^{-5}$)	0,655 ($6{,}4 \times 10^{-6}$)	0,834 ($3{,}6 \times 10^{-4}$)	0,820 ($1{,}6 \times 10^{-4}$)

Fonte: Autoria própria.

Tabela 5 – Média e Variância (entre parênteses) das métricas de perda de cada modelo no treinamento (Train) nas 10 ultimas épocas.

Versão	**Modelo**	**Train/Box_loss**	**Train/Cls_loss**	**Train/Obj_loss**
V1	YoloV5n	$1{,}6 \times 10^{-2}$ ($4{,}9 \times 10^{-7}$)	$5{,}3 \times 10^{-3}$($3{,}9 \times 10^{-7}$)	$8{,}7 \times 10^{-3}$($4{,}9 \times 10^{-8}$)
V1	YoloV5s	$1{,}3 \times 10^{-2}$($3{,}9 \times 10^{-7}$)	$2{,}6 \times 10^{-3}$($1{,}7 \times 10^{-7}$)	$7{,}2 \times 10^{-3}$($4{,}1 \times 10^{-8}$)
V1	YoloV5m	$1{,}0 \times 10^{-2}$($1{,}9 \times 10^{-7}$)	$2{,}1 \times 10^{-3}$($1{,}3 \times 10^{-7}$)	$6{,}0 \times 10^{-3}$($4{,}5 \times 10^{-8}$)
V1	YoloV5l	$9{,}6 \times 10^{-3}$($3{,}0 \times 10^{-7}$)	$1{,}7 \times 10^{-3}$($1{,}3 \times 10^{-7}$)	$5{,}4 \times 10^{-3}$($5{,}0 \times 10^{-8}$)
V2	YoloV5n	$1{,}1 \times 10^{-2}$($2{,}5 \times 10^{-8}$)	$1{,}2 \times 10^{-3}$($1{,}1 \times 10^{-8}$)	$7{,}5 \times 10^{-3}$($7{,}6 \times 10^{-9}$)
V2	YoloV5s	$1{,}0 \times 10^{-2}$($9{,}2 \times 10^{-8}$)	$1{,}0 \times 10^{-3}$($2{,}2 \times 10^{-8}$)	$6{,}6 \times 10^{-3}$($3{,}6 \times 10^{-9}$)
V2	YoloV5m	$1{,}1 \times 10^{-2}$($3{,}7 \times 10^{-8}$)	$1{,}5 \times 10^{-3}$($7{,}2 \times 10^{-9}$)	$6{,}4 \times 10^{-3}$($3{,}3 \times 10^{-9}$)

Fonte: Autoria própria.

Tabela 6 – Média e Variância (entre parenteses) das métricas de perda de cada modelo na validação (Val) nas 10 ultimas épocas.

Versão	**Modelo**	**Val/Box_loss**	**Val/Cls_loss**	**Val/Obj_loss**
V1	YoloV5n	$2{,}8 \times 10^{-2}$($9{,}0 \times 10^{-8}$)	$1{,}6 \times 10^{-2}$($1{,}0 \times 10^{-7}$)	$5{,}7 \times 10^{-3}$($1{,}1 \times 10^{-9}$)
V1	YoloV5s	$2{,}8 \times 10^{-2}$($3{,}0 \times 10^{-8}$)	$1{,}6 \times 10^{-2}$($4{,}3 \times 10^{-8}$)	$6{,}2 \times 10^{-3}$($1{,}4 \times 10^{-9}$)
V1	YoloV5m	$2{,}4 \times 10^{-2}$($1{,}5 \times 10^{-8}$)	$1{,}8 \times 10^{-2}$($4{,}2 \times 10^{-8}$)	$6{,}4 \times 10^{-3}$($7{,}2 \times 10^{-10}$)
V1	YoloV5l	$2{,}5 \times 10^{-2}$($1{,}2 \times 10^{-7}$)	$1{,}5 \times 10^{-2}$($6{,}0 \times 10^{-8}$)	$6{,}2 \times 10^{3}$($7{,}6 \times 10^{-9}$)
V2	YoloV5n	$1{,}6 \times 10^{-2}$($1{,}4 \times 10^{-10}$)	$1{,}1 \times 10^{-2}$($1{,}1 \times 10^{-9}$)	$7{,}1 \times 10^{-3}$($1{,}4 \times 10^{-10}$)
V2	YoloV5s	$1{,}4 \times 10^{-2}$($4{,}2 \times 10^{-11}$)	$6{,}2 \times 10^{-3}$($2{,}2 \times 10^{-9}$)	$6{,}8 \times 10^{-3}$($6{,}7 \times 10^{-11}$)
V2	YoloV5m	$1{,}5 \times 10^{-2}$($8{,}2 \times 10^{-8}$)	$6{,}7 \times 10^{-3}$($2{,}1 \times 10^{-7}$)	$7{,}0 \times 10^{-3}$($2{,}5 \times 10^{-8}$)

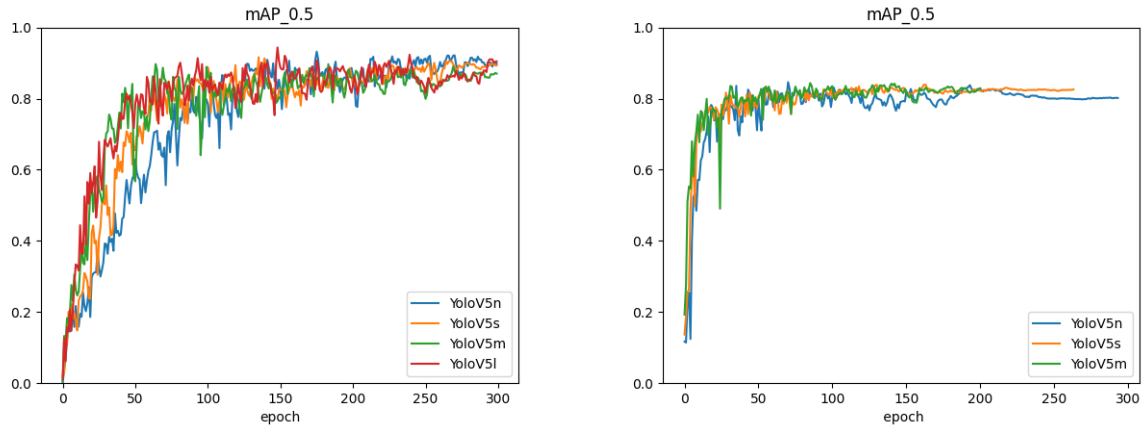
Fonte: Autoria própria.

5.2 Avaliação da YoloV5 ao longo das épocas

Após retirar os melhores valores de todas as métricas ao longo das épocas para avaliação do modelo, utilizamos os dados gerados época por época para avaliar o desempenho do modelo ao longo das épocas. Com isso montamos gráficos, tanto para os resultados obtidos com o EMDS5v1 quanto com o EMDS5v2, que mostram este desempenho e a evolução do nosso modelo ao longos das épocas. E estes dados serão apresentados na seguinte ordem: do lado esquerdo será apresentado o gráfico da versão 1 do EMDS5 e do lado direito os gráficos da versão 2 do EMDS5.

Analisando os gráficos que foram gerados com os dados das 300 épocas é possível notar que em relação ao mAP_0.5 e o mAP_0.5:0.95 do gráficos da Figuras 19 e 20 que na versão 1 houve uma maior variação em relação a versão 2 porém tendo suas versões mais próximas de 1, ou seja, obteve melhores resultados mesmo variando mais, pois a versão 2 ao chegar em um determinado ponto a variação fica menor e os dados se mantém estabilizados ao longo das épocas. É possível notar analisando por exemplo a YoloV5m no qual ele para antes de concluir as 300 épocas assim demonstrando que a partir daquele momento os resultados não mudariam. Como poderá ser visto mais adiante isso pode estar ocorrendo na versão 2 a partir de aproximadamente 150 épocas devido a um processo de *overfitting*, indicando que o treinamento não deveria ir além daquela quantidade épocas.

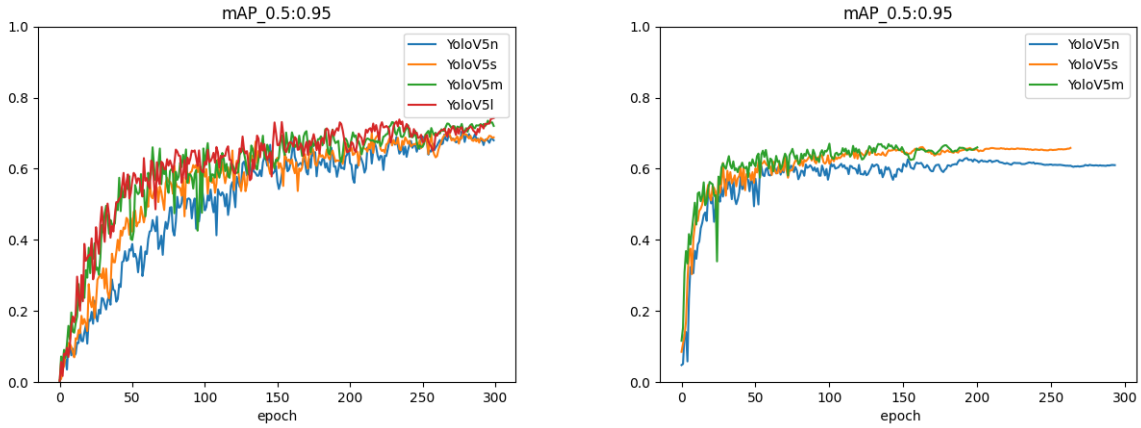
Figura 19 – Métrica mAP_0.5. No lado esquerdo o resultado para a V1 e no lado direito para a V2.



Fonte: Autoria própria.

Já quando analisamos a precisão e o *recall* dos gráficos das Figuras 21 e 22 nos mostram o desempenho em relação ao quão preciso nosso modelo foi para detectar os objetos e o quão capaz foi de identificar as instâncias das classes, nota-se que em relação a precisão a versão 1 demonstrou melhores resultados mesmo contendo uma variação maior nos dados. Já a versão 2, mesmo com uma variação menor, mostra que a partir de uma

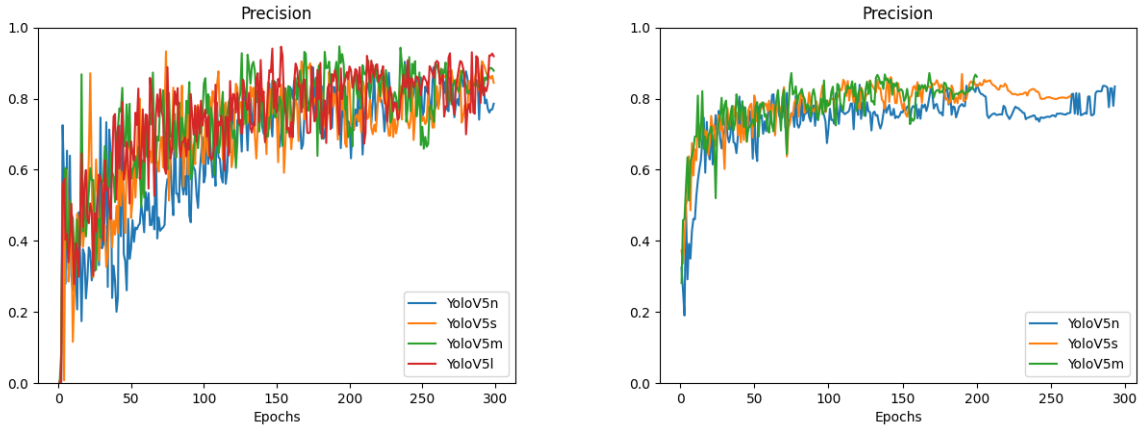
Figura 20 – Métrica mAP_0.5:0.95. No lado esquerdo o resultado para a V1 e no lado direito para a V2.



Fonte: Autoria própria.

determinada época os dados, mesmos variando, apresentam-se estagnados e com isso seus treinamentos acabam finalizando antes, como por exemplo a YoloV5s e Yolov5m.

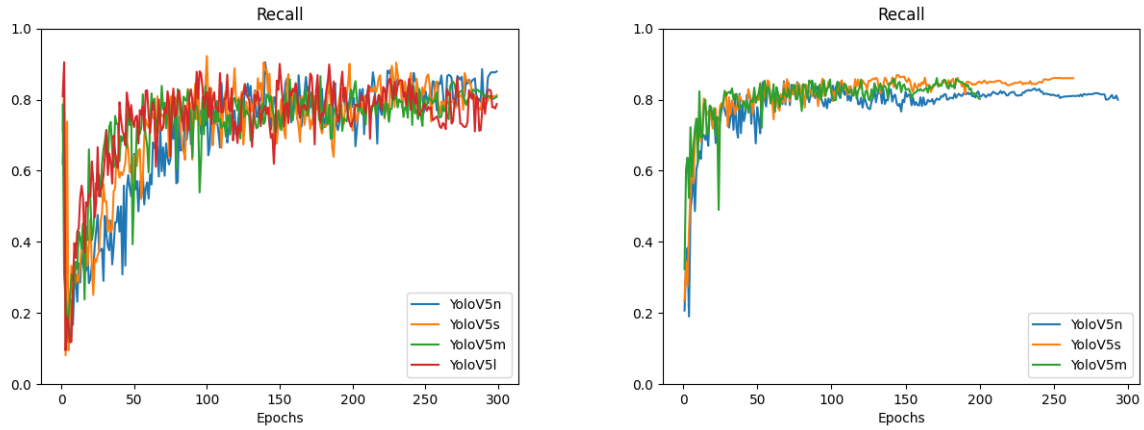
Figura 21 – Métrica Precisão. No lado esquerdo o resultado para a V1 e no lado direito para a V2.



Fonte: Autoria própria.

Como citado anteriormente estes gráficos representam o desempenho do modelo em relação ao acerto, ou seja, o quanto nosso modelo acertou durante as 300 épocas. Porém é preciso avaliar o quanto nosso modelo errou e se esse erro foi diminuindo ao longo das épocas, com isso será apresentado os gráficos criados que demonstram o quanto nosso modelo aprendeu e consequentemente o quanto ele errou, e assim avaliando durante as épocas se nosso modelo errou mais ou menos durantes elas. Esses dados serão apresentados de forma em que do lado esquerdo é representada a versão 1 e do lado direito a versão 2 do EMDS5.

Figura 22 – Métrica Recall. No lado esquerdo o resultado para a V1 e no lado direito para a V2.



Fonte: Autoria própria.

Analisando os gráficos da Figura 23 que foram gerados com os dados das 300 épocas é possível notar que em relação as métricas de perda do treinamento da versão 1 possui variações ao longo das épocas diferente da versão 2. Na versão 2 a perda decai mais rapidamente que na versão 1.

O Box_loss teve menos variações entre os modelos e já a Box_loss da versão 2 teve uma varição maior entre seus modelos e dois dos modelos demonstraram que ao chegar em uma determinada época os valores não mudariam e assim não chegando a gerar resultados até a época 300.

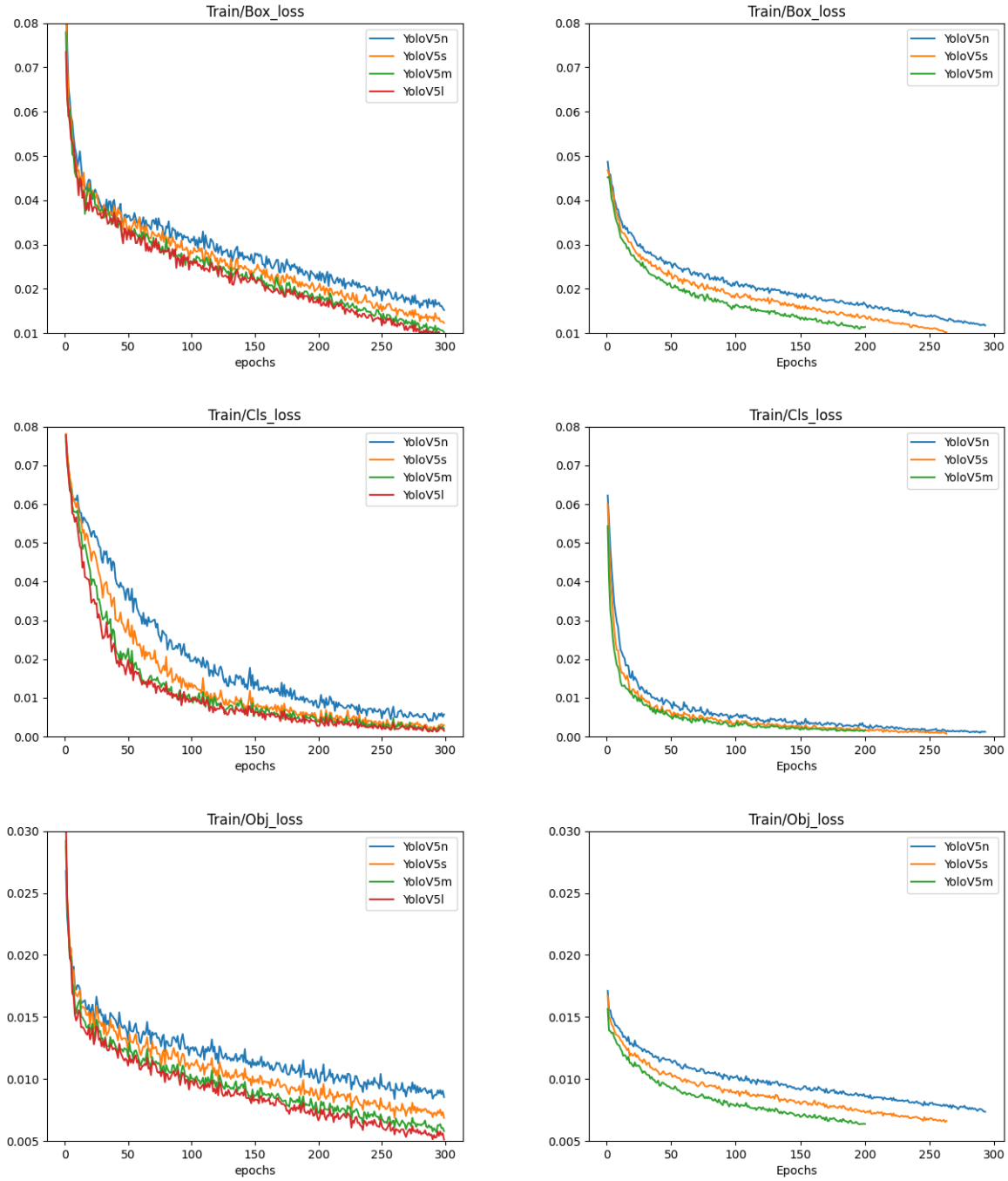
Em relação ao Cls_loss a versão 1 demonstrou maior variação entre seus modelos diferentemente da versão 2 que os modelos trouxeram resultados mais próximos ao longo das 300 épocas, porém apesar da pouca variação da versão 2 é possível notar que dois modelos não chegaram até as 300 épocas demostrando que a partir de uma determinada época os resultados seriam os mesmos.

E em relação ao Obj_loss a versão 1 demonstrou menor variação entre os modelos e assim percorrendo as 300 épocas, já a versão dois demonstrou uma variação maior entre seus modelos e que a partir de uma determinada épocas os resultados não mudam e assim não concluindo as 300 épocas.

Ao realizar à analise dos gráficos na Figura 24, que foram gerados a partir dos dados de validação das 300 épocas, é possível notar que em relação as métricas de perda ambas as versões mostraram ter variações diferentes. Porém, o processo de *overfitting* se encontra presente em todas as métricas da versão 2, podendo ser visto com o aumento da perda ao longo das épocas a partir da época 150 aproximadamente, enquanto o valor de perda continuava a cair no treinamento na mesma época.

Analisando as métricas separadamente, o Box_loss da versão 1 apresentou uma varição de dados maior que a versão 2 ao longo das épocas, já o Cls_loss apesar de ambos

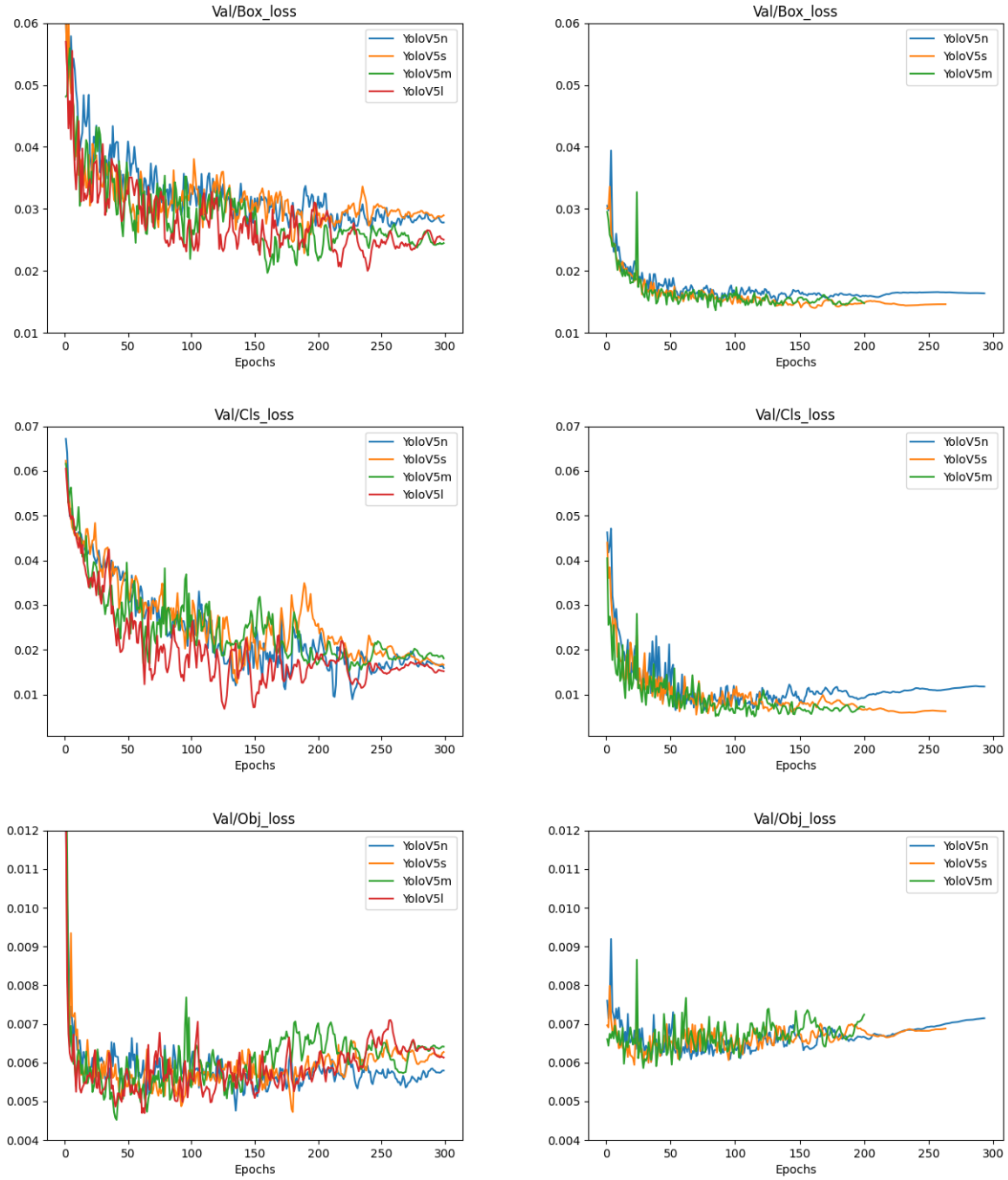
Figura 23 – Métricas de perda (loss) - Treinamento. No lado esquerdo os resultados para a V1 e no lado direito para a V2.



Fonte: Autoria própria.

terem variado bastante a versão 1 variou ao longo das épocas diferentemente da versão 2 que variou mais em um intervalo de épocas menor, já em relação ao Obj_loss a versão 1 variou menos que a versão 2, além disso a versão 2 ainda demonstrou que ao se aproximar da época 300 os valores voltaram a aumentar, mostrando que nosso modelo na versão 2 pode ter entrado em *overfitting*.

Figura 24 – Métricas de perda (loss) - Validação. No lado esquerdo os resultados para a V1 e no lado direito para a V2.



Fonte: Autoria própria.

5.3 Avaliação do melhor peso encontrado pela YoloV5

Após termos todos esses dados que foram obtidos durante o treinamento do nosso modelo, utilizamos o comando detect.py para avaliar o melhor peso (best.pt) que nosso modelo conseguiu identificar durante o treinamento, e com isso montarmos as matrizes

de confusão para avaliar o quão preciso esse peso identificado é ao realizar a detecção de objetos.

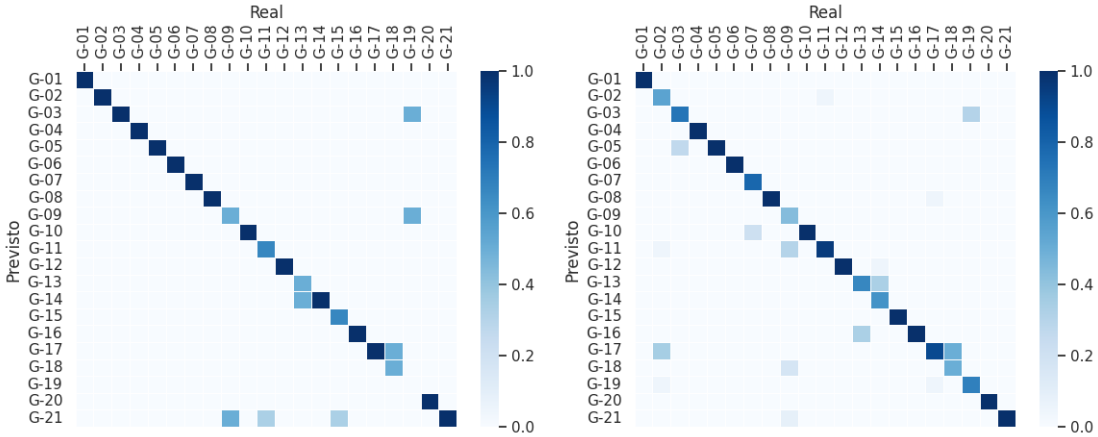
Para gerar as matrizes de confusão utilizamos o comando detect.py passando o peso ”best.pt”, utilizando o comando para que salve um arquivo csv com os resultados da da detecção, e assim salvando na pasta as imagens demonstrando se realizou a detecção e o arquivo csv contendo os dados: nome da imagem, classe detectada e o valor de confiança que o modelo acredita ser aquela classe. Para isso foi utilizado o seguinte comando porem informando os caminhos de acordo com os locais que as pastas se encontravam:

```
python detect.py --weights path/best.pt --source path/
```

Para entender melhor os comandos disponíveis e conseguir adaptar de acordo com o que é necessário para ter os dados foi utilizado os dados disponibilizados por Jocher (2020)

Analisando as matrizes de confusão apresentadas na Figura 25 que representa o modelo YoloV5N, é possível verificar que o modelo conseguiu detectar corretamente diversas vezes de acordo com as classes esperadas, porem também apresentou casos onde foram identificadas varias classes, como por exemplo na linha G21 da V1, onde além da classe esperada o modelo também detectou outras classes e ocorreu também de não conseguir realizar a detecção como por exemplo na linha G-19 também da V1. Na V2 também ocorre de o modelo, em alguns casos, detectar mais de uma classe.

Figura 25 – Matriz de confusão YoloV5n. Lado esquerdo V1 e lado direito V2.

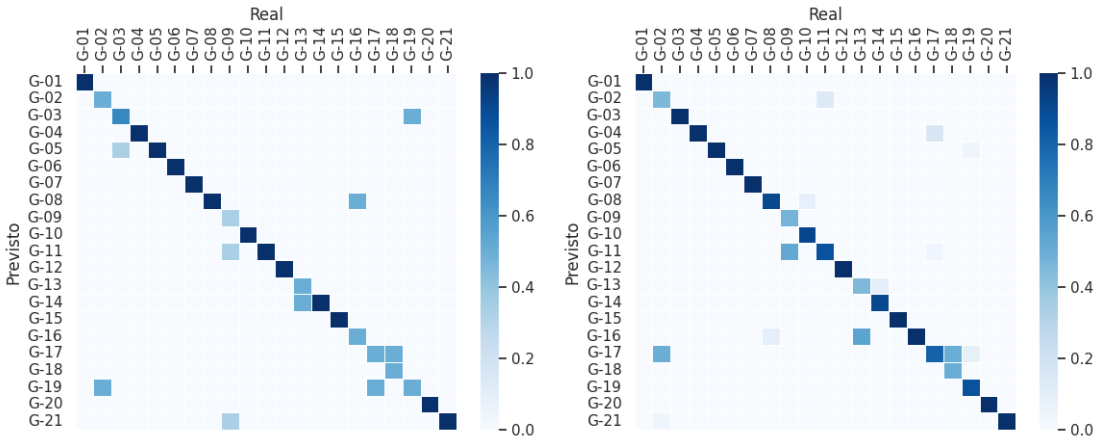


Fonte: Autoria própria.

Já na Figura 26 representa o modelo YoloV5S, onde é possível analisar que assim como o modelo N também ocorreu casos em que o modelo detectou outras classes além das reais que eram esperadas.

Já a Figura 27 representa o modelo YoloV5M, e é possível notar que em ralação aos modelo N e S da YoloV5 é um dos modelos que menos detectou classes diferentes da real esperada em relação aos outros modelos. Porém assim como na versão N o modelo

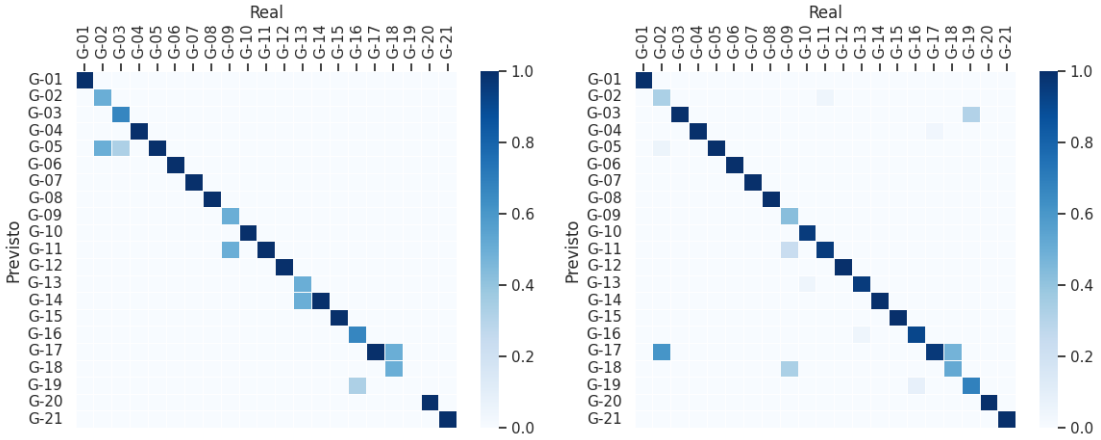
Figura 26 – Matriz de confusão YoloV5s. Lado esquerdo V1 e lado esquerdo V2.



Fonte: Autoria própria.

apresentou um caso onde não conseguiu realizar a detecção de objeto(s) na imagem de forma correta e apenas detectou uma classe que não era a real esperada e isso é demonstrado na linha G-19, no qual deveria estar marcado na coluna G-19 porém esta na G-16.

Figura 27 – Matriz de confusão Yolov5m. Lado esquerdo V1 e lado esquerdo V2.

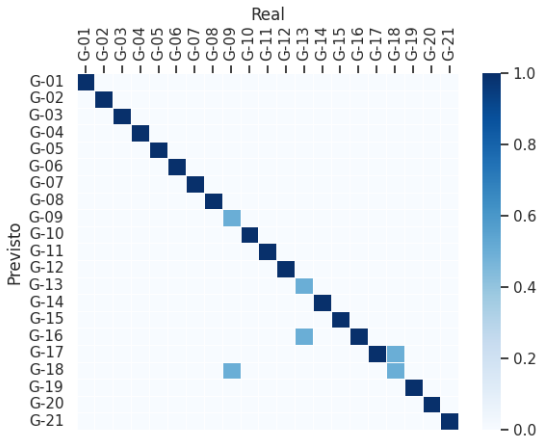


Fonte: Autoria própria.

Já a matriz de confusão apresentada na Figura 28 que representa o modelo YoloV5L, a qual só foi gerada pela versão 1 devido não ter sido possível gerar para versão 2 e realizando a analise da matriz é possível notar que foi o modelo que menos detectou outras classes além da real em relação aos outros modelos, porem mesmo assim ainda apresentou casos onde foi detectado outras classes como por exemplo nas linhas G-16, G-17 e G-18.

Além de nos retornar um arquivo csv que foi utilizado para montar as matrizes de confusão, foi retornado também as imagens que foram lidas com o comando detect.py onde é possível analisar as imagens que o modelo conseguiu realizar a detecção dos objetos, seja de maneira correta ou não.

Figura 28 – Matriz de confusão YoloV5l - V1.



Fonte: Autoria própria.

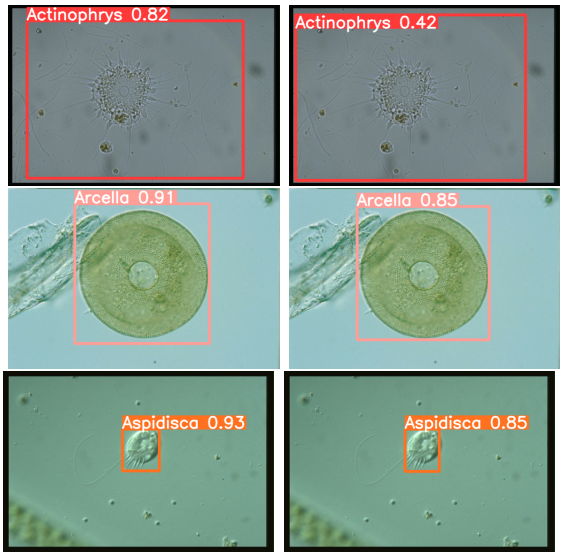
Abaixo são apresentados três casos para exemplificar esse resultado, no qual foi escolhido imagens da YoloV5n tanto para versão 1 quanto para versão 2 do EMDS5. Na Figura 29 são apresentados casos onde o modelo realizou a detecção de forma correta.

Na Figura 30 são apresentados casos onde o modelo realizou a **localização** de forma correta porém a **classificação** de forma incorreta. Isso pode ocorrer algumas vezes quando o modelo detecta mais de um objeto na imagem, como por exemplo nesta figura onde na primeira linha são apresentadas as classes Keratella Quadrala e Arcella; na segunda linha as classes Stentor e Aspidisca; e na terceira linha as classes Phacus e Euglypha. Em todos os casos o modelo realizou a localização de forma correta porem a classificação de forma incorreta, onde o modelo identificou para essas imagens outras classes.

Na Figura 31 são apresentados casos onde o modelo não conseguiu realizar a detecção do objeto presente na imagem, sendo que na primeira linha são apresentadas duas imagens da classe Arcella; na segunda linha as classes Paramecium e Codosiga; e na terceira linha as classes Rotifera e Epistylis. Porém o modelo não conseguiu detectar estas classes.

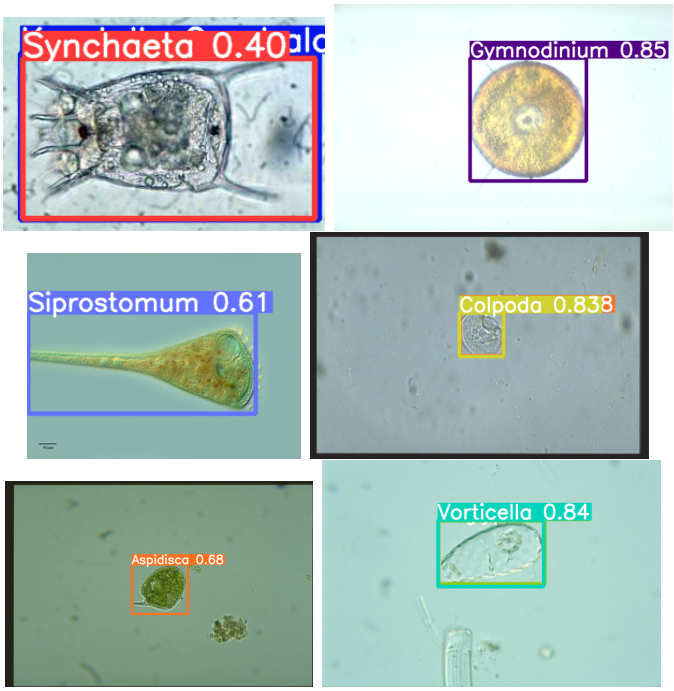
Outra análise realizada é a comparação do modelo com o EMDS5v1 e o EMDS5v2, onde ocorreu casos em que na versão 1 o modelo errou a classificação porém acertou na versão 2, como por exemplo na Figura 32, no qual esta apresentado imagens pertencente a classe 13 (Stentor) e na versão 1 o modelo erra a classificação e na versão 2, na versão da imagem gerada pelo data augmentation, conseguiu fazer a classificação de forma correta.

Figura 29 – Imagens geradas pelo detect onde o modelo acertou a detecção. V1 a esquerda e V2 a direita.



Fonte: Autoria própria.

Figura 30 – Exemplos de imagens geradas pelo *detect*, onde o modelo localizou corretamente porém errou em todas as classificações. À esquerda base na versão V1, à direita base na versão V2.

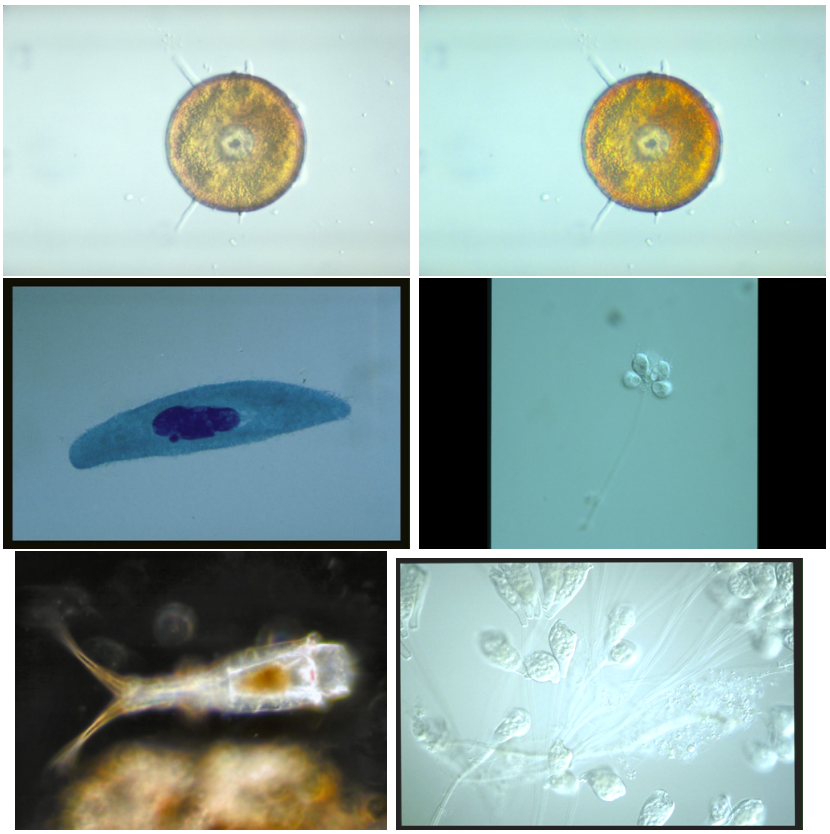


Fonte: Autoria própria.

5.3.1 Comparação com outros métodos

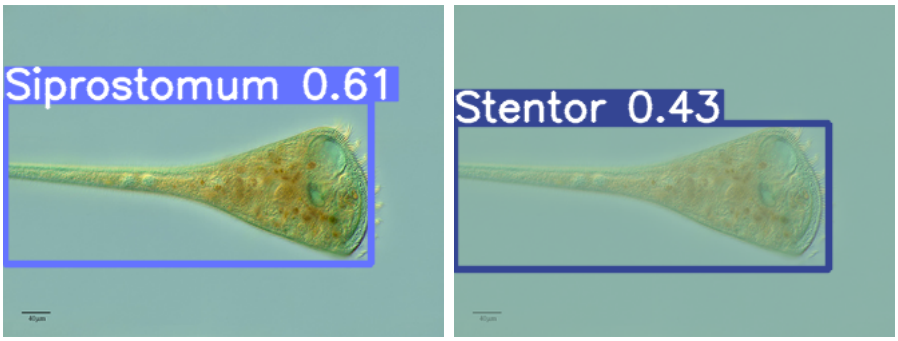
Realizando a comparação com outros métodos temos a Tabela 7 que foi criada com os dados retirados do trabalho de Zhao et al. (2022) e com os dados de teste gerados através do comando do python val.py, juntamente com as imagens separadas para teste.

Figura 31 – Exemplos de imagens geradas pelo *detect*, onde o modelo não conseguiu realizar a detecção. À esquerda base na versão V1, à direita base na versão V2.



Fonte: Autoria própria.

Figura 32 – Exemplo de imagens geradas pelo *detect*, onde modelo erra a classificação na versão 1 e acerta na versão 2. À esquerda base na versão V1, à direita base na versão V2.



Fonte: Autoria própria.

Estão presentes nessa tabela os dados de mAP de cada classe do *dataset* e no final o mAP de todas, sendo apresentado em negrito o melhor resultado pertencente a cadas classe.

Tabela 7 – Tabela comparativa de mAP:0.5 dos métodos utilizados em Zhao et al. (2022) e com dados obtidos neste trabalho, com as versões EMDS5-V1 e EMDS5-V2.

Classe	**Fast RCNN**	**Mask RCNN**	**EMDS5_V1 (YOLOV5L)**	**EMDS5_V2 (YOLOV5M)**
Actinophrys	**0.95**	0.705	0.401	0.383
Arcella	0.75	**0.85**	0.799	0.651
Aspidisca	0.39	0.40	**0.846**	0.832
Codosiga	0.13	0.18	**0.477**	0.314
Colpoda	0.52	0.35	**0.65**	0.508
Epistylis	0.24	**0.53**	0.351	0.371
Euglypha	0.68	0.25	**0.995**	0.847
Paramecium	0.70	0.70	**0.946**	0.902
Rotifera	**0.69**	0.40	0.456	0.399
Vorticella	0.30	0.15	**0.946**	0.914
Noctiluca	0.56	**0.90**	0.519	0.561
Ceratium	0.61	0.70	**0.946**	0.871
Stentor	0.47	0.65	**0.895**	0.783
Siprostomum	0.60	0.70	**0.896**	0.758
Keratella Quadrala	0.22	0.45	0.597	**0.662**
Euglena	0.37	0.25	**0.995**	0.852
Gymnodinium	0.53	**0.60**	0.0701	0.0303
Gonyaulax	0.25	0.28	**0.745**	0.532
Phacus	0.43	0.50	**0.648**	0.393
Stylongchia	0.48	0.68	**0.822**	0.754
Synchaeta	0.61	0.48	**0.946**	0.89
mAP:0.5 - Geral	0.50	0.51	**0.712**	0.629

Comando utilizado para versão 1:

```
python3 val.py --weights path/best.pt --data emds5_v1.yaml
```

Comando utilizado para versão 2:

```
python3 val.py --weights path/best.pt --data emds5_v2.yaml
```

Os comandos foram utilizados via terminal, no qual para o peso best.pt é passado o caminho da pasta que ele se encontra, o data é passado dentro do arquivo .yaml assim como o caminho da pasta que contem as imagens que o comando irá utilizar para realizar as devidas validações.

6 CONCLUSÃO

A ideia deste trabalho foi utilizar técnicas de aprendizado de máquina profundo para detectar microrganismos ambientais presentes em imagens. Foram utilizados um conjunto de dados público e realizados algumas estratégias de *data augmentation* para aumentar o conjuntos de dados e assim tendo duas versões deste conjunto (V1 e V2).

Após a análise dos resultados obtidos é possível concluir que o trabalho conseguiu atingir o objetivo proposto e o modelo utilizado (YoloV5), monstrando-se relativamente eficaz na detecção de objeto(s) em ambas versões. Na V1 os resultados foram melhores que na V2, mesmo utilizando os modelos mais simples da YoloV5 como o nano, que possui uma quantidade consideravelmente menor de parâmetros que os demais, obtendo mAP acima de 0.93.

As estratégias de aumento de dados utilizadas não tiveram um impacto de melhoria como esperado. Isso ocorreu devido às imagens terem ficado muito parecidas às originais após aplicação das técnicas de aumento de dados. O que acaba por não adicionar informações novas à rede neural convolucional e pode ter levado ao *overfitting* que foi verificado nessa versão (V2).

A utilização de uma estratégia de detecção com um estágio (YoloV5) em comparação com outras estratégias de detecção em dois estágios (Fast RCNN e Mask RCNN), na mesma base de dados, mostrou bons resultados nos testes feitos ao longo deste trabalho, apesar de não ter evitado possível *overfitting* em alguns casos.

Como trabalhos futuros podem ser testadas outras proporções de imagens na separação dos dados de treino, validação e teste, assim tendo uma variação de imagens para cada etapa diferente da utilizada neste trabalho.

Outro fator importante é encontrar outras maneiras de aumentar a base de dados evitando o *overfitting* que identificamos neste trabalho durante o treinamento e na avaliação dos resultados.

Referências

ACADEMY, D. S. *Capítulo 18 – Entropia Cruzada Para Quantificar a Diferença Entre Duas Distribuições de Probabilidade*. 2022. Ultimo acesso em: 8 de março de 2024. Disponível em: <https://www.deeplearningbook.com.br/entropia-cruzada-para-quantificar-a-diferenca-entre-duas-distribuicoes-de-probabilidade/>. Citado na página 16.

AMERICAN ACADEMY FOR MICROBIOLOGY. *From outside to inside: Environmental Microorganisms as Human Pathogens*. 2004. Ultimo acesso em: 07 de Fevereiro de 2024. Disponível em: <https://www.ncbi.nlm.nih.gov/books/NBK560445/>. Citado na página 1.

AWAN, A. A. *O que é aprendizagem profunda? Um tutorial para iniciantes*. 2024. Acessado em: 29 de outubro de 2024. Disponível em: <https://www.datacamp.com/pt/tutorial/tutorial-deep-learning-tutorial>. Citado na página 6.

DIWAN, T.; ANIRUDH, G.; TEMBHURNE, J. V. Object detection using yolo: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, Springer, v. 82, n. 6, p. 9243–9275, 2023. Citado 5 vezes nas páginas 5, 6, 8, 9 e 14.

GINORIS, Y. et al. Recognition of protozoa and metazoa using image analysis tools, discriminant analysis, neural networks and decision trees. *Analytica Chimica Acta*, Elsevier, v. 595, n. 1-2, p. 160–169, 2007. Citado 3 vezes nas páginas 2, 18 e 20.

GOMES, J.; VELHO, L. *Fundamentos da Computação Gráfica*. [S.l.]: IMPA, 2004. (Sére de computação de matemática). ISBN 9788524402005. Citado na página 4.

GONZALEZ, R. C.; WOODS, R. E. *Processamento Digital de Imagens*. [S.l.]: Pearson Universidades, 3a edição, 2009. Citado na página 6.

GOOGLE. *Classificação: Precisão, recall e métricas relacionadas*. 2024. Último acesso em: 10 de outubro de 2024. Disponível em: <https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall?hl=pt-br>. Citado 2 vezes nas páginas 13 e 14.

JECKEL, H.; DRESCHER, K. Advances and opportunities in image analysis of bacterial cells and communities. *FEMS Microbiology Reviews*, Oxford University Press, v. 45, n. 4, p. fuaa062, 2021. Citado 5 vezes nas páginas 2, 6, 7, 18 e 19.

JIANG, P. et al. A review of yolo algorithm developments. *Procedia Computer Science*, Elsevier, v. 199, p. 1066–1073, 2022. Citado na página 10.

JOCHER, G. *YOLOv5: A State-of-the-Art Object Detection Model*. 2020. Accessed: 2024-09-21. Disponível em: <https://github.com/ultralytics/yolov5/blob/master/detect.py>. Citado na página 43.

JOCHER, G. et al. ultralytics/yolov5: v7. 0-yolov5 sota realtime instance segmentation. *Zenodo*, 2022. Citado na página 23.

JUNIOR, C. de O. *Métricas para Regressão: Entendendo as métricas R^2, MAE, MAPE, MSE e RMSE*. 2021. Ultimo acesso em: 11 de fevereiro de 2025. Disponível em: <https://medium.com/data-hackers/prevendo-n%C3%BAmeros-entendendo-m%C3%A9tricas-de-regress%C3%A3o-35545e011e70>. Citado na página 15.

LI, Z. et al. Emds-5: Environmental microorganism image dataset fifth version for multiple image analysis tasks. *Plos one*, 2021. Citado 8 vezes nas páginas 1, 6, 18, 20, 21, 22, 23 e 31.

LIU, H. et al. Sf-yolov5: A lightweight small object detection algorithm based on improved feature fusion mode. *Sensors*, MDPI, v. 22, n. 15, p. 5817, 2022. Citado 5 vezes nas páginas 4, 7, 8, 9 e 18.

MAGNO, L. *Matriz de Confusão: nunca mais se confunda utilizando esse exemplo*. 2023. Ultimo acesso em: 19 de Abril de 2024. Disponível em: <https://medium.com/comunidadeds/matriz-de-confus%C3%A3o-nunca-mais-se-confunda-utilizando-esse-exemplo-35a9ac63b88a>. Citado na página 12.

PARK, S. *A guide to Two-stage Object Detection: R-CNN, FPN, Mask R-CNN*. 2021. Ultimo acesso em: 15 de Junho de 2024. Disponível em: <https://medium.com/codex/a-guide-to-two-stage-object-detection-r-cnn-fpn-mask-r-cnn-and-more-54c2e168438c>. Citado na página 8.

RAJESH, S. *Confusion Matrix for Multiclass Classification*. 2024. Accessed: 2024-12-30. Disponível em: <https://medium.com/@sreenilarajesh/confusion-matrix-for-multiclass-classification-fe7b6901a541>. Citado 2 vezes nas páginas 12 e 13.

REDMON, J. et al. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 779–788. Citado 6 vezes nas páginas , 10, 11, 12, 15 e 16.

SAXENA, S. *Underfitting and Overfitting in Machine Learning*. 2023. Acessado em: 12 de fevereiro de 2025. Disponível em: <https://www.analyticsvidhya.com/blog/2020/02/underfitting-overfitting-best-fitting-machine-learning/>. Citado 2 vezes nas páginas 16 e 17.

SCHAEFFER-FILHO, A. E. et al. *Escola de Computação PPGC UFRGS 50 Anos: Transformando Desafios em Oportunidades para o Futuro*. [S.l.]: Sociedade Brasileira de Computação - SBC, 2022. ISBN 978-85-7669-558-5. Citado na página 4.

SOLAWETZ, J. *Responding to the Controversy about YOLOv5*. 2020. Acessado em: 12 de fevereiro de 2025. Disponível em: <https://blog.roboflow.com/yolov4-versus-yolov5/>. Citado na página 11.

SOLAWETZ, J. *What is YOLOv5? A Guide for Beginners*. 2020. Acessado em: 12 de fevereiro de 2025. Disponível em: <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>. Citado na página 11.

SOLAWETZ, J. *What is YOLOv4? A Detailed Breakdown*. 2024. Acessado em: 12 de fevereiro de 2025. Disponível em: <https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/>. Citado na página 11.

TCHILIAN, F. *O que é acurácia: o conceito, a importância e como aplicar*. 2022. Ultimo acesso em: 17 de Maio de 2024. Disponível em: <https://blogbr.clear.sale/conheca-e-saiba-como-aplicar-a-acuracia>. Citado na página 13.

TENSORFLOW. *Aumento de dados*. 2022. Ultimo acesso em: 30 de Janeiro de 2024. Disponível em: <https://www.tensorflow.org/tutorials/images/data_augmentation?hl=pt-br>. Citado na página 29.

ULTRALYTICS. *Performance Metrics Deep Dive*. 2023. Ultimo acesso em: 19 de Abril de 2024. Disponível em: <https://docs.ultralytics.com/guides/yolo-performance-metrics/>. Citado 2 vezes nas páginas 14 e 15.

ULTRALYTICS. *Treina dados personalizados*. 2024. Ultimo acesso em: 30 de Janeiro de 2024. Disponível em: <https://docs.ultralytics.com/pt/yolov5/tutorials/train_custom_data/>. Citado na página 24.

ZHANG, J. et al. Applications of artificial neural networks in microorganism image analysis: A comprehensive review from conventional multilayer perceptron to popular convolutional neural network and potential visual transformer. *https://link.springer.com/article/10.1007/s10462-022-10192-7*, 2023. Citado na página 2.

ZHAO, P. et al. Emds-6: Environmental microorganism image dataset sixth version for image denoising, segmentation, feature extraction, classification, and detection method evaluation. *Frontiers in Microbiology*, Frontiers Media SA, v. 13, p. 829027, 2022. Citado 7 vezes nas páginas , 18, 19, 20, 21, 46 e 48.