|  |
| --- |
| UNIVERSIDADE REGIONAL DE BLUMENAU  CENTRO DE CIÊNCIAS EXATAS E NATURAIS  CURsO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO |
| ESTUDO COMPARATIVO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA CONTAGEM E IDENTIFICAÇÃO DE NEURÔNIOS ATÍPICOS  leonardo gian pottmayer |
| bLUMENAU  2025 |

|  |
| --- |
| LEONARDO GIAN POTTMAYER  ESTUDO COMPARATIVO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA PARA CONTAGEM E IDENTIFICAÇÃO DE NEURÔNIOS ATÍPICOS  Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.  Prof. Aurélio Faustino Hoppe - Orientador |
| bLUMENAU  2025 |
| Esta página deverá ser substituída pela folha de assinaturas entregue na Banca.  Digitalize a folha e cole aqui para a entrega da versão final do TCC.  Atenção: não ultrapasse as margens! |
|  |

Dedico este trabalho à minha família, e a todos os meus colegas e amigos que me apoiaram e auxiliaram em todo o processo de graduação.

AGRADECIMENTOS

À minha família que sempre me apoiou em minhas decisões.

Aos meus amigos, que estiveram ao meu lado e me ajudaram nestes últimos anos.

Ao meu orientador, professor Aurélio Faustino Hoppe, pelo apoio e contribuição na confecção desse trabalho e pelos conselhos e ajuda durante essa jornada.

Aos colegas Gabriel e Lucas, que forneceram o material necessário para a execução deste trabalho.

A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original.

Albert Einstein

RESUMO

A identificação e a contagem de neurônios atípicos em imagens de microscopia óptica representam um desafio relevante no campo da neurociência, pois o processo manual, tradicionalmente realizado por especialistas em laboratório, demanda tempo, exige alto nível de experiência e está sujeito a variabilidade entre avaliadores. Diante dessa limitação, este trabalho apresenta a aplicação de técnicas de aprendizado de máquina para automatizar a detecção desses neurônios, ampliando a análise iniciada por Krzizanowski e Carvalho (2024) e avaliando alternativas capazes de melhorar a padronização e a eficiência das análises morfológicas. Para isso, o estudo desenvolve um *pipeline* completo que abrange a preparação da base de dados, a aplicação de pré-processamentos e o uso controlado de *data augmentation*. A metodologia utiliza o modelo You Only Look Once (YOLOv8) em diferentes escalas, avaliando o impacto de variações no tamanho das imagens, na intensidade das transformações, nos parâmetros de treinamento e no uso ou não da técnica de *tiling*. Os resultados indicam que configurações mais conservadoras tendem a produzir métricas mais estáveis, enquanto combinações agressivas de aumento de dados nem sempre resultam em melhorias proporcionais, especialmente diante do tamanho reduzido do *dataset*. Observou-se também que o uso de *tiling*, embora aumente a granularidade das regiões analisadas, amplia o tempo de processamento e reduz o desempenho geral dos modelos. Entre as variantes testadas, modelos maiores como o YOLOv8x demonstraram melhor equilíbrio entre custo computacional e eficácia, apresentando desempenho superior nas métricas de detecção. Conclui-se que o YOLOv8, aliado a um *pipeline* modular, reprodutível e adaptado ao *dataset* utilizado, constitui uma alternativa viável para apoiar a identificação automática de neurônios atípicos, oferecendo maior padronização para as análises. O estudo também evidencia limitações importantes, como a quantidade restrita de exemplos, e sugere como trabalhos futuros a ampliação da base de imagens, o refinamento de hiperparâmetros e a marcação explícita dos neurônios saudáveis nas imagens.

Palavras-chave: Neurônios atípicos. Microscopia. Detecção. Aprendizado de máquina. YOLO. Visão computacional.

ABSTRACT

The identification and counting of atypical neurons in optical microscopy images represent a significant challenge in the field of neuroscience, since the manual process traditionally performed by laboratory specialists is time-consuming, requires substantial expertise and is subject to variability among evaluators. In view of this limitation, this study aims to investigate and apply machine learning techniques to automate the detection of these neurons, expanding the analysis initiated by Krzizanowski and Carvalho (2024) and evaluating alternatives capable of improving the standardization and efficiency of morphological assessments. To this end, the study develops a complete pipeline encompassing dataset preparation, image preprocessing and the controlled use of data augmentation. The methodology employs the YOLOv8 model in different scales, assessing the impact of variations in image size, transformation intensity, training parameters and the use or non-use of the tiling technique. The results indicate that more conservative configurations tend to produce more stable metrics, while aggressive augmentation strategies do not always yield proportional improvements, particularly given the limited size of the dataset. It was also observed that tiling, although increasing the granularity of the regions analyzed, raises processing time and reduces overall model performance. Among the tested variants, larger models such as YOLOv8x demonstrated a better balance between computational cost and effectiveness, achieving superior detection metrics. The study concludes that YOLOv8, combined with a modular and reproducible pipeline adapted to the dataset used, represents a viable alternative to support the automatic identification of atypical neurons, offering greater standardization for the analyses. The study also highlights important limitations, such as the restricted number of samples, and suggests future work involving dataset expansion, deeper hyperparameter tuning and the explicit annotation of healthy neurons in the images.

Key-words: Atypical neurons. Microscopy. Detection. Machine learning. YOLOv8. Computer vision.

LISTA DE Figuras

[Figura 1 – Representação do neurônio 16](#_Toc215565439)

[Figura 2 – Formação do espaço pericelular 17](#_Toc215565440)

[Figura 3 – Neurônio sofrendo morte celular devido a doença de Alzheimer. 19](#_Toc215565441)

[Figura 4 – Arquitetura de uma Rede Neural Convolucional 20](#_Toc215565442)

[Figura 5 – Aplicação de *max-pooling* 2x2 em uma imagem 4x4 21](#_Toc215565443)

[Figura 6 – Arquitetura YOLOv8 22](#_Toc215565444)

[Figura 7 – Telas de listagem de informações do aplicativo 28](#_Toc215565445)

[Figura 8 – Etapas para detecção de neurônios atípicos 32](#_Toc215565446)

[Figura 9 – Processo de captura das imagens 33](#_Toc215565447)

[Figura 10 – Exemplo do resultado das anotações na ferramenta LabelBox 34](#_Toc215565448)

[Figura 11 – Visualização do arquivo ndjson 34](#_Toc215565449)

[Figura 12 – Diagrama fluxo do *pipeline* geral 40](#_Toc215565450)

[Figura 13 – Exemplo de *tiling* 42](#_Toc215565451)

[Figura 14 – Exemplos de imagens de *debug* 43](#_Toc215565452)

[Figura 15 – *Mosaic* aplicado com imagens em resolução original 45](#_Toc215565453)

[Figura 16 – *Mosaic* aplicado com *tiles* 45](#_Toc215565454)

[Figura 17 – Aplicação do *mixup* 46](#_Toc215565455)

[Figura 18 – Gráfico de precisão, recall e F1-score do melhor experimento 52](#_Toc215565456)

[Figura 19 – Exemplo de imagem com anotações de *ground truth* e predições 53](#_Toc215565457)

[Figura 20 – Percentual de acerto por imagem 56](#_Toc215565458)

[Figura 21 – Comparativo entre imagens com melhor e pior desempenho 57](#_Toc215565459)

LISTA DE Quadros

[Quadro 1 – Resultado de buscas por termos contidos em artigos em língua portuguesa 24](#_Toc215565460)

[Quadro 2 – Resultado de buscas por termos contidos em artigos em língua inglesa 24](#_Toc215565461)

[Quadro 3 – Síntese dos trabalhos correlatos selecionados 25](#_Toc215565462)

[Quadro 4 – Artificial Neural Networks in Action for an Automated Cell-Type Classification of Biological Neural Networks 26](#_Toc215565463)

[Quadro 5 – Aprendizado de máquina aplicado à detecção, classificação e análise de células tumorais em imagens de microscopia de luz visível 26](#_Toc215565464)

[Quadro 6 – Algoritmos de Aprendizado de Máquina para Classificação de Células Nucleadas do Sangue Periférico – Uma Experiência do Projeto Hemovision 27](#_Toc215565465)

[Quadro 7 – Requisitos Funcionais do modelo 30](#_Toc215565466)

[Quadro 8 – Requisitos Não Funcionais do modelo 31](#_Toc215565467)

[Quadro 9 – Download das imagens e construção do arquivo manifest.ndjson 35](#_Toc215565468)

[Quadro 10 – Início da classe ConfiguracaoExperimento 37](#_Toc215565469)

[Quadro 11 – Segunda parte da classe ConfiguracaoExperimento 38](#_Toc215565470)

[Quadro 12 – Terceira parte da classe ConfiguracaoExperimento 39](#_Toc215565471)

[Quadro 13 – Estrutura final de diretórios 41](#_Toc215565472)

[Quadro 14 – Função de treinamento do modelo YOLO 44](#_Toc215565473)

[Quadro 15 – Conteúdo do arquivo de arquitetura P2 customizada 49](#_Toc215565474)

[Quadro 16 – Exemplos de imagens da fase de teste 53](#_Toc215565475)

[Quadro 17 – Análise de correspondências de dez imagens 56](#_Toc215565476)

Lista de tabelas

[Tabela 1 – Desempenho médio por tamanho de modelo 51](#_Toc215565477)

[Tabela 2 – Desempenho médio por tamanho de imagem 51](#_Toc215565478)

[Tabela 3 – Experimentos com melhor desempenho na fase de teste 52](#_Toc215565479)

[Tabela 4 – Desempenho em experimentos com configurações diversas 55](#_Toc215565480)

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

AVC – Acidente Vascular Cerebral

COCO – Common Objects in Context

CNN – Convolutional Neural Network

CSV – Comma-Separated Values

DA – Doença de Alzheimer

DNN – Deep Neural Network

GPU – Graphics Processing Unit

HSV – Hue, Saturation, Value

IoU – Intersection over Union

JSON – JavaScript Object Notation

MAPE – Mean Absolute Percentage Error

MIME – Multipurpose Internet Mail Extensions

NMS – Non-Maximum Suppression

RAM – Random Access Memory

RF – Requisitos Funcionais

RMSE – Root Mean Squared Error

RNF – Requisitos Não Funcionais

SVM – Support Vector Machine

URL – Uniform Resource Locator

YAML – Yet Another Markup Language

YOLO – You Only Look Once

SUMÁRIO

[1 Introdução 13](#_Toc215565481)

[1.1 OBJETIVOS 15](#_Toc215565482)

[1.2 estrutura 15](#_Toc215565483)

[2 FUNDAMENTAÇÃO TEÓRICA 16](#_Toc215565484)

[2.1 NEURÔNIOS 16](#_Toc215565485)

[2.2 REDES NEURAIS CONVOLUCIONAIS 19](#_Toc215565486)

[2.3 YOLO 21](#_Toc215565487)

[2.4 TRABALHOS CORRELATOS 24](#_Toc215565488)

[2.5 MODELO ATUAL 27](#_Toc215565489)

[3 DESENVOLVIMENTO 30](#_Toc215565490)

[3.1 requisitos 30](#_Toc215565491)

[3.2 DESCRIÇÃO DO MODELO 31](#_Toc215565492)

[3.3 BUSCA E PREPARAÇÃO DA BASE DE DADOS 33](#_Toc215565493)

[3.4 CONFIGURAÇÃO DOS EXPERIMENTOS 36](#_Toc215565494)

[3.5 EXECUÇÃO do experimento 39](#_Toc215565495)

[3.5.1 Criação da estrutura de pastas para o experimento 40](#_Toc215565496)

[3.5.2 Construção do *dataset* 41](#_Toc215565497)

[3.5.3 Treinamento do modelo 43](#_Toc215565498)

[3.5.4 Geração de relatórios e execução do teste do modelo 46](#_Toc215565499)

[3.6 CONFIGURAÇÕES Dos EXPERIMENTOs 47](#_Toc215565500)

[3.7 RESULTADOS E DISCUSSÕES 50](#_Toc215565501)

[4 CONCLUSÕES 59](#_Toc215565502)

[Referências 62](#_Toc215565503)

# Introdução

As doenças neurológicas representam um dos maiores desafios da saúde pública global, impactando bilhões de pessoas em todo o mundo e sendo a principal causa de morbidade e incapacidade em escala internacional. Estima-se que mais de 3 bilhões de indivíduos convivam com condições como Acidente Vascular Cerebral (AVCs), enxaquecas, meningite, demências e lesões cerebrais neonatais. Desde 1990, a carga dessas doenças cresceu 18%, impulsionada pelo envelhecimento populacional, maior expectativa de vida e fatores como poluição, obesidade e dietas inadequadas. Além disso, cerca de 80% das mortes relacionadas ocorrem em países de baixa e média renda, evidenciando graves desigualdades no acesso a cuidados de saúde e na implementação de estratégias preventivas (GBD 2021 Nervous System Disorders Collaborators, 2024).

Diante desse cenário, segundo Moreira (2013), a neurociência tem buscado compreender os mecanismos subjacentes às doenças neurológicas, e a morfologia neuronal desempenha um papel central, especialmente no contexto de estruturas críticas como o hipocampo. Alterações morfológicas, que se referem a mudanças na forma e estrutura dos neurônios, podem refletir disfunções no processamento de informações e na conectividade cerebral, afetando diretamente funções cognitivas como memória e aprendizado. Ainda de acordo com o autor, essas funções dependem fortemente da integridade estrutural e funcional do hipocampo, o que torna essa região especialmente relevante no estudo de condições neurológicas e transtornos do desenvolvimento.

Para Amaral e Lavenex (2007), o hipocampo se destaca como uma estrutura crucial para funções cognitivas como a memória episódica e espacial. Ele abriga diferentes tipos de neurônios, incluindo os piramidais, granulares e interneurônios, que desempenham papéis específicos e complementares no processamento de informações. Esses neurônios possuem estruturas fundamentais para sua função, como dendritos e axônios: os dendritos são projeções ramificadas que recebem informações de outros neurônios, enquanto os axônios são prolongamentos que transmitem sinais elétricos para células vizinhas. Em condições normais, esses componentes apresentam características bem definidas, como dendritos organizados e axônios integrados, garantindo a transmissão eficiente de impulsos nervosos e a conectividade funcional do hipocampo. Essas características são observadas nos chamados neurônios típicos, cuja integridade estrutural é essencial para a manutenção das funções cognitivas. Por outro lado, Cervantes (2019), destaca que os neurônios atípicos podem apresentar alterações morfológicas significativas, como dendritos desorganizados ou axônios danificados, que comprometem sua capacidade funcional. Essas alterações estruturais frequentemente estão associadas a condições neurológicas graves, como epilepsia, demência e lesões cerebrais traumáticas. A relação entre os diferentes tipos de neurônios e a ocorrência de morfologias atípicas no hipocampo reforça a importância de estudar essas alterações para compreender os mecanismos subjacentes às disfunções cognitivas e neurológicas.

Segundo Cervantes (2019), a identificação de neurônios atípicos, no entanto, representa um grande desafio para a neurociência. Tradicionalmente, essa tarefa é realizada manualmente por especialistas utilizando microscópios, um processo que é trabalhoso, demorado e suscetível a falhas humanas. Além disso, a subjetividade inerente ao método manual dificulta a padronização e a reprodutibilidade das análises, especialmente em cenários com grandes volumes de dados.

Troullinou *et al*. (2021) destacam que avanços em aprendizado de máquina têm oferecido soluções promissoras para superar essas limitações. Técnicas baseadas em redes neurais profundas (*Deep Neural Networks*, DNNs) demonstraram capacidade de automatizar a análise morfológica de neurônios, alcançando alta precisão na identificação de diferentes tipos celulares. Por exemplo, DNNs baseadas em sinais de cálcio identificaram padrões específicos de atividade neural em experimentos in vivo. Modelos aplicados à classificação de células tumorais também apresentaram resultados significativos, destacando a eficácia do aprendizado de máquina na detecção de padrões celulares complexos. Além de DNNs e Faster R-CNN, outras abordagens, como Máquinas de Vetores de Suporte (Support Vector Machines - SVM) e Florestas Aleatórias, ampliam as possibilidades de análise automatizada, explorando diferentes arquiteturas para atender a variados cenários (Angonezi, 2022).

Krzizanowski e Carvalho (2024) utilizaram a arquitetura Faster R-CNN, uma metodologia avançada de detecção de objetos, para a identificação automatizada de neurônios atípicos em imagens obtidas por microscópios e dispositivos móveis. Os autores relataram que o modelo acelerou significativamente as análises, permitindo uma melhor compreensão das alterações morfológicas associadas a diferentes condições neurológicas. Segundo os autores, a identificação automatizada e precisa de neurônios atípicos pode contribuir para diagnósticos mais precoces, maior padronização analítica e o desenvolvimento de intervenções terapêuticas mais eficazes.

A partir deste cenário, esta pesquisa busca responder à seguinte questão: “Quais técnicas de aprendizado de máquina apresentam melhor desempenho na identificação de neurônios atípicos em imagens microscópicas?” Para isso, serão realizados experimentos comparativos utilizando a base de dados do estudo de Krzizanowski e Carvalho (2024), com o objetivo de contribuir para o avanço das abordagens diagnósticas e terapêuticas em neurociência.

## OBJETIVOS

O objetivo principal deste trabalho é dar continuidade ao desenvolvimento do aplicativo proposto por Krzizanowski e Carvalho (2024) para a contagem e identificação automática de neurônios atípicos, utilizando técnicas avançadas de processamento de imagens e aprendizado de máquina, como a *Convolutional Neural Network* (CNN) YOLO, visando aprimorar a automação, a precisão e a eficiência na análise da morfologia neuronal.

Os objetivos específicos são:

1. aplicar novas técnicas de pré-processamento de imagens para aprimorar a qualidade das imagens de lâminas, com o objetivo de otimizar a segmentação e a extração de características morfológicas dos neurônios;
2. aperfeiçoar a classificação de neurônios atípicos utilizando CNNs, explorando novas abordagens de aprendizado de máquina para melhorar a acurácia e robustez do modelo;
3. validar o desempenho do sistema automatizado, comparando-o ao método manual e ao aplicativo desenvolvido por Krzizanowski e Carvalho (2024), com base em métricas como tempo de processamento, taxa de acerto e precisão, a fim de avaliar sua viabilidade e superioridade na análise de neurônios atípicos.

## estrutura

A estrutura do trabalho está divido em quatro capítulos. No primeiro capítulo está descrita a introdução do trabalho juntamente com o objetivo principal e objetivos específicos. No segundo capítulo está detalhada a fundamentação teórica, contendo os principais tópicos do trabalho e os trabalhos correlatos encontrados. O terceiro capítulo descreve o desenvolvimento do modelo, contendo as especificações funcionais e não funcionais do modelo desenvolvido, técnicas e bibliotecas utilizadas em cada etapa, implementação do modelo e apresentação dos resultados encontrados. Por fim, o quarto e último capítulo descreve a conclusão do trabalho, alinhando as expectativas e resultados, juntamente com as limitações e sugestões de trabalhos futuros.

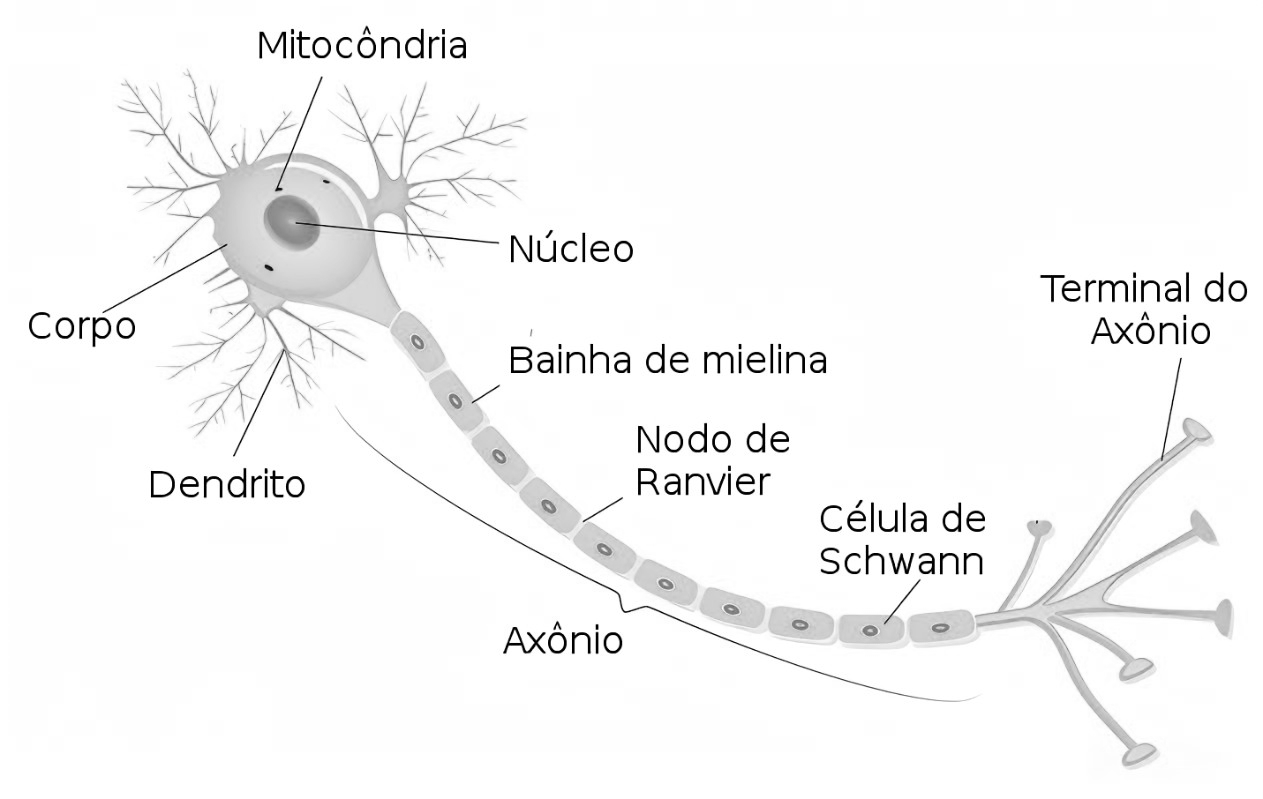
# FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma visão geral dos tópicos que servem como base para o estudo a ser desenvolvido. A seção 2.1 aborda os neurônios, enquanto a seção 2.2 trata das redes neurais convolucionais e a seção 2.3 discute o modelo YOLO. A seção 2.4 apresenta uma revisão dos trabalhos correlatos, que são considerados similares ao projeto proposto e, por último, a seção 2.5 detalha o modelo atual.

## NEURÔNIOS

No contexto de doenças neuronais, compreender a estrutura e o funcionamento dos neurônios é essencial, uma vez que esses são as unidades funcionais básicas do sistema nervoso. De acordo com Moreira (2013), o neurônio é uma célula altamente excitável que transmite informações por meio de impulsos eletroquímicos. Essa célula é composta por três regiões fundamentais: o corpo celular (soma), onde ocorrem funções metabólicas; os dendritos, que recebem estímulos externos; e o axônio, responsável pela condução do impulso nervoso até outras células (Figura 1). As terminações do axônio formam sinapses, estruturas especializadas nas quais neurotransmissores são liberados, possibilitando a comunicação entre os neurônios ou com células efetoras.

Figura – Representação do neurônio



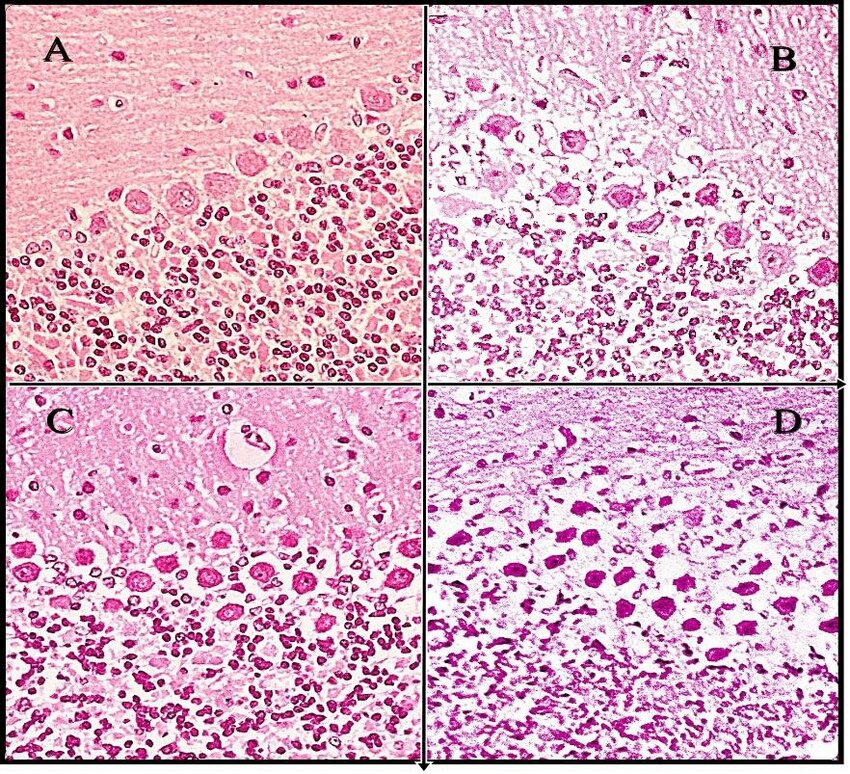
Fonte: Valeso (2021).

Machado (2013) complementa essa definição ao destacar que os neurônios possuem estruturas celulares típicas, como núcleo, retículo endoplasmático rugoso e liso, aparelho de Golgi e mitocôndrias. Em neurônios saudáveis, o núcleo caracteriza-se por ser claro, com cromatina frouxa, conforme descrito por Montanari (2006). O citoplasma abriga organelas essenciais para o metabolismo celular, desempenhando um papel crucial na manutenção da integridade e funcionalidade das células nervosas.

A integridade estrutural dos neurônios é um marcador fundamental para avaliar o estado de saúde neural. Neurônios típicos apresentam membranas intactas, núcleo esférico e claro, e uma organização eficiente de organelas. Em contraste, neurônios atípicos, frequentemente associados a condições patológicas, podem exibir sinais de degeneração, como picnose nuclear (condensação da cromatina) e vacuolização citoplasmática (Goel *et al.* 2022). Segundo Ma *et al.* (2011), células submetidas a estresses irreversíveis, que superam a capacidade de reparo dos seus sistemas, podem entrar em apoptose, ou seja, morte celular programada. Durante esse processo, os neurônios podem apresentar núcleos picnóticos, com cromatina condensada e em forma de corpúsculo heterocromático. Além disso, as organelas podem ser englobadas pelo sistema de endomembranas da célula, formando vacuolizações citoplasmáticas (Aljarari, 2023).

Nos estágios finais da degeneração celular, observa-se a formação de um espaço pericelular — um disco branco ao redor da célula — indicando a iminência da morte celular. Essa característica morfológica, juntamente com outras alterações estruturais, permite distinguir um neurônio saudável (típico) de um neurônio em degeneração ou morto (atípico). Essas distinções são fundamentais para avaliar o impacto de condições patológicas na saúde do indivíduo. Na Figura 2, observa-se essa diferença entre os grupos: enquanto o painel A representa o tecido saudável, sem espaços claros ao redor, os painéis B, C e D exibem áreas claras pericelulares — os discos brancos — caracterizando a formação do espaço perineuronal típico de processos degenerativos.

Figura – Formação do espaço pericelular



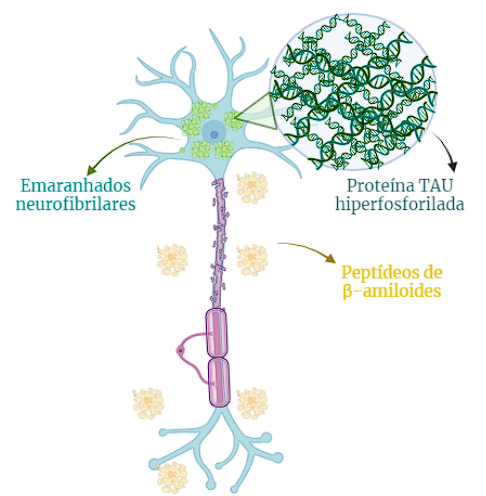
Fonte: Ali *et al.* (2020).

Dentre as doenças neurodegenerativas, a Doença de Alzheimer (DA) é uma das principais causas de demência em idosos. Sua fisiopatologia envolve o acúmulo extracelular de placas de beta-amiloide e a formação de emaranhados neurofibrilares intracelulares compostos por proteína Tau hiperfosforilada, levando à neuroinflamação, disfunção sináptica e morte progressiva de neurônios, sobretudo no hipocampo e no córtex cerebral (Machado *et al.*, 2020). Esses processos culminam em atrofia cerebral e prejuízo cognitivo acentuado. Estudos histológicos ilustram que neurônios afetados podem apresentar alterações morfológicas visíveis, como núcleos retraídos e organelas degradadas (Matanó; Pacheco; Zemlenoi*,* 2023).

A diferenciação entre neurônios típicos e atípicos é, portanto, essencial para a identificação de padrões morfológicos associados a diferentes doenças neurológicas. Essa análise possibilita inferências sobre o grau de comprometimento funcional das células, sendo um dos pilares no diagnóstico de doenças neurodegenerativas. Além disso, ela serve de base para modelos computacionais de detecção automática de anomalias, como os que utilizam técnicas de aprendizado de máquina, ampliando as possibilidades de diagnóstico precoce e apoio à pesquisa em neurociência computacional (Krzizanowski; Carvalho, 2024).

Nesse contexto, estudos como o de Matanó, Pacheco e Zemlenoi (2022) reforçam que a progressão da Doença de Alzheimer está intimamente relacionada ao acúmulo de peptídeos β-amiloides no meio extracelular e à formação de emaranhados neurofibrilares de Tau hiperfosforilada no interior do neurônio, conforme ilustrado na Figura 3. Esses depósitos patológicos alteram profundamente a organização estrutural da célula, comprometendo o transporte axonal, a estabilidade do citoesqueleto e, consequentemente, a manutenção da morfologia neuronal. A presença simultânea de inflamação crônica, desregulação metabólica e perda de sinapses contribui de forma progressiva para o declínio funcional observado nos estágios iniciais e avançados da doença (Krzizanowski; Carvalho, 2024). Assim, a visualização dessas alterações morfológicas torna-se um elemento chave para compreender a deterioração celular associada à DA.

Figura – Neurônio sofrendo morte celular devido a doença de Alzheimer.



Fonte: Matano, Pacheco e Zemlenoi (2022).

A partir dessas observações histopatológicas, torna-se evidente a importância de distinguir, de maneira sistemática, neurônios estruturalmente íntegros daqueles que exibem sinais de degeneração. Tal como destacado por Krzizanowski e Carvalho (2024), características como retração do corpo celular, condensação de cromatina, rompimento de membranas, vacuolização e aumento do espaço pericelular configuram marcadores consistentes de comprometimento morfológico. Essa diferenciação não apenas permite compreender o impacto das alterações celulares sobre o tecido nervoso como também estabelece bases sólidas para abordagens computacionais de análise neurobiológica. Em especial, a identificação automática desses padrões por meio de modelos de aprendizado profundo pode auxiliar na quantificação objetiva de danos neuronais, contribuindo para estudos diagnósticos, prognósticos e de pesquisa experimental — perspectiva que alinha diretamente com a proposta deste trabalho.

## REDES NEURAIS CONVOLUCIONAIS

As CNNs são modelos de aprendizado profundo projetados especificamente para analisar imagens, extraindo automaticamente padrões visuais. De acordo com LeCun *et al*. (1998), essas redes foram inspiradas no funcionamento do córtex visual humano, onde neurônios especializados respondem a pequenas estruturas como linhas e curvas, que posteriormente são combinadas para formar representações mais complexas. Os autores afirmam que as redes convolucionais foram inspiradas pela organização do córtex visual animal, demonstrando a inspiração biológica do método.

Segundo Rosa (2018), uma CNN consiste em múltiplas camadas com funções diferentes. É comum inicialmente aplicar sobre o dado de entradas as camadas que dão o nome à rede neural artificial, chamadas de convolução. Uma camada de convolução é composta por diversos neurônios, cada uma com o trabalho de aplicar um filtro em uma região da imagem. Cada neurônio é conectado a um conjunto de *pixels* da camada anterior e a cada uma dessas conexões é atribuído um peso. A combinação das entradas de um neurônio, utilizando os pesos respectivos de cada uma de suas conexões, produz um resultado que será passado para a camada seguinte. O filtro de convolução é representado por uma matriz que contém os pesos atribuídos as conexões de um neurônio. As camadas são organizadas de forma a detectar padrões mais simples primeiro (linhas, curvas etc.) e padrões mais complexos (faces, objetos etc.) (Mishra, 2020). A Figura 4 apresenta a arquitetura básica de uma rede neural convolucional.

Figura – Arquitetura de uma Rede Neural Convolucional

Diagrama, Esquemático

Descrição gerada automaticamente

Fonte: adaptado de Soares e Carmo(2020).

Outra parte fundamental da arquitetura das CNNs são as camadas de *pooling*, responsáveis por reduzir a dimensão espacial dos mapas de características. Isso torna o processo mais eficiente e ajuda o modelo a se tornar menos sensível a pequenas variações na posição dos objetos. Conforme descrito em Goodfellow *et al.* (2016), o *max pooling* é o método mais comum. O *max-pooling* é uma camada de *pooling* que computa o máximo local de uma determinada região do mapa de atributos, eliminando valores não máximos conforme pode-se observar na Figura 5. Ribeiro (2020) destaca que além de reduzir o tamanho da imagem e o processamento para a próxima camada, esta técnica auxilia no tratamento de invariâncias locais.

Figura – Aplicação de *max-pooling* 2x2 em uma imagem 4x4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |
|  | 1 | 12 | 10 | 5 |  |  |  |  |  |
|  | 8 | 6 | 3 | 21 |  |  | 12 | 21 |  |
|  | 7 | 0 | 5 | 9 |  |  | 7 | 9 |  |
|  | 4 | 2 | 4 | 0 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Fonte: adaptado de Ribeiro (2020).

Segundo Ribeiro (2020), em níveis mais altos a saída das camadas convolucionais e *pooling* representam características da imagem de entrada, e estas são passadas para as camadas totalmente conectadas para a classificação da imagem de acordo com as classes e conjuntos de treinamentos. As camadas totalmente conectadas possuem conexões com todos os neurônios da camada anterior e, estão ligadas a todos os neurônios da camada seguinte.

Diversas arquiteturas de CNN influenciaram o estado da arte da visão computacional. A VGG16, proposta por Simonyan e Zisserman (2015), utiliza uma sequência de convoluções 3×3 que simplifica a estrutura da rede. A família Inception (Szegedy *et al*., 2015) apresentou módulos capazes de combinar filtros de diferentes tamanhos dentro de uma mesma camada, permitindo capturar padrões em múltiplas escalas. Já a ResNet, proposta por He *et al*. (2016), introduziu as *skip connections*, que ajudam a evitar a degradação do desempenho em redes profundas. Na introdução do artigo, os autores afirmam que redes mais profundas são mais difíceis de treinar e propõem as conexões residuais como solução para esse problema (He *et al*., 2016, p. 2).

Essas arquiteturas consolidaram as CNNs como base de inúmeros modelos modernos de classificação, detecção e segmentação. Devido à sua capacidade de extrair padrões sutis, elas são amplamente empregadas em aplicações biomédicas, incluindo a identificação de estruturas celulares em imagens microscópicas, como demonstram Angonezi (2022) e Santos *et al*. (2022) ao utilizarem CNNs para identificação de alterações morfológicas em tecidos biológicos.

## YOLO

A família de modelos You Only Look Once (YOLO) representa uma das abordagens mais eficientes para detecção de objetos em tempo real. Diferentemente de métodos tradicionais, como o Faster R-CNN, que dividem o processo em duas etapas — geração de regiões propostas e posterior classificação — o YOLO realiza toda a detecção em uma única passagem pela rede. Conforme apresentado por Redmon *et al.* (2016), essa abordagem transforma a detecção em um problema de regressão direta, no qual a imagem é dividida em uma grade e cada célula dessa grade prevê as classes presentes e as caixas delimitadoras correspondentes.

Essa unificação permite que a rede analise a imagem como um todo, evitando redundâncias presentes em modelos de duas etapas. Redmon *et al.* (2016) explicam que essa visão global reduz erros associados ao fundo da cena, além de gerar resultados muito mais rápidos que técnicas anteriores. Para muitas aplicações práticas, como vigilância, robótica, microscopia digital e sistemas embarcados, essa velocidade é determinante.

Com o tempo, versões mais avançadas da arquitetura foram sendo desenvolvidas. A versão YOLOv3 (Redmon; Farhadi, 2018) introduziu o *backbone* Darknet-53, combinando convoluções e conexões residuais. O YOLOv4 (Bochkovskiy *et al*., 2020) incorporou uma série de aperfeiçoamentos conhecidos como *bag of freebies* e *bag of specials*, que aumentaram significativamente a precisão sem comprometer a velocidade. Mais recentemente, o YOLOv7 (Wang *et al*., 2022) otimizou o processo de treinamento, criando um conjunto de aprimoramentos que elevaram a precisão em tarefas de detecção densa.

Baseado em aprendizado profundo e visão computacional, possui uma precisão considerável. O YOLOv8 é altamente adaptável, permitindo uma implementação eficiente em *hardware* avançado e ambientes de nuvem, mantendo custos sob controle. Sua excepcional precisão é respaldada por avaliações em referências da indústria, como o conjunto de dados *Common Objects in Context* (COCO) e Roboflow, estabelecendo um novo padrão para modelos de detecção de objetos (Jocher *et al*., 2023).

O detector de objetos YOLOv8 faz uso de uma rede neural convolucional de ponta a ponta, ou seja, não existem camadas densas na arquitetura. Isso na prática permite que sejam utilizadas imagens de diferentes resoluções, mas com dimensões iguais, o que ajuda o modelo a ter uma maior precisão na detecção de objetos menores. A Figura 6 apresenta a arquitetura do YOLOv8.

Figura – Arquitetura YOLOv8

Diagrama

Descrição gerada automaticamente

Fonte: Solawetz *et al*. (2023).

A arquitetura YOLO inicialmente recebe uma imagem de entrada, esta é processada através de uma rede neural convolucional chamada de *backbone*. O *head* é a segunda parte da arquitetura, construída a partir das camadas desenvolvidas pelo *backbone,* quedesempenha um papel fundamental para a extração de características juntamente com a previsão realizada na inferência exercida durante as tarefas de detecção de objetos. Em casos de classificação de imagens, a saída final da rede é suficiente para fazer uma previsão. Em contrapartida, na detecção de objetos não somente é necessário identificar a classe do objeto, como também a área exata na imagem que ele ocupa, denominada *bouding-box* (Bochkovskiy *et al.* 2020). Em síntese, é necessário sortear cuidadosamente as camadas de recursos do *backbone*, isto acontece na parte do algoritmo denominada *neck*. Além disso, os algoritmos de detecção de objetos podem ser categorizados em dois tipos: *one-stage detectors* e *two-stage detectors*. *Two-stage detectors* realizam a localização e classificação dos objetos separadamente para cada caixa delimitadora. Em contrapartida *one-stage detectors* realizam essas duas tarefas simultaneamente. Assim, somente uma execução na rede se demonstra necessária para realizar a detecção dos objetos na imagem. A arquitetura YOLO é um exemplo de *one-stage detectors* (Pham *et al.* 2020).

De acordo com Bochkovskiy *et al.* (2020), a YOLO *head* é segunda parte da arquitetura, sendo construída a partir das camadas desenvolvidas pelo *backbone,* esta desempenha um papel fundamental para a extração de características juntamente com a previsão realizada na inferência exercida durante as tarefas de detecção de objetos. Esta parte da arquitetura foi estruturada na versão YOLOv3 e então consolidada na versão YOLOv4, e por conseguinte, permanecendo até a versão atual. Esta estrutura é utilizada inicialmente para a detecção dos objetos desejados, incorporando etapas de detecção baseadas em âncoras e três níveis de granularidade. Redmon *et al*. (2018) demonstra que as detecções de objetos baseadas em âncoras se caracterizam por ser um método de detecção de objetos o qual utiliza *deep learning* e *bounding-boxes* pré-definidas (denominadas âncoras), como propostas para realizar a detecção. A ideia por traz de uma detecção de objetos utilizando âncoras consiste em referenciar e prever estas *bounding-boxes*. Portanto, cada objeto pode possuir um *label* caracterizando a ancora em questão, e cada imagem pode ter ou não a presença destes, sendo que podem existir classes distintas de *labels/*âncoras*.*

A eficiência e a velocidade do YOLO fazem com que ele seja especialmente adequado a contextos nos quais a detecção precisa ocorrer rapidamente, mesmo em sistemas de hardware limitado. Trabalhos como o de Krzizanowski e Carvalho (2024) demonstram, por exemplo, o uso de smartphones acoplados a microscópios para captura e detecção de estruturas biológicas — um cenário onde modelos em tempo real como YOLO têm forte aplicabilidade. A capacidade de localizar objetos pequenos e distribuídos espacialmente, como neurônios em imagens microscópicas, reforça sua adequação para problemas semelhantes ao deste trabalho.

## TRABALHOS CORRELATOS

Para realizar a busca por trabalhos correlatos, utilizou-se a ferramenta Google Acadêmico para encontrar artigos relacionados ao tema, abrangendo o período de 2018 a 2024, visando priorizar pesquisas recentes na área. Conforme demonstra o Quadro 1, as *strings* de busca incluíram termos como “células”, “classificação”, “aprendizado de máquina”, “neurônios”, “*deep learning*”, “microscopia”, entre outros termos relevantes para o contexto da pesquisa. Todos os termos foram também traduzidos para o inglês, sendo as buscas replicadas com os mesmos conjuntos de palavras-chave nos portais mencionados, como pode ser observado no Quadro 2. Considera-se que esta abordagem garante uma ampla cobertura dos trabalhos disponíveis, identificando os que empregam diversas técnicas de aprendizado de máquina para a identificação e classificação de células, com foco especial em neurônios.

Quadro – Resultado de buscas por termos contidos em artigos em língua portuguesa

|  |  |
| --- | --- |
| **Termos de busca** | **Google Acadêmico** |
| “células anormais” AND “redes neurais” | 65 |
| “redes neurais convolucionais” AND “imagens de microscópio” AND “neurônio” | 13 |
| “aprendizado de máquina” AND “células” AND “microscopia” | 282 |
| “aprendizado de máquina” AND “células” AND “classificação” | 3.760 |
| “classificação de neurônios” AND “aprendizado de máquina” | 1 |
| “neurônios atípicos” AND “deep learning” | 0 |
| “redes neurais” AND “classificação de tipo de célula” | 0 |
| “deep learning” AND “atípico” AND “células” | 195 |
| “redes neurais” AND “classificação de tipo de célula” AND “cérebro” | 0 |
| **Total** | **4.316** |

Fonte: elaborado pelo autor.

Quadro – Resultado de buscas por termos contidos em artigos em língua inglesa

|  |  |
| --- | --- |
| **Termos de busca** | **Google Acadêmico** |
| “abnormal cells” AND “neural networks” | 5.530 |
| “convolutional neural networks” AND “microscope images” AND “neuron” | 829 |
| “machine learning” AND “cells” AND “microscopy” | 52.300 |
| “machine learning” AND “cells” AND “classification” | 351.000 |
| “neuron classification” AND “machine learning” | 384 |
| “atypical neurons” AND “deep learning” | 3 |
| “neural networks” AND “cell-type classification” | 1.170 |
| “deep learning” AND “atypical" AND “cells” | 16.200 |
| “neural networks” AND “cell-type classification” AND “brain” | 1.080 |
| **Total** | **428.496** |

Fonte: elaborado pelo autor.

Inicialmente, realizou-se uma triagem exploratória com o objetivo de reduzir o número de artigos encontrados nas buscas iniciais, que ultrapassavam centenas de milhares, para um conjunto mais gerenciável e relevante ao escopo deste estudo. Foram considerados critérios como a quantidade de citações, a presença de termos relacionados à classificação celular e a aplicação de técnicas de aprendizado de máquina em dados biológicos, especialmente em imagens de microscopia. Durante a análise dos títulos e resumos, constatou-se que muitos trabalhos não estavam diretamente ligados à proposta deste estudo, abordando outras áreas da biomedicina, como diagnóstico de câncer, detecção de doenças infecciosas, entre outros. Como o número de estudos voltados especificamente à classificação de neurônios típicos e atípicos era bastante restrito, optou-se por incluir também trabalhos que tratassem da classificação de células em geral ou da contagem automática de neurônios e células em imagens microscópicas.

Foram priorizados os artigos que descreveram técnicas computacionais detalhadas, CNNs e modelos supervisionados, além de métodos comparativos aplicados à segmentação ou classificação celular. Dessa forma, os trabalhos selecionados representam abordagens complementares que contribuem para o entendimento e fundamentação metodológica do presente estudo, mesmo quando aplicados a tipos celulares distintos dos neurônios. Dessa forma, o Quadro 3 apresenta os trabalhos selecionados que seguiram os critérios mencionados.

Quadro – Síntese dos trabalhos correlatos selecionados

|  |  |  |
| --- | --- | --- |
| Assunto | Filtro | Referência |
| Artificial Neural Networks in Action for an Automated Cell-Type Classification of Biological Neural Networks | “neural networks” AND “cell-type classification” | Troullinou *et al.* (2021) |
| Deep Learning Models for Atypical Serotonergic Cells Recognition | “deep learning” AND “atypical” AND “cells” | Corradetti, Bernardi e Corradetti (2024) |
| Uso de visão computacional para contagem automática de células em imagens obtidas por microscópios | “redes neurais convolucionais” AND “imagens de microscópio” AND “neurônio” | Secretário e Pires (2018) |
| Aprendizado de máquina aplicado à detecção, classificação e análise de células tumorais em imagens de microscopia de luz visível | “aprendizado de máquina” AND “células” AND “classificação” | Angonezi (2022) |
| Algoritmos de Aprendizado de Máquina para Classificação de Células Nucleadas do Sangue Periférico – Uma Experiência do Projeto Hemovision | “aprendizado de máquina” AND “células” AND “microscopia” | Santos *et al*. (2022) |

Fonte: elaborado pelo autor.

Conforme apresentado no Quadro 3, os trabalhos correlatos analisados neste estudo abordam diferentes estratégias para a classificação de células, com foco na aplicação de técnicas de aprendizado de máquina em imagens de microscopia ou dados biológicos relacionados à morfologia ou atividade celular. Devido à escassez de publicações especificamente voltadas à classificação de neurônios típicos e atípicos, foram incluídos também estudos que tratam da classificação ou detecção de outros tipos celulares, desde que compartilhem aspectos metodológicos e/ou técnicos com a proposta deste trabalho. A seleção priorizou artigos que aplicam modelos de aprendizado profundo e que apresentam resultados quantitativos relevantes para análise comparativa de desempenho entre diferentes arquiteturas. Os quadros Quadro 4, Quadro 5 e Quadro 6 exibem os artigos selecionados e seus detalhamentos.

Quadro 4 – Artificial Neural Networks in Action for an Automated Cell-Type Classification of Biological Neural Networks

|  |  |
| --- | --- |
| Referência | Troullinou *et al*. (2021) |
| Objetivos | Desenvolver e comparar arquiteturas de redes neurais profundas para a classificação automática de quatro tipos celulares neuronais, descritos a seguir: células piramidais, interneurônios positivos para parvalbumina (PV), interneurônios positivos para somatostatina (SOM) e interneurônios positivos para peptídeo intestinal vasoativo (VIP) com base em sinais de atividade de cálcio. |
| Principais funcionalidades | Classificação automática de tipos neuronais a partir de dados funcionais obtidos de redes biológicas. Eliminação da necessidade de extração manual de características. Avaliação comparativa entre arquiteturas. |
| Ferramentas de desenvolvimento | Arquiteturas CNN, RNN e LSTM implementadas com TensorFlow e Keras. Utilização de séries temporais de fluorescência de cálcio como entrada. Aplicação de normalização e aumento de dados sintéticos. |
| Resultados e conclusões | O modelo baseado em CNN apresentou o melhor desempenho, atingindo 88,67% de acurácia, com robustez frente a dados de sessões distintas. O estudo demonstrou que diferentes tipos neuronais podem ser classificados com alta precisão a partir de dados funcionais brutos, sem necessidade de marcação morfológica. A abordagem mostra potencial de uso em ambientes experimentais com redes neuronais biológicas reais. |

Fonte: elaborado pelo autor.

Quadro – Aprendizado de máquina aplicado à detecção, classificação e análise de células tumorais em imagens de microscopia de luz visível

|  |  |
| --- | --- |
| Referência | Angonezi (2022) |
| Objetivos | Construir um modelo de rede neural convolucional para classificar células tumorais com base em características morfológicas extraídas automaticamente de imagens de microscopia óptica. |
| Principais funcionalidades | Reconhecimento automatizado de células com base em padrões morfológicos extraídos de imagens, com foco em contagem, localização e classificação entre mitótica e não mitótica. |
| Ferramentas de desenvolvimento | Avaliação de oito modelos diferentes (CSL, DCL, R3Det, RetinaNet, RSDet, Refine RetinaNet, R3Det-DCL e R2CNN. O processo envolveu técnicas de data augmentation, como rotação, espelhamento e variações de resolução, além do redimensionamento das imagens. Também foram aplicados ajustes nas funções de perda dos modelos, especialmente para lidar com a regressão de ângulos, utilizando codificações específicas que favorecem a detecção precisa da orientação celular. |
| Resultados e conclusões | Todos os modelos apresentaram erros médios inferiores a 10% para contagem celular (MAPE), erro médio de posição abaixo de 1% (RMSE) e taxa de erro de classificação celular (mitótica vs. não mitótica) inferior a 8%. |

Fonte: elaborado pelo autor.

Quadro – Algoritmos de Aprendizado de Máquina para Classificação de Células Nucleadas do Sangue Periférico – Uma Experiência do Projeto Hemovision

|  |  |
| --- | --- |
| Referência | Santos *et al*. (2022) |
| Objetivos | Aplicar CNNs combinadas com SVM para classificar diferentes tipos de células sanguíneas em imagens de microscopia óptica. |
| Principais funcionalidades | Classificação automática de células (leucócitos, plaquetas, eritrócitos nucleados). Redução de resolução e análise do impacto em ambientes com baixa capacidade computacional. |
| Ferramentas de desenvolvimento | CNN ResNet18 utilizada para extração automática de características das imagens, substituindo as camadas densas tradicionais por um classificador SVM configurado via Grid Search com abordagem One-vs-Rest para múltiplas classes. O modelo aplicou data augmentation para ampliar a diversidade do conjunto de treinamento e redução de resolução (de 360×363 para 64×64 pixels) para simular ambientes com restrições computacionais, como o uso de câmeras de celular em laboratórios com poucos recursos. |
| Resultados e conclusões | O modelo alcançou acurácia média de 97,2% e F1-score de 97%. Os granulócitos imaturos, por apresentarem variações morfológicas internas, impactaram negativamente a classificação de monócitos e basófilos. Ainda assim, os altos índices de acerto e a robustez frente à redução de resolução indicam potencial para uso em ambientes educacionais e ao diagnóstico clínico. |

Fonte: elaborado pelo autor.

## MODELO ATUAL

O trabalho desenvolvido por Krzizanowski e Carvalho (2024) teve como principal objetivo automatizar a contagem e identificação de neurônios atípicos em imagens de microscopia óptica, utilizando técnicas de Visão Computacional e Machine Learning (ML). Para isso, adotou-se como modelo a arquitetura Faster R-CNN, amplamente utilizada para detecção de objetos em imagens, especialmente por sua eficiência e precisão em tarefas que envolvem localização e classificação simultânea. A proposta surgiu como uma alternativa promissora ao processo manual, que demanda tempo, conhecimento técnico especializado e está sujeito a variações subjetivas entre analistas humanos.

A base de dados utilizada no projeto foi composta por 595 imagens obtidas por meio de captura com câmeras de celulares acopladas a microscópios ópticos. Essas imagens foram previamente anotadas utilizando a plataforma LabelBox, permitindo a identificação manual dos neurônios atípicos a serem utilizados como referência para o treinamento supervisionado. O conjunto de dados foi dividido em 80% para treinamento e 20% para validação, sendo complementado por técnicas de *Data Augmentation*, como rotação, espelhamento e variação de tonalidade, para aumentar a robustez do modelo frente às variações naturais das imagens.

O treinamento da Faster R-CNN foi realizado na plataforma Google Colab, com implementação em PyTorch. O modelo foi treinado por 10 épocas, utilizando métricas como precisão, revocação e F1-score para avaliar seu desempenho. Os resultados obtidos demonstraram uma média de acerto superior a 90% nas imagens de teste, com desempenho promissor em muitos casos, embora ainda com presença de falsos positivos e limitações atribuídas ao tamanho reduzido do *dataset* e à complexidade morfológica dos neurônios.

A interface da ferramenta foi concebida como um aplicativo móvel desenvolvido em Flutter, com *backend* em Python (utilizando Flask) e banco de dados Firebase (Figura 7). O aplicativo permite ao usuário realizar cadastro, upload de imagens e visualização dos resultados da análise, com as detecções sendo sobrepostas às imagens enviadas. A ferramenta ainda oferece funcionalidades de repositório de análises e histórico de exames, sendo testada por estudantes da área da saúde, que contribuíram com *feedbacks* positivos e sugestões de melhoria para usabilidade.

Figura – Telas de listagem de informações do aplicativo

Screenshots of a screenshot of a cell phone

Description automatically generated

Fonte: Krzizanowski e Carvalho (2024).

Apesar dos avanços obtidos, Krzizanowski e Carvalho (2024) não realizaram comparações com outras arquiteturas de CNNs, nem abordaram ajustes mais avançados de hiperparâmetros, que são variáveis definidas antes do processo de treinamento e controlam o comportamento do modelo, como a taxa de aprendizado e o número de camadas, e influenciam diretamente sua performance e capacidade de generalização (Probst; Boulsteix; Bischl, 2019). Além disso, os testes com usuários foram limitados, e não houve a implementação de funcionalidades específicas para treinamento supervisionado contínuo ou refinamento da base de dados. A presente proposta de trabalho parte deste ponto, visando justamente expandir a análise por meio da comparação com novas arquiteturas de CNNs e aprofundar a investigação sobre a aplicabilidade de modelos mais modernos e precisos para a classificação automática de neurônios atípicos.

# DESENVOLVIMENTO

Esse capítulo está dividido em 3 seções. A seção 3.1 trata sobre os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) do modelo de classificação. A seção 3.2 detalha o modelo. Depois, a seção 3.3 apresenta o processo de busca e preparação do *dataset*. Após isso, a seção 3.4 apresenta o funcionamento da configuração dos experimentos. A seção 3.5 explica o fluxo principal de execução do experimento, e a seção 3.6 descreve as configurações de experimentos abordadas durante o estudo. Por fim, a seção 3.7 detalha os resultados do estudo.

## requisitos

Inicialmente foram levantados os requisitos para o desenvolvimento do modelo de aprendizado de máquina. O Quadro 7 apresenta os RFs que servem como base para o desenvolvimento do modelo proposto, descrevendo as capacidades mínimas necessárias para garantir o seu funcionamento. Esses requisitos contemplam desde o carregamento e a preparação das imagens — incluindo normalização, redimensionamento, *tiling* e técnicas de *data augmentation* — até a possibilidade de configurar de forma flexível diferentes parâmetros do experimento, sem a necessidade de modificar o código-fonte. Também são descritas as funcionalidades essenciais relacionadas à etapa de detecção, como a utilização do modelo YOLOv8 aplicado diretamente nas imagens originais ou nos *tiles* gerados. Por fim, inclui-se a necessidade de registrar e armazenar os resultados obtidos, produzindo artefatos como *bounding boxes*, imagens anotadas e métricas quantitativas, permitindo análises mais completas.

Quadro – Requisitos Funcionais do modelo

|  |
| --- |
| RF01 – Carregar as imagens e suas anotações a partir dos arquivos de data.ndjson exportado da plataforma LabelBox |
| RF02 – Realizar o pré-processamento das imagens, incluindo normalização, redimensionamento, *tiling* e técnicas de *data augmentation* |
| RF03 – Permitir a configuração customizada dos experimentos, possibilitando ajustar parâmetros como tamanho do *tile, overlap, tresholds* e diretórios sem necessidade de alterar o código-fonte |
| RF04 – Identificar neurônios atípicos utilizando o modelo YOLOv8, aplicando a detecção sobre cada imagem ou *tile* |
| RF05 – Gerar e salvar os resultados da análise, incluindo *bounding boxes*, imagens anotadas, mosaicos, tiles processados e métricas de desempenho como precisão, recall e F1-score |

Fonte: elaborado pelo autor.

O Quadro 8 apresenta os RNFs considerados no desenvolvimento do sistema, os quais complementam os aspectos técnicos e garantem sua qualidade operacional. Esses requisitos enfatizam o uso de ferramentas abertas, como a linguagem Python e bibliotecas como PyTorch e OpenCV, assegurando a facilidade de reprodução dos experimentos. Além disso, também incluem a estruturação do código-fonte de forma modular. Também incluem a organização automática da estrutura de diretórios, permitindo rastrear de forma consistente imagens originais, *tiles*, anotações, modelos, *logs* e resultados, o que é essencial para auditorias experimentais. Outro ponto relevante diz respeito ao desempenho: o processamento deve ocorrer de maneira eficiente, possibilitando a execução mesmo em máquinas convencionais, sem dependência de GPUs de alto desempenho. Por fim, estabelece-se a necessidade de disponibilizar visualizações claras e interpretáveis — como gráficos, imagens anotadas e relatórios — que auxiliem tanto na depuração quanto na apresentação e validação dos resultados obtidos ao longo dos experimentos.

Quadro – Requisitos Não Funcionais do modelo

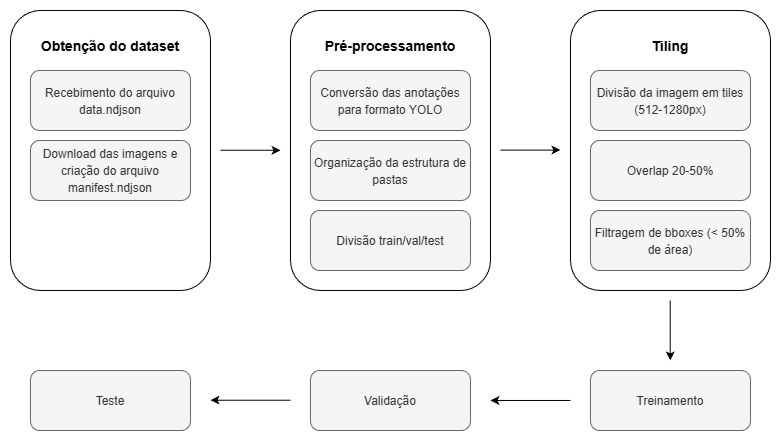
|  |
| --- |
| RNF01 – Utilizar ferramentas de código aberto, como Python e as bibliotecas PyTorch e OpenCV, permitindo que todo o processo de desenvolvimento e teste possa ser acompanhado, repetido e aprimorado |
| RNF02 – Estruturar o código de forma modular, separando claramente as etapas de preparação dos dados, treinamento, avaliação e geração de artefatos de *debug*, facilitando manutenção e reutilização |
| RNF03 – Organizar automaticamente toda a estrutura de diretórios, garantindo rastreabilidade entre imagens originais, *tiles*, anotações, modelos, *logs* e resultados |
| RNF04 – Manter desempenho adequado, processando imagens de forma eficiente, permitindo o uso em *hardware* doméstico sem GPU de alta performance |
| RNF05 – Oferecer visualizações claras e interpretáveis, utilizadas tanto para depuração quanto para apresentação dos resultados, facilitando a validação da detecção |

Fonte: elaborado pelo autor.

## DESCRIÇÃO DO MODELO

O diagrama de atividades, apresentado na Figura 8, representa as etapas realizadas para o desenvolvimento do modelo.

Figura – Etapas para detecção de neurônios atípicos



Fonte: elaborado pelo autor.

Inicialmente, obteve-se o *dataset* utilizado para o estudo, por meio do recebimento do arquivo data.ndjson contendo as URLs das imagens e as anotações fornecidas pela plataforma LabelBox. A partir desse arquivo, efetuou-se o *download* automático das imagens e a geração do arquivo manifest.ndjson, com o objetivo de organizar e padronizar os dados necessários para o *pipeline*. Em seguida, iniciou-se a etapa de pré-processamento, na qual as anotações foram convertidas para o formato YOLO, e a estrutura completa do diretório do *dataset* foi organizada de acordo com o padrão recomendado pelo modelo. Nessa fase, também se efetuou a divisão das imagens em subconjuntos de treino, validação e teste, preservando a integridade das distribuições originais.

Posteriormente, aplicou-se a etapa de *tiling*, dividindo cada imagem em *tiles* menores, variando entre 512 e 1.280 *pixels*, de modo a facilitar a detecção de objetos extremamente pequenos, como os neurônios atípicos. Essa divisão utilizou um *overlap* entre 20% e 50%, assegurando que os neurônios não fossem cortados de forma prejudicial ao treinamento. *Tiles* que possuíam menos de 50% da área do *bounding box* original foram descartados, garantindo a consistência das informações morfológicas. Posteriormente, realizou-se o treinamento do modelo YOLOv8 utilizando os *tiles* gerados, seguido pelo processo de validação, que permitiu ajustar hiperparâmetros e analisar o comportamento do modelo ao longo das épocas. Por fim, foram realizados testes com o conjunto reservado de imagens, permitindo avaliar a eficácia final da arquitetura proposta e comparar seus resultados com os obtidos no estudo original.

## BUSCA E PREPARAÇÃO DA BASE DE DADOS

O estudo atual utilizou a mesma base de imagens construída no trabalho anterior, desenvolvido por Krzizanowski e Carvalho (2024), na qual todo o processo de obtenção e anotação das imagens foi conduzido com apoio da equipe do curso de Medicina da FURB. As imagens foram capturadas ao longo de aproximadamente seis meses (maio a outubro de 2024), totalizando 595 registros obtidos diretamente das lâminas analisadas em microscópio. Esse conjunto inicial incluiu tanto as imagens originais quanto suas anotações manuais, que marcaram visualmente os neurônios avaliados como atípicos, servindo de referência para os experimentos posteriores. A Figura 9 ilustra o processo de captura das imagens.

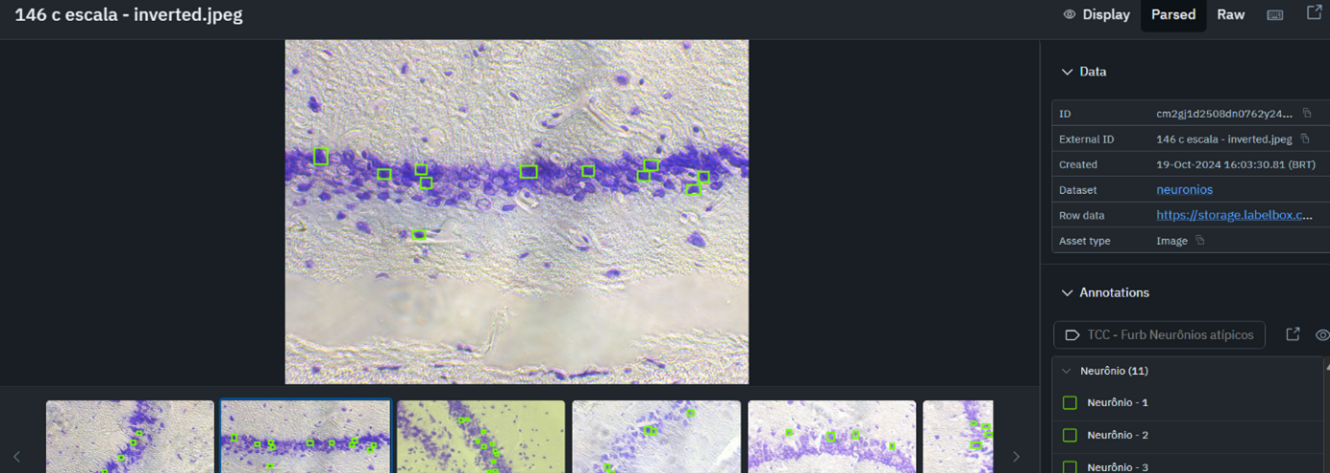
Figura – Processo de captura das imagens



Fonte: Krzizanowski e Carvalho (2024).

Para organização e padronização do *dataset*, as imagens foram importadas para a plataforma LabelBox, onde foi criado o projeto TCC – Furb Neurônios Atípicos e um *dataset* dedicado denominado neurônios. No estudo anterior, todas as imagens passaram por um processo de rotulagem, no qual os alunos de Medicina identificaram e demarcaram os neurônios atípicos utilizando caixas delimitadoras (*bounding boxes*). Cada imagem teve seu respectivo rótulo salvo com o mesmo nome, formando pares entre a imagem original e sua segmentação, o que facilitou a catalogação e permitiu a criação de um conjunto de dados padronizado para análises computacionais. A Figura 10 mostra um exemplo de anotação feita através da ferramenta LabelBox.

Figura – Exemplo do resultado das anotações na ferramenta LabelBox



Fonte: Krzizanowski e Carvalho (2024).

Com o *dataset* completamente anotado na plataforma, foi realizada a exportação no formato Data Export v2, que gera um arquivo data.ndjson contendo, em cada linha, um objeto JSON com as informações estruturadas de cada imagem. Esse arquivo reúne metadados essenciais — como dimensões da imagem, identificador único, URL para *download* e todas as coordenadas das *bounding boxes* — viabilizando a reconstrução integral de cada amostra. Neste trabalho, esse arquivo serviu como ponto de partida para a construção da base final utilizada no modelo, permitindo automatizar o *download* das imagens, gerar rótulos no formato YOLO e preparar toda a estrutura necessária para o desenvolvimento e avaliação dos modelos de detecção. A Figura 11 ilustra o arquivo data.ndjson, com um objeto JSON em cada linha.

Figura – Visualização do arquivo ndjson

Texto

Descrição gerada automaticamente

Fonte: Krzizanowski e Carvalho (2024).

Após a obtenção do arquivo data.ndjson, que foi fornecido pelos autores do estudo anterior, desenvolveu-se um *script* na linguagem Python, chamado “prepare.py”, que realiza a etapa de *download* das imagens. O *script* percorre o arquivo data.ndjson linha a linha, extrai os metadados de cada imagem e faz o *download* do arquivo original a partir da URL fornecida pelo LabelBox. Cada imagem é salva localmente utilizando o identificador único, chamado data\_row\_id, como nome de arquivo e a extensão é inferida a partir do mime\_type ou da própria URL. O Quadro 9 apresenta a implementação inicial do *script* mencionado.

Quadro – Download das imagens e construção do arquivo manifest.ndjson

|  |
| --- |
| def main():  itens = list(iterar\_ndjson(NDJSON\_PATH))  with open(MANIFEST\_PATH, "a", encoding="utf-8") as manifest:  for exemplo in itens:  data\_row = exemplo.get("data\_row") or {}  url = data\_row.get("row\_data")  row\_id = data\_row.get("id")  extensao = inferir\_extensao(exemplo, url)  destino = montar\_caminho\_destino(exemplo, extensao)  if not destino.exists():  baixar\_imagem(url, destino)  caixas = extrair\_bounding\_boxes(exemplo)  registro = {  "data\_row\_id": row\_id,  "imagem": destino.as\_posix(),  "url": url,  "width": exemplo.get("media\_attributes", {}).get("width"),  "height": exemplo.get("media\_attributes", {}).get("height"),  "annotations": caixas,  }  manifest.write(json.dumps(registro, ensure\_ascii=False) + "\n") |

Fonte: elaborado pelo autor.

O trecho apresentado no Quadro 9 corresponde à função principal responsável por executar todo o fluxo de preparação do *dataset*. Inicialmente, o *script* carrega para a memória um conjunto contendo os identificadores (data\_row\_id) já registrados anteriormente no arquivo manifest.ndjson, por meio da função load\_already\_in\_manifest. Isso permite que a execução do *script* seja retomada de forma segura, evitando *downloads* duplicados e reprocessamento desnecessário. Em seguida, o arquivo data.ndjson é lido pela função iter\_ndjson, produzindo uma lista de exemplos provenientes da plataforma LabelBox. O *script* também imprime a quantidade de itens encontrados, permitindo verificar se o *dataset* foi carregado corretamente.

A etapa seguinte consiste no processamento individual de cada exemplo presente no arquivo ndjson. Para cada item, o *script* extrai o identificador da imagem e sua respectiva URL a partir do campo data\_row. Caso a imagem já tenha sido processada anteriormente — isto é, se o data\_row\_id estiver no conjunto de registros existentes — o *script* ignora aquele item. Para os itens que ainda não foram processados, o *script* determina a extensão correta do arquivo por meio da função auxiliar get\_extension, que utiliza preferencialmente o tipo MIME declarado no *dataset* e, caso este não esteja disponível, infere a extensão diretamente da URL. A seguir, o nome final da imagem é definido pela função make\_dest\_path, que utiliza o valor do data\_row.id como nome estável do arquivo, ou um *hash* como alternativa caso esse identificador não exista.

Após definir o caminho de destino, o *script* realiza o *download* da imagem utilizando a função download\_with\_retries. Somente após um *download* bem-sucedido a imagem é registrada no manifest. Na sequência, o *script* extrai as anotações de *bounding boxes* por meio da função extract\_bounding\_boxes, reduzindo a estrutura do LabelBox para um formato compacto e adequado ao *pipeline* de detecção de objetos. Por fim, todas as informações relevantes do item — como identificadores, dimensões da imagem, caminho salvo, URL de origem e anotações — são organizadas em um dicionário e gravadas como uma linha no arquivo manifest.ndjson. Assim, esse trecho implementa o processo de *download* e padronização do *dataset*, produzindo um manifest de resumo que será utilizado pelo restante do *pipeline*.

## CONFIGURAÇÃO DOS EXPERIMENTOS

Para permitir maior flexibilidade no teste de diferentes configurações durante o estudo, foi adicionada ao projeto a classe ConfiguracaoExperimento. Esta classe contém todas as configurações que o *script* permite modificar, de forma que não seja necessário alterar o restante do código-fonte. Estas opções incluem modelo, tamanho de imagem, ativação de *tiling*, número de épocas para o treinamento, parâmetros de *data augmentation* etc. Os Quadros Quadro 10, Quadro 11 e Quadro 12 apresentam a implementação da classe de configuração, que serve como base para o resto do *pipeline*.

Quadro – Início da classe ConfiguracaoExperimento

|  |
| --- |
| @dataclass  class ConfiguracaoExperimento:      """      Agrupa TODAS as configurações que definem um experimento.      Inclui:      - Identificação      - Lista de tamanhos de imagem (imgsz)      - Modelo base (padrão ou custom P2)      - Tiling      - Splits      - Treino (epochs, batch, LR, etc.)      - Data augmentation      - Avaliação (lista de thresholds)      - Debug (quantidade de imagens)      - Device (GPU/CPU)      """      # Identificação      nome\_experimento: str      descricao: str      # tamanhos de imagem a testar (ex.: [512], [640, 896], etc.)      lista\_tamanhos\_imagem: List[int]      # Modelo base      # - "yolov8s.pt", "yolov8m.pt", etc (modelos oficiais)      # - "custom\_p2" para criar YAML com P2      modelo\_base: str = "yolov8s.pt"      # Tiling      usar\_tiling: bool = True      sobreposicao\_tile: float = 0.20      proporcao\_minima\_corte: float = 0.50      tamanho\_minimo\_objeto\_px: int = 6      # Splits      proporcao\_treino: float = 0.7      proporcao\_validacao: float = 0.2      proporcao\_teste: float = 0.1 |

Fonte: elaborado pelo autor.

O Quadro 10 apresenta o início da definição da classe ConfiguracaoExperimento, responsável por agrupar de forma centralizada todos os parâmetros que caracterizam cada experimento realizado ao longo deste trabalho. Trata-se de uma dataclass em Python, utilizada para organizar as configurações de maneira estruturada e legível, evitando a necessidade de espalhar constantes e hiperparâmetros ao longo do código.

Na primeira parte da classe, exibida no Quadro 10, são definidos os campos relacionados à identificação do experimento (nome\_experimento e descricao), bem como a lista de tamanhos de imagem a serem testados (lista\_tamanhos\_imagem), permitindo registrar, por exemplo, configurações com 512, 640 ou 1.280 *pixels*. Em seguida, o parâmetro modelo\_base indica qual variante do YOLOv8 será utilizada (como yolov8s.pt ou yolov8m.pt), incluindo a opção de empregar um modelo customizado com camada P2. São também declarados os parâmetros ligados ao *tiling*, como o uso ou não dessa técnica (usar\_tiling), a sobreposição entre *tiles* (sobreposicao\_tile), a proporção mínima de corte dos objetos e o tamanho mínimo do neurônio em *pixels*, além das proporções de treino, validação e teste que definem a divisão do conjunto de dados. No Quadro 11, são apresentados os demais campos da classe, abrangendo os hiperparâmetros de treinamento, configurações de *data augmentation*, critérios de avaliação e parâmetros de depuração.

Quadro – Segunda parte da classe ConfiguracaoExperimento

|  |
| --- |
| # Treino  epochs\_fase1: int = 100  epochs\_fase2: int = 100  batch\_size: int = 8  learning\_rate: float = 0.002  weight\_decay: float = 0.05  momentum: float = 0.937  patience: int = 50  # Data augmentation  usar\_augmentation: bool = True  mosaic: float = 0.0  mixup: float = 0.0  copy\_paste: float = 0.0  degrees: float = 5.0  translate: float = 0.05  scale: float = 0.10  shear: float = 0.0  flipud: float = 0.20  fliplr: float = 0.50  hsv\_h: float = 0.015  hsv\_s: float = 0.30  hsv\_v: float = 0.20  # Razão de tiles negativos no TREINO  max\_negativos\_por\_positivo: float = 3.0  # Avaliação / inferência  lista\_confs\_avaliacao: Optional[List[float]] = None  iou\_nms\_avaliacao: float = 0.50  max\_det\_avaliacao: int = 1000  # Debug (quantidade de imagens para salvar)  max\_imagens\_debug\_treino: int = 30  max\_imagens\_debug\_validacao: int = 50  max\_imagens\_debug\_teste: int = 50 |

Fonte: elaborado pelo autor.

Dando continuidade à configuração do experimento, o Quadro 11 apresenta os hiperparâmetros relacionados ao processo de treinamento do modelo. O campo epochs\_fase1 define o número de épocas utilizadas na etapa principal de treinamento. Hiperparâmetros como batch\_size, learning\_rate, weight\_decay e momentum controlam diretamente o comportamento do otimizador durante o aprendizado, influenciando a estabilidade e a convergência do modelo. O parâmetro patience determina o limite de épocas sem melhora significativa para acionamento do early stopping, evitando *overfitting*. Essa estrutura centraliza todos os elementos essenciais do treino, permitindo que experimentos distintos sejam configurados de forma simples, e reprodutível.

Na sequência, o Quadro 11 apresenta os campos associados às transformações de *data augmentation*, úteis para aumentar a diversidade visual do *dataset* e promover a generalização do modelo. O parâmetro usar\_augmentation habilita ou desabilita essa etapa, enquanto os demais — como mosaic, mixup, copy\_paste, degrees, translate, scale, shear, flipud, fliplr e ajustes no espaço Hue, Saturation, Value (HSV) — controlam a intensidade de cada técnica aplicada. Também é especificado o parâmetro max\_negativos\_por\_positivo, utilizado para regular a proporção máxima de *tiles* negativos incluídos no treinamento, mantendo um balanço adequado entre amostras com e sem neurônios. Por fim, são declaradas variáveis relacionadas à avaliação, como a lista de valores de confiança a serem analisados (lista\_confs\_avaliacao), o limiar de Intersection over Union (IoU) utilizado no Non-Maximum Suppression (iou\_nms\_avaliacao) e o número máximo de detecções permitidas (max\_det\_avaliacao). O trecho final inclui ainda parâmetros de depuração, que determinam quantas imagens anotadas serão salvas durante as etapas de treino, validação e teste, auxiliando na inspeção visual do comportamento do modelo. No Quadro 12, são apresentados os valores padrão e os ajustes automáticos executados pela classe por meio do método \_\_post\_init\_\_.

Quadro – Terceira parte da classe ConfiguracaoExperimento

|  |
| --- |
| # Device (GPU 0 ou CPU)      device: str = "0"      def \_\_post\_init\_\_(self):          # Garante que os splits somam 1.0          soma = self.proporcao\_treino + self.proporcao\_validacao + self.proporcao\_teste          if abs(soma - 1.0) > 1e-6:              raise ValueError("As proporções de treino/val/test devem somar 1.0")          # Se não foi passada lista de confs de avaliação, cria padrão [0.01 .. 0.20]          if not self.lista\_confs\_avaliacao:              self.lista\_confs\_avaliacao = [i / 100.0 for i in range(1, 21)] |

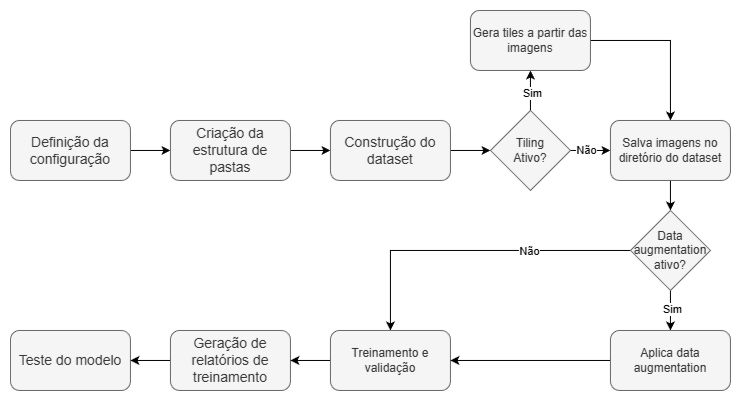
Fonte: elaborado pelo autor.

O Quadro 12 apresenta os parâmetros finais da configuração do experimento, juntamente com o método especial \_\_post\_init\_\_, responsável por executar verificações automáticas sempre que uma instância da classe é criada. O campo device permite definir se o treinamento será executado na GPU (especificada pelo índice, como "0") ou no processador, oferecendo flexibilidade para executar experimentos em diferentes ambientes computacionais. Já o método \_\_post\_init\_\_ é responsável por duas funções: inicialmente, verifica se as proporções destinadas aos conjuntos de treino, validação e teste somam exatamente 1.0, garantindo a integridade da divisão do *dataset*; em seguida, caso a lista de limiares de confiança para avaliação (lista\_confs\_avaliacao) não tenha sido explicitamente definida, o método gera automaticamente um conjunto padrão variando de 0.01 a 0.20. Dessa forma, a classe assegura que todas as configurações essenciais estejam corretas e completas.

## EXECUÇÃO do experimento

A etapa de experimentação foi a mais longa do processo, pois envolveu diversas combinações de parâmetros para treinamento do modelo, a fim de extrair os melhores resultados da rede. A maior parte dos testes foi realizada em uma máquina doméstica com uma placa gráfica RTX 3070 de 8Gb, 24Gb de memória RAM, processador Intel i5-10400f e 2Tb de armazenamento, porém, para execução de alguns experimentos foi utilizada a plataforma Google Colab, que dispõe de placas gráficas mais poderosas e quantidades maiores de memória RAM. Esta plataforma auxiliou para que fosse possível rodar combinações mais pesadas de parâmetros, que demorariam dias para finalizar, se executadas numa máquina doméstica. O *pipeline* de execução de experimentos é composto por diversas etapas sequenciais, a Figura 12 ilustra um diagrama com o processo macro do *pipeline*.

Figura – Diagrama fluxo do *pipeline* geral



Fonte: elaborado pelo autor.

O processo inicia pela configuração do experimento, onde são definidos os parâmetros para a execução, como tamanho de modelo, tamanho de imagem, *data augmentation* etc. No segundo momento, é criada toda a estrutura de pastas necessária e, após isso, realizada a construção do *dataset*, que considera a configuração de uso de *tiling* para salvamento das imagens. Depois, é realizado o treinamento e validação do modelo, etapa que já é integrada ao uso de *data augmentation*. Por último, são gerados os relatórios do treinamento e o teste, fazendo inferências em um conjunto de imagens ainda desconhecido pelo modelo. As seções a seguir descrevem o *pipeline* de forma mais detalhada.

### Criação da estrutura de pastas para o experimento

A primeira etapa para a execução de um experimento é a criação da estrutura de pastas, que comportará todos os arquivos gerados no processo. Cada etapa do processo — incluindo geração do *dataset*, treinamento, validação e demais processos — foi separada em uma subpasta específica. Essa organização padronizada permite comparar configurações de maneira clara, auxiliando na análise de resultados e na documentação geral do projeto. Um exemplo da estrutura final adotada é ilustrado no Quadro 13.

Quadro – Estrutura final de diretórios

|  |
| --- |
| nome\_do\_experimento/  imgsz\_640/  dataset/  images/  test/  train/  val/  labels/  test/  train/  val/  debug/  teste/  treino/  validacao/  logs/  metrics/  modelos/  fase1/  treino/  weights/  plots/ |

Fonte: elaborado pelo autor.

Conforme ilustrado no Quadro 13, é gerada uma estrutura de pastas completa para cada tamanho de imagem utilizado no experimento. A pasta dataset segue a estrutura padrão do modelo YOLO, separando uma pasta para as imagens, divididas em treino, validação e teste, e uma pasta para os labels, também em treino, validação e teste.

Também existe a pasta debug, onde são salvas imagens durante o processo de execução, que permite visualizar e comparar a anotação original e a predição do modelo. Adicionalmente foram criadas as pastas metrics e logs para armazenar arquivos com a sumarização dos resultados e *logs* de execução, respectivamente. Também foi criada a pasta plots, onde são gravados os arquivos de gráficos com métricas após o fim do experimento. Por fim, a pasta modelos, onde são salvos os arquivos registrados pelo próprio modelo YOLO, incluindo resultados, pesos do modelo e alguns gráficos.

### Construção do *dataset*

A segunda etapa do *pipeline* é a construção do *dataset*, que consiste na leitura do arquivo manifest.ndjson contendo os dados de todas as imagens e na organização dessas imagens nos diretórios corretos. Inicialmente é realizada a divisão aleatória dos dados nos subconjuntos de treino, validação e teste, respeitando as proporções parametrizadas na configuração do experimento. Essa divisão é conduzida pela função construir\_dataset, da classe ManipuladorDados, que internamente utiliza um método da biblioteca sklearn para garantir aleatoriedade no particionamento.

Para cada imagem de cada subconjunto citado, é executada a função gerar\_imagens\_para\_item, responsável por converter as anotações de *bounding boxes* originalmente fornecidas no formato LabelBox para o formato YOLO através da função xyxy\_para\_yolo. Nessa etapa também ocorre o salvamento das imagens e arquivos de rótulos nas pastas de debug e dataset. Dependendo das configurações do experimento, essa função ainda pode executar a etapa adicional de *tiling*, dividindo cada imagem em múltiplas regiões menores com sobreposição ajustável, o que aumenta a densidade de amostras e auxilia na detecção de objetos muito pequenos. A Figura 13 ilustra um exemplo de entrada e saída do processo de salvamento das imagens. O item (a) apresenta a imagem original, utilizada como base para gerar a imagem de saída, o item (b) apresenta a primeira opção de saída, sendo a imagem original redimensionada para a resolução de 800x800 pixels, e o item (c) apresenta a segunda opção de saída, gerando os *tiles* a partir da imagem original, com um *overlap* de 20%.

Figura – Exemplo de *tiling*

|  |  |  |
| --- | --- | --- |
|  |  |  |
| (a) Entrada: imagem original | (b) Saída: imagem original redimensionada | (c) Saída: *tiles* gerados a partir da imagem original |

Fonte: elaborado pelo autor.

Quando o *tiling* está habilitado, o processo inclui ainda o recálculo das coordenadas das caixas delimitadoras para cada *tile* gerado, além da filtragem de objetos cuja interseção com o *tile* seja inferior ao limite estabelecido. Isso garante que apenas anotações relevantes sejam preservadas, evitando que regiões contendo apenas pequenas frações de neurônios sejam tratadas como exemplos positivos. Esse mecanismo contribui para reduzir ruído e melhora a qualidade dos dados de treino, especialmente em imagens de alta resolução contendo neurônios muito pequenos e distribuídos de forma esparsa.

Por outro lado, quando o *tiling* está desativado, a função simplesmente preserva a imagem original em sua resolução completa e converte todas as anotações diretamente para o formato YOLO baseado no sistema de coordenadas da imagem inteira. Nessa abordagem, o redimensionamento é executado internamente pelo YOLO durante o carregamento dos dados, permitindo manter a fidelidade espacial da imagem original ao mesmo tempo em que o modelo recebe entradas padronizadas. Esse modo é especialmente útil para análises comparativas entre *tiles* e imagens inteiras, além de servir como referência para verificar o impacto do recorte no desempenho final.

Como parte complementar, o *pipeline* também gera imagens de *debug* com anotações sobrepostas, tanto das imagens originais quanto de cada *tile* produzido. Esses artefatos visuais desempenham um papel importante na validação do *dataset*, pois permitem inspecionar rapidamente se as conversões, recortes e normalizações foram aplicados corretamente. A Figura 14 apresenta um exemplo de imagem de *debug* em tamanho original e um exemplo de *tile*.

Figura – Exemplos de imagens de *debug*

|  |  |
| --- | --- |
|  |  |
| (a) Imagem de *debug* em resolução original | (b) Imagem de *debug* em formato de *tile* |

Fonte: elaborado pelo autor.

Esse processo reduz a chance de erros silenciosos no *pipeline* e facilita ajustes finos posteriores na etapa de pré-processamento, contribuindo diretamente para a consistência e confiabilidade dos experimentos.

### Treinamento do modelo

A etapa de treinamento do modelo representa o núcleo do *pipeline*, pois é nela que o YOLO aprende efetivamente a detectar neurônios atípicos a partir dos exemplos fornecidos no *dataset* construído anteriormente. Durante o treinamento, cada imagem do conjunto de treino é processada pela rede juntamente com suas anotações no formato YOLO, permitindo que o modelo compare as previsões geradas com os valores reais e atualize seus parâmetros por meio do algoritmo de retropropagação. Esse processo é repetido ao longo de diversas épocas, nas quais o modelo se torna progressivamente mais capaz de identificar padrões morfológicos sutis presentes nos neurônios, como variações de formato, textura, brilho e distribuição espacial.

Para controlar esse processo, o *script* utiliza a API da biblioteca Ultralytics, que permite configurar parâmetros essenciais como o número de épocas, taxa de aprendizado, estratégia de *batching* e limiares utilizados durante a inferência. A configuração pode ser centralizada em estruturas de dicionário ou em arquivos YAML, dependendo da necessidade do experimento. Além disso, o modelo automaticamente aplica *letterboxing* e normalização de *pixels*, padronizando as imagens antes de enviá-las para a rede. O Quadro 14 apresenta a função principal de treinamento do modelo.

Quadro – Função de treinamento do modelo YOLO

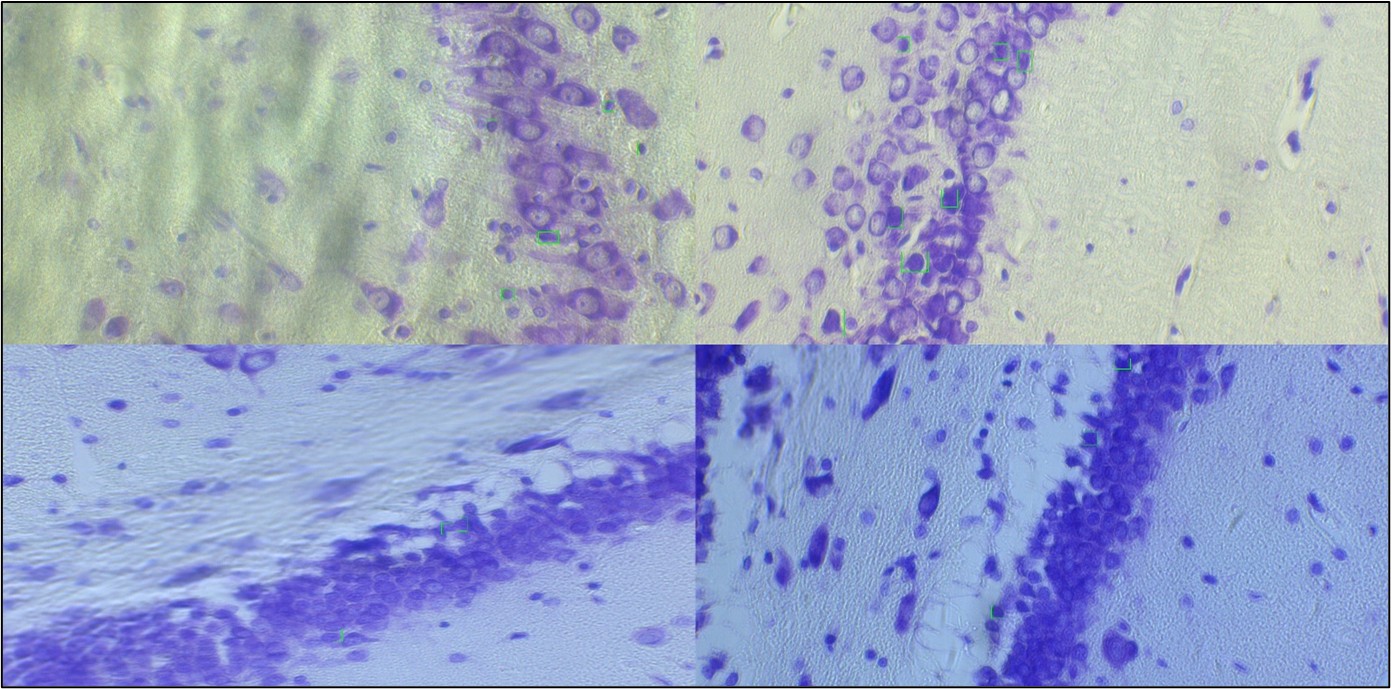
|  |
| --- |
| def treinar\_fase1(      self,      caminho\_data\_yaml: str,      imgsz: int,      num\_classes: int  ):      print("\n=== Treinando Fase 1 (sem ) ===")      pasta\_fase1 = self.caminhos\_pastas["modelos\_fase1"]      modelo = self.\_inicializar\_modelo(num\_classes)      params = self.\_montar\_parametros\_treino(         caminho\_data\_yaml, imgsz, self.config.epochs\_fase1, pasta\_fase1      )      modelo.train(\*\*params)    caminho\_best = os.path.join(pasta\_fase1, "treino", "weights", "best.pt")      self.modelo\_fase1 = YOLO(caminho\_best)      self.caminho\_best\_fase1 = caminho\_best      print(f"Fase 1 concluída. best.pt em: {caminho\_best}") |

Fonte: elaborado pelo autor.

Outro aspecto relevante do treinamento é o uso das técnicas de *data augmentation* fornecidas pelo próprio YOLOv8, combinadas ou não com aquelas aplicadas anteriormente na etapa de construção do *dataset*. Em alguns experimentos foram utilizados recursos como *mosaic*, *mixup*, inversão horizontal e *jitter* de cor aumentam significativamente a variabilidade das amostras vistas pelo modelo, reduzindo o risco de *overfitting* e melhorando a generalização em imagens capturadas sob condições diferentes das utilizadas para anotar o *dataset* original. Essa etapa de enriquecimento das amostras pode ser ajustada diretamente na configuração do experimento.

A técnica *Mosaic*, que combina quatro imagens em uma única composição maior, aumenta a diversidade espacial e pode melhorar o aprendizado para objetos raros, entretanto, para objetos muito pequenos pode reduzir ainda mais seu tamanho relativo, dificultando a detecção. Em alguns experimentos foi aplicada a técnica *mosaic*, que tem como objetivo expor o modelo a composições mais variadas de objetos e contextos durante o treinamento. Essa técnica combina quatro imagens em uma única composição maior e pode melhorar o aprendizado para objetos raros, entretanto, para objetos muito pequenos pode reduzir ainda mais seu tamanho relativo, dificultando a detecção. A Figura 15 exemplifica o resultado da técnica *mosaic*.

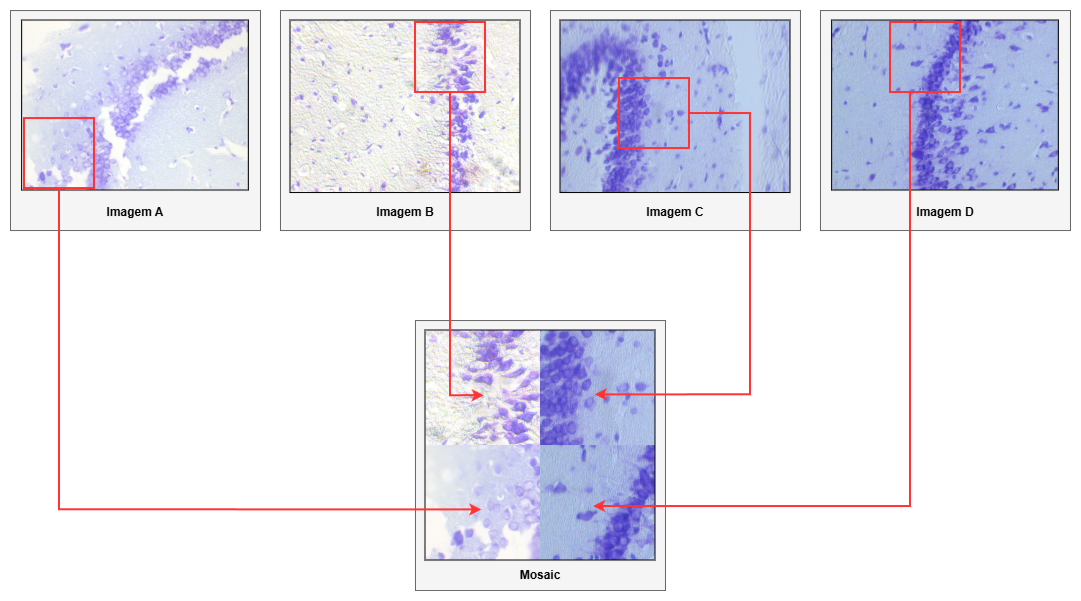
Figura – *Mosaic* aplicado com imagens em resolução original



Fonte: elaborado pelo autor.

Quando usadas imagens em sua resolução original, a imagem gerada pode ter uma resolução muito maior do que a indicada para o modelo. Por este motivo, constatou-se que esta técnica é mais adequada para os experimentos que utilizam o *tiling*, visto que agrupa imagens menores, não ultrapassando um limite comum de resolução. A seguir, a Figura 16 apresenta um exemplo da técnica *mosaic* aplicado em *tiles*, onde são selecionados *tiles* aleatórios, extraídos das imagens originais para formar uma nova imagem.

Figura – *Mosaic* aplicado com *tiles*



Fonte: elaborado pelo autor.

Já o *mixup*, também utilizado neste projeto, mistura duas imagens de forma ponderada juntamente com seus rótulos, sendo útil para regularização e para suavizar fronteiras de decisão. No entanto, essa técnica pode diluir visualmente objetos pequenos, tornando-os menos definidos e, no contexto deste estudo, dificultando a identificação de estruturas muito sutis, como os neurônios atípicos. Por esse motivo, o *mixup* foi aplicado com leve intensidade, apenas em alguns experimentos, servindo mais como apoio à diversidade do *dataset* do que como elementos centrais do processo de treinamento. A Figura 17 apresenta um exemplo da aplicação do *mixup*, ilustrando como as imagens são combinadas e como as anotações são preservadas após a fusão.

Figura – Aplicação do *mixup*

Interface gráfica do usuário, Diagrama

O conteúdo gerado por IA pode estar incorreto.

Fonte: elaborado pelo autor.

Por fim, ao término do treinamento, o YOLO salva automaticamente os melhores pesos (best.pt) e o último estado do modelo (last.pt), permitindo realizar testes posteriores e comparações entre diferentes execuções do experimento. Esse conjunto de artefatos é utilizado na etapa seguinte para validação e análise de desempenho nas imagens do conjunto de validação e teste.

### Geração de relatórios e execução do teste do modelo

A etapa seguinte do *pipeline* tem como objetivo consolidar os resultados do treinamento e produzir materiais visuais que auxiliem na análise do desempenho do modelo. Para isso, inicialmente é utilizada a classe GeradorRelatorios, que é responsável por organizar e sintetizar as informações produzidas durante o treinamento. Este processo salva arquivos JSON e CSV com dados sobre o experimento e gera gráficos com métricas para avaliação.

Após consolidar os resultados da fase de treinamento, o *pipeline* carrega o modelo final por meio do método obter\_modelo\_final, que seleciona o arquivo best.pt, ou seja, o conjunto de pesos que obteve o melhor desempenho no conjunto de validação. Esse passo garante que as próximas análises utilizem a melhor versão do modelo produzida durante o processo de treinamento, evitando influências de épocas menos eficientes.

Com o modelo final carregado, utiliza-se a classe AvaliadorYOLO, responsável por executar a inferência em todo o *dataset* e gerar um *debug* completo das predições. O método gerar\_debug\_completo percorre todas as imagens dos subconjuntos de treino, validação e teste, e produz saídas visuais contendo as caixas previstas pelo modelo sobrepostas às imagens originais. Além disso, quando o *dataset* é baseado em *tiles*, essa etapa também reconstrói visualmente as predições no contexto da imagem original, permitindo identificar regiões onde o modelo teve maior dificuldade, erros específicos e padrões de acertos. Essa análise é essencial para entender a qualidade prática do modelo, complementando as métricas numéricas produzidas pelo YOLO.

Por fim, essa etapa funciona como uma ponte entre o treinamento e a análise final dos resultados, reunindo tanto informações quantitativas quanto qualitativas. Enquanto os relatórios da fase 1 ajudam a visualizar a evolução do modelo ao longo das épocas, o *debug* completo permite uma inspeção direta das predições, facilitando a identificação de limitações, padrões e oportunidades de ajuste para experimentos futuros. Dessa forma, essa parte do *pipeline* garante que o desempenho do modelo seja avaliado de forma abrangente, ultrapassando a interpretação dos valores numéricos e explorando/analisando os resultados de maneira visual.

## CONFIGURAÇÕES Dos EXPERIMENTOs

Ao longo do desenvolvimento do trabalho diversas combinações de parâmetros foram avaliadas, a fim de identificar quais seriam as melhores configurações para o cenário atual, dentre eles, parâmetros de *data augmentation*, tamanhos de imagem e tamanhos de modelo. O YOLOv8 é disponibilizado em diferentes escalas de modelo — *nano* (n), *small* (s), *medium* (m), *large* (l) e *extra-large* (x) — que variam principalmente em profundidade da rede, largura dos blocos convolucionais e quantidade total de parâmetros. Essas versões seguem uma lógica de *trade-off* entre velocidade e capacidade de representação: modelos menores são mais rápidos e leves, porém tendem a capturar menos detalhes em estruturas pequenas; modelos maiores possuem maior poder discriminativo, mas demandam mais memória e tempo de treinamento. No contexto deste trabalho, todas as variantes foram avaliadas justamente para investigar como essa escala de complexidade influencia o desempenho na detecção dos neurônios atípicos, que são objetos muito pequenos e de baixa densidade nas imagens. Essa comparação permitiu identificar não apenas qual configuração obteve melhor F1-score, mas também entender o equilíbrio entre custo computacional e precisão para a tarefa específica.

Embora o YOLO ofereça configurações padrão de detecção baseadas nas camadas P3, P4 e P5, foi realizada uma experimentação adicional incluindo uma camada P2, que opera em uma resolução espacial ainda mais alta. A inclusão da P2 faz sentido neste contexto pois os objetos são muito pequenos e podem desaparecer conforme a imagem passa por etapas de *downsampling* dentro da rede. Ao introduzir essa camada extra, o modelo passa a receber recursos mais detalhados das primeiras fases da convolução, preservando informações finas que poderiam ser perdidas. Assim, testar a P2 permitiu investigar se essa representação de alta resolução poderia melhorar a sensibilidade da detecção em objetos tão reduzidos. A arquitetura P2 pode ser visualizada no Quadro 15, que apresenta o conteúdo do arquivo yolov8\_p2\_small\_custom.yaml.

Quadro – Conteúdo do arquivo de arquitetura P2 customizada

|  |
| --- |
| # YOLOv8-P2 small customizado para detecção de objetos pequenos (neurônios atípicos)  nc: 1  depth\_multiple: 0.67  width\_multiple: 1.0  # -------------------------------------  # BACKBONE – Extração inicial de características  # -------------------------------------  backbone:  - [ -1, 1, Conv, [64, 3, 2] ]  - [ -1, 1, Conv, [128, 3, 2] ]  - [ -1, 3, C2f, [128, True] ]  - [ -1, 1, Conv, [256, 3, 2] ]  - [ -1, 6, C2f, [256, True] ]  - [ -1, 1, Conv, [512, 3, 2] ]  - [ -1, 6, C2f, [512, True] ]  - [ -1, 1, Conv, [1024, 3, 2] ]  - [ -1, 3, C2f, [1024, True] ]  - [ -1, 1, SPPF, [1024, 5] ]  # -------------------------------------  # HEAD – Combinação das escalas e detecção  # -------------------------------------  head:  - [ -1, 1, Conv, [512, 1, 1] ]  - [ -1, 1, nn.Upsample, [None, 2, 'nearest'] ]  - [ [ -1, 6 ], 1, Concat, [1] ]  - [ -1, 3, C2f, [512] ]  - [ -1, 1, Conv, [256, 1, 1] ]  - [ -1, 1, nn.Upsample, [None, 2, 'nearest'] ]  - [ [ -1, 4 ], 1, Concat, [1] ]  - [ -1, 3, C2f, [256] ]  - [ -1, 1, Conv, [256, 3, 2] ]  - [ [ -1, 6 ], 1, Concat, [1] ]  - [ -1, 3, C2f, [512] ]  - [ -1, 1, Conv, [512, 3, 2] ]  - [ [ -1, 8 ], 1, Concat, [1] ]  - [ -1, 3, C2f, [1024] ]  - [ [ 11, 14, 17, 20 ], 1, Detect, [nc] ] |

Fonte: elaborado pelo autor.

A arquitetura personalizada baseia-se no modelo YOLOv8, adaptado para melhorar o desempenho na detecção de objetos muito pequenos, como os neurônios atípicos presentes nas imagens microscópicas. Essa adaptação, chamada de YOLOv8-P2, adiciona uma resolução extra na parte inicial da rede (camada P2), permitindo que o modelo análise detalhes mais finos que normalmente seriam perdidos em arquiteturas tradicionais. Em termos simples, o *backbone* da rede funciona como um extrator de padrões: ele reduz progressivamente a resolução da imagem ao mesmo tempo em que aprende características relevantes, como bordas, texturas e formas. Para isso, utiliza blocos de convolução e módulos C2f, que ajudam a combinar informações de maneira eficiente e estável.

Na segunda parte da rede, chamada de *head*, ocorre a junção de informações de diferentes tamanhos de imagem, o que permite detectar objetos tanto grandes quanto muito pequenos. Essa combinação é feita por meio de operações de *upsampling*, que recuperam detalhes, e concatenação, que junta informações de várias escalas. Por fim, a camada de detecção recebe mapas de características de diferentes resoluções e gera as predições finais — as *bounding boxes* e suas respectivas probabilidades. Em conjunto, essa arquitetura permite que o YOLOv8-P2 seja mais eficiente para tarefas que exigem atenção a estruturas pequenas.

Além das variações de arquitetura, diferentes resoluções de entrada também foram avaliadas, testando imagens com 512, 640, 768, 800, 896, 1.024 e 1.280 *pixels* de cada lado. Essa variação é relevante pois o tamanho da imagem influencia diretamente a capacidade do modelo de preservar detalhes finos — um ponto crítico neste trabalho, dado que os objetos a serem detectados ocupam poucos *pixels* na cena original. Resoluções menores tendem a acelerar o treinamento e reduzir o custo computacional, mas podem comprometer a detecção de objetos muito pequenos devido à perda de informação após o *downsampling*. Por outro lado, resoluções maiores mantêm mais detalhes e tendem a melhorar o *recall*, embora com maior uso de GPU e tempo de processamento. Assim, a experimentação com múltiplos tamanhos permitiu analisar o impacto concreto da resolução na detecção dessas estruturas microscópicas, garantindo uma comparação equilibrada entre desempenho e eficiência.

Os modelos YOLO costumam demandar um número maior de épocas no treinamento, portanto, o número de épocas também foi variado entre 100 e 200, permitindo observar como a rede evoluía ao longo do aprendizado e em que ponto passava a estabilizar ou saturar. Como o *dataset* é relativamente pequeno, poucas épocas poderiam resultar em *underfitting*, deixando o modelo incapaz de capturar adequadamente as características morfológicas sutis dos objetos. Por outro lado, épocas demais poderiam levar ao *overfitting*, especialmente em um cenário com baixa densidade de verdadeiros positivos. Junto ao número maior de épocas, foi utilizado um recurso nativo do YOLO, que permite interromper o treinamento após determinado número de épocas sem ganho razoável (early stopping), economizando tempo computacional.

## RESULTADOS E DISCUSSÕES

Os experimentos conduzidos neste trabalho variaram em três eixos principais: tamanho do modelo YOLOv8, resolução de entrada e estratégias de pré-processamento, incluindo ou não *tiling* e diferentes níveis de *data augmentation*. Também foram testados ajustes de hiperparâmetros, como tamanho de *tile*, sobreposição entre *tiles* e número de épocas de treinamento. Em termos gerais, os experimentos sem *tiling* e com *pipeline* de pré-processamento mais conservador apresentaram resultados mais estáveis, enquanto combinações mais agressivas — com *tiling* ou *augmentations* mais intensas — produziram maior variação de desempenho.

A Tabela 1 apresenta a média aritmética simples do desempenho por tamanho de modelo na fase de teste, ou seja, o desempenho após o treinamento, com um conjunto de imagens que o modelo nunca viu nas etapas anteriores.

Tabela – Desempenho médio por tamanho de modelo

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Sem *data augmentation* | | | Com *data augmentation* | | |
| Modelo | Precisão | Recall | F1 | Precisão | Recall | F1 |
| yolov8n | 38.47% | 39.39% | 38.88% | 37.29% | 38.73% | 37.87% |
| yolov8s | 40.47% | 40.72% | 40.42% | 40.02% | 40.96% | 40.40% |
| yolov8m | 39.43% | 43.07% | 41.05% | 38.36% | 41.63% | 39.77% |
| yolov8l | 31.61% | 40.15% | 40.79% | 37.71% | 39.94% | 38.57% |
| yolov8x | 38.53% | 45.68% | 41.73% | 41.32% | 42.67% | 41.87% |

Fonte: elaborado pelo autor.

Como comentado anteriormente, a aplicação do *data augmentation*, na maioria das vezes, reduz o desempenho do modelo. Conforme mostra a Tabela 1, dentre os testes sem *data augmentation*, o modelo yolov8x apresentou o melhor desempenho médio nesta comparação, com 38.53% de precisão e 45.68% de *recall*, que juntos resultam em 41.73% de F1-score. Por outro lado, a melhor média foi do mesmo modelo, porém com *data augmentation* ativo, atingindo 41.87% de F1-score. A seguir, a Tabela 2 apresenta outra comparação, segmentando os dados por tamanho de imagem.

Tabela – Desempenho médio por tamanho de imagem

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Sem *data augmentation* | | | Com *data augmentation* | | |
| Modelo | Precisão | Recall | F1 | Precisão | Recall | F1 |
| 768px | | 40.03% | 41.59% | 40.78% | 37.16% | 39.76% | 38.33% |
| 800px | | 37.07% | 42.57% | 39.47% | 42.40% | 39.32% | 40.71% |
| 896px | | 42.28% | 41.27% | 41.63% | 37.95% | 41.33% | 39.39% |
| 1.024px | | 37.97% | 43.52% | 40.52% | 37.35% | 40.52% | 38.86% |
| 1.280px | | 41.84% | 38.66% | 40.16% | 39.84% | 43.01% | 41.17% |

Fonte: elaborado pelo autor.

Os dados presentes na Tabela 2 confirmam, novamente, que o *data augmentation* neste contexto pode interferir negativamente nos resultados. Para a maioria dos tamanhos de imagem, a ausência do aumento de dados gerou resultados melhores. Nesse cenário, os experimentos com imagens de 896 *pixels* resultaram no melhor resultado médio, sem aumento de dados, atingindo F1-score de 41.63%. A média mais alta foi das imagens com 1.280 *pixels* e *data augmentation* ativo, atingindo precisão de 39.84%, recall de 43.01% e F1-score de 41.17%.

Os experimentos foram executados sistematicamente, gerando um resultado uniforme e possibilitando o cálculo da média aritmética e comparação dos resultados com maior precisão, mas também é importante demonstrar os resultados específicos dos experimentos. A Tabela 3 apresenta os 3 experimentos com o melhor F1-score na fase de teste.

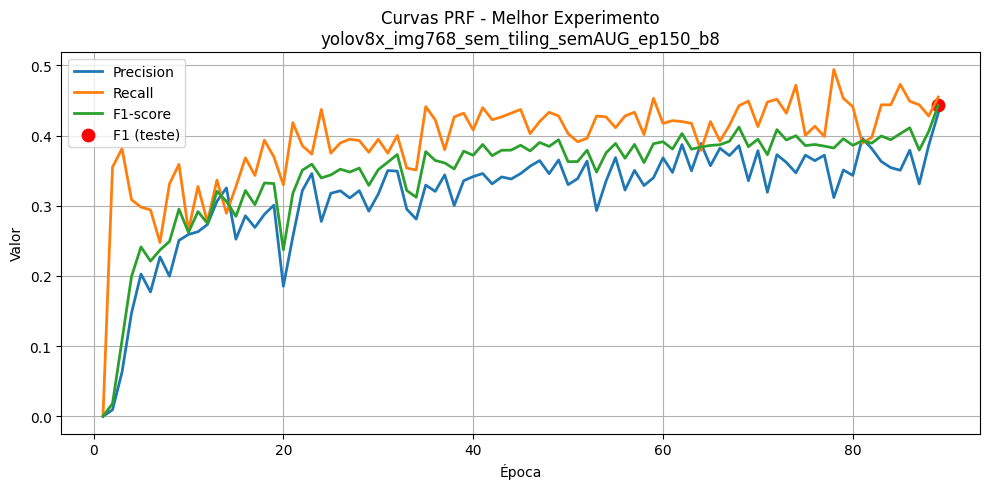
Tabela – Experimentos com melhor desempenho na fase de teste

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Modelo | Tamanho imagem | *Data augmentation* | Precisão | Recall | F1 |
| yolov8x | 768px | Não | 43.20% | 45.48% | 44.31% |
| yolov8x | 896px | Não | 40.81% | 46.69% | 43.55% |
| yolov8s | 800px | Sim | 45.50% | 40.66% | 42.94% |

Fonte: elaborado pelo autor.

O primeiro item da Tabela 3 representa o experimento com o melhor desempenho, utilizando o modelo yolov8x, um tamanho de imagem de 768 pixels e sem *data augmentation*, o experimento atingiu o F1-score de 44,31%. Na Figura 18 é possível visualizar o gráfico comparativo das métricas deste experimento. As linhas do gráfico representam as métricas durante a fase de validação do modelo, com valores para cada época. O valor do F1-score da fase de teste é destacado pelo ponto vermelho.

Figura – Gráfico de precisão, recall e F1-score do melhor experimento



Fonte: elaborado pelo autor.

Analisando o gráfico, pode-se observar que as métricas mantiveram valores constantes durante as épocas. Neste experimento, a métrica *recall* atingiu 45.48% na melhor época, isso significa que neste momento o modelo conseguiu encontrar 45.48% de todos os neurônios atípicos. O modelo também alcançou 43.20% de precisão, ou seja, das predições realizadas por ele, 43.20% estavam corretas. A união das duas métricas nessa época resultou no F1-score de 44.31%, um valor muito bom, considerando a dificuldade de caracterizar os neurônios atípicos.

De forma complementar, é possível analisar os resultados deste experimento visualizando uma imagem do conjunto de teste. A Figura 19 ilustra, no item (a), a imagem original. As caixas de cor verde representam a anotação realizada na plataforma LabelBox, ou seja, os neurônios atípicos. Já as caixas vermelhas são os resultados da predição do modelo. No item (b) da Figura 19 é apresentada a matriz de confusão deste exemplo, que mostra a predição do modelo em comparação ao que deveria ser detectado como neurônio atípico.

Figura – Exemplo de imagem com anotações de *ground truth* e predições

|  |  |
| --- | --- |
|  |  |
| 1. Amostra de teste | (b) Matriz de confusão da amostra |

Fonte: elaborado pelo autor.

A partir da Figura 19 pode-se observar na imagem e na sua matriz de confusão, que o desempenho do modelo é razoável. Dentre os 12 objetos originais na imagem, o modelo conseguiu identificar 9, porém, 2 deles tiveram o *intersection over union* (IoU) abaixo de 50%. Isso significa que a localização da caixa predita estava muito imprecisa em relação ao *ground truth*. Consequentemente, para a métrica mAP@50 essas duas predições são classificadas como falsos positivos.

A seguir, o Quadro 16 apresenta outros exemplos de predições feitas pelo modelo em imagens do conjunto de teste, novamente, com as anotações em verde e vermelho representando a anotação original do LabelBox e a predição do modelo, respectivamente.

Quadro – Exemplos de imagens da fase de teste

|  |  |
| --- | --- |
|  | Nesta imagem existem 7 neurônios marcados como atípicos. Destes, 6 foram encontrados pelo modelo, restando 1 que não foi predito. Também houve uma predição incorreta, afirmando haver um neurônio atípico onde na verdade não havia. A métrica de F1-score ficou em 85.71%. |
|  | Neste caso, existem 5 objetos a serem detectados, dos quais 4 foram encontrados com sucesso. Pode-se observar que embora a qualidade da imagem possa parecer inferior, o modelo ainda conseguiu entender o padrão e encontrar os objetos. O F1-score foi de 88.88%. |
|  | Nesta imagem existiam 6 neurônios atípicos anotados. O modelo conseguiu encontrar 4 deles, errando 2 e, realizando 1 predição incorreta. O F1-score foi de 72.72%. |
|  | Por fim, nesta imagem o modelo não performou bem. A imagem continha um total de 5 neurônios atípicos anotados, e apenas 1 foi encontrado. Os outros 4 não foram apontados pelo modelo, que também fez predições em outros objetos, gerando vários falsos positivos e falsos negativos. O F1-score obtido foi de 20%. |

Fonte: elaborado pelo autor

Além dos experimentos mencionados, também foram executados com outras configurações, como por exemplo o uso do *tiling* e *data augmentation* mais variados. Porém, estes não foram executados de forma sistemática e, portanto, não há a possibilidade de fazer uma comparação completa entre eles. Mas, ainda é possível detalhar alguns destes experimentos, a fim de entender o comportamento do modelo em diversas configurações, conforme demonstra a Tabela 4.

Tabela – Desempenho em experimentos com configurações diversas

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Modelo | Tamanho imagem | Data augmentation | Precisão | Recall | F1 |
| yolov8s-P2 | 800px | Sim | 38.62% | 27.33% | 32.01% |
| yolov8s | 800px | Não | 20.16% | 25.38% | 22.50% |

Fonte: elaborado pelo autor.

Pode-se observar que os experimentos com configurações diversas não obtiveram bons resultados, em comparação com os experimentos mais conservadores. Isso é um indicativo de que no contexto atual, com um *dataset* reduzido e objetos difíceis de identificar, qualquer tipo de manipulação dos dados pode distorcer os objetos nas imagens e interferir no resultado, sendo assim, abordagens mais conservadoras são mais indicadas.

No início do desenvolvimento do trabalho, foram avaliadas outras arquiteturas baseadas em redes neurais convolucionais (CNNs), incluindo EfficientDet, VGG16 e InceptionV3. Entretanto, essas abordagens não apresentaram desempenho satisfatório para a tarefa específica de identificação de neurônios atípicos, seja por limitações na detecção de objetos muito pequenos ou pelo tempo computacional elevado para o treinamento. Diante desse cenário, optou-se por concentrar os esforços no modelo YOLO, cuja estrutura mostrou-se mais adequada às características do *dataset* e aos requisitos da tarefa.

Para definir a média de acertos por imagem, capturaram-se dez imagens do conjunto de teste, ou seja, que não haviam sido utilizadas durante o treinamento. Em seguida, essas dez imagens foram submetidas ao modelo de identificação de neurônios atípicos, o que permitiu a obtenção dos dados necessários para calcular a média de acertos do modelo, conforme exemplifica o Quadro 17. Nesse processo, a correspondência entre as detecções do modelo e as marcações de referência foi verificada visualmente, considerando-se como acertos apenas as detecções que coincidiam com as anotações reais. Assim, a taxa de acerto foi definida como o número de correspondências dividido pelo número total de marcações presentes em cada imagem.

Quadro – Análise de correspondências de dez imagens

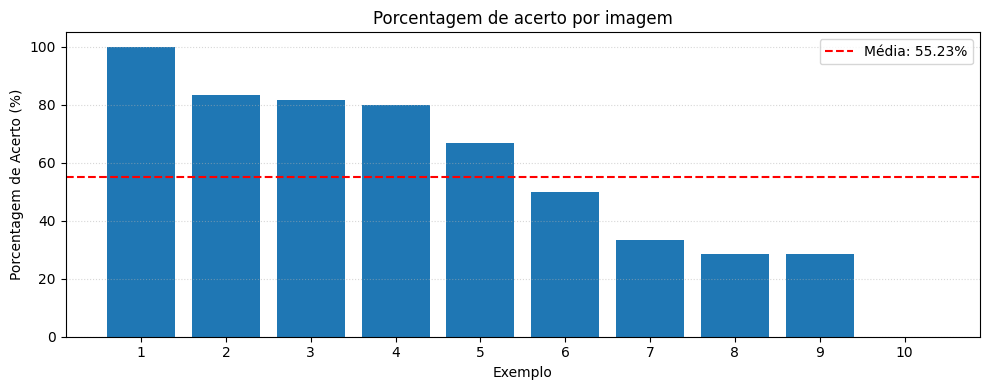
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Exemplo | Marcações do Acadêmicos | Identificações Modelo | Correspondências | % Acerto |
| 1 | 3 | 5 | 3 | 100,00% |
| 2 | 6 | 11 | 5 | 83,33% |
| 3 | 11 | 15 | 9 | 81,81% |
| 4 | 5 | 4 | 4 | 80,00% |
| 5 | 6 | 5 | 4 | 66,66% |
| 6 | 4 | 3 | 2 | 50,00% |
| 7 | 6 | 6 | 2 | 33,33% |
| 8 | 7 | 4 | 2 | 28,57% |
| 9 | 7 | 4 | 3 | 28,57% |
| 10 | 4 | 1 | 0 | 00,00% |

Fonte: elaborado pelo autor.

Ao analisar os dados, constatou-se que a média geral de acertos do modelo neste conjunto de imagens aleatórias é de 55,23%. Observando as imagens processadas e os dados coletados, nota-se que o modelo tem a tendência de identificar menos neurônios atípicos do que os presentes nas imagens. No entanto, em grande parte dos casos, o modelo consegue apontar corretamente a maioria dos neurônios indicados pelos acadêmicos do curso de medicina.

Uma análise positiva do modelo pode ser observada na primeira linha do Quadro 17. A imagem em questão, possuía 5 neurônios atípicos, e o modelo conseguiu identificar todos eles, não perdendo nenhum objeto. Por outro lado, na última linha, que representa uma imagem com 4 neurônios atípicos, o modelo não conseguiu localizar nenhum dos objetos, fazendo somente uma predição incorreta. A Figura 20 apresenta os dados em formato gráfico para facilitar a leitura.

Figura – Percentual de acerto por imagem



Fonte: elaborado pelo autor.

Cada coluna no gráfico da Figura 20 representa um exemplo do Quadro 17, e a linha vermelha pontilhada no gráfico da Figura 20 representa a média geral do percentual de acerto para os dez exemplos de imagens analisadas.

Além disso, a fim de ilustrar visualmente a variação de desempenho observada entre as imagens analisadas, a Figura 21 apresenta um comparativo entre a imagem com maior percentual de acertos, que atingiu 100% de correspondência, à esquerda, e aquela em que o modelo não conseguiu identificar nenhum neurônio atípico resultando em 0% de acerto, à direita. A comparação revela diferenças importantes tanto na qualidade da imagem quanto nas características morfológicas percebidas pelo modelo.

Figura – Comparativo entre imagens com melhor e pior desempenho

|  |  |
| --- | --- |
|  |  |
| 1. Melhor desempenho | 1. Pior desempenho |

Fonte: elaborado pelo autor.

Na imagem de melhor desempenho, observa-se uma camada celular nítida, bem definida e com excelente contraste entre os neurônios e o tecido ao redor. Os contornos celulares são claramente delimitados, há pouca interferência de artefatos ópticos e a coloração uniforme facilita a separação visual entre os objetos. Essas condições favorecem a atuação do modelo, permitindo que todas as estruturas marcadas pelos acadêmicos fossem devidamente reconhecidas.

Por outro lado, a imagem de pior desempenho apresenta menor nitidez, com regiões desfocadas e coloração mais difusa, o que reduz significativamente o contraste entre os neurônios e o fundo. Além disso, a organização da camada celular é menos evidente, dificultando a distinção entre os corpos celulares e o tecido adjacente. Tais condições visuais, combinadas com possíveis artefatos e variações de iluminação, tornam mais desafiadora a identificação automática dos neurônios atípicos, resultando em perda total das detecções nesta imagem específica.

Torna-se evidente que fatores como nitidez, qualidade óptica, contraste, uniformidade da coloração e organização estrutural do tecido influenciam diretamente o desempenho do modelo. Assim, mesmo com uma média geral razoável, casos extremos de imagens com baixa qualidade podem comprometer a capacidade de detecção, reforçando a necessidade de padronização dos procedimentos de captura, preparação e processamento das lâminas.

Ao comparar os resultados obtidos neste trabalho com os apresentados por Krzizanowski e Carvalho (2024) fica evidente que, apesar das diferenças específicas de implementação e metodologia, o desempenho final permanece em faixas próximas. No estudo original, a Faster R-CNN alcançou aproximadamente 40% na métrica F1-score, já neste trabalho, ao empregar o YOLOv8x com resolução de 768 pixels, sem *tiling* e sem *data augmentation*, foi obtido um F1-score de 44,31%. Ou seja, quando se observa a métrica comparável entre os dois estudos (F1-score), nota-se que o modelo YOLOv8 se mantém ligeiramente acima da Faster R-CNN.

Além disso, a variedade de experimentos conduzidos neste trabalho permitiu compreender melhor o impacto de ajustes metodológicos (como *tiling*, resoluções maiores e *data augmentation*), algo que não foi explorado no estudo anterior. Assim, embora os modelos adotem abordagens distintas, os resultados mostram um equilíbrio entre as propostas.

# CONCLUSÕES

A identificação automática de neurônios atípicos representa um desafio significativo para a neurociência, especialmente pela sutileza das alterações morfológicas presentes nesse tipo celular e pela limitação de *datasets* disponíveis. Este trabalho buscou aprofundar a investigação iniciada por Krzizanowski e Carvalho (2024), avaliando se arquiteturas alternativas de aprendizado profundo — como a família YOLO — poderiam alcançar desempenho equivalente ou superior ao obtido pelos autores, ao mesmo tempo em que reduzissem o tempo de processamento e aprimorassem a aplicabilidade prática do sistema proposto.

O objetivo geral, que consistia em aplicar novas técnicas de aprendizado profundo para a identificação e comparação do desempenho de modelos na detecção de neurônios atípicos, foi plenamente atendido. Os objetivos específicos também foram contemplados. Novas estratégias de pré-processamento foram aplicadas, como por exemplo o *tiling*. Um conjunto diversificado de experimentos foi conduzido com YOLOv8 — variando resoluções, número de épocas, camadas P2 e configurações de *data augmentation* — permitindo avaliar profundamente o comportamento da arquitetura na tarefa proposta. Realizou-se a validação comparando os resultados com o estudo original, utilizando métricas de precisão, recall e F1-score, além de uma avaliação visual das predições.

Do ponto de vista metodológico, as técnicas utilizadas mostraram-se adequadas ao problema estudado. A modularização do *pipeline* e a execução de diversas configurações de experimentos permitiram análises sistemáticas e controladas. Observou-se que algumas técnicas — como *Mosaic* e *Mixup* — nem sempre foram benéficas neste cenário, devido ao tamanho reduzido dos objetos a serem detectados. Da mesma forma, verificou-se que o *tiling* não trouxe vantagem nas configurações, já que a fragmentação excessiva de imagens pode prejudicar a contextualização espacial do objeto de interesse.

O melhor experimento, utilizando YOLOv8x com resolução de 768 pixels, sem *tiling* e sem *data augmentation*, atingiu F1-score de 44,31%, valor muito próximo do desempenho relatado no estudo original, no qual o modelo Faster R-CNN obteve aproximadamente 40%. Essa proximidade demonstra que arquiteturas modernas de detecção em etapa única também são capazes de lidar com a tarefa de identificar neurônios atípicos, contanto que ajustes metodológicos sejam adotados. A estabilidade observada entre diferentes configurações do YOLO reforça sua robustez, mesmo diante das limitações impostas pelo *dataset*.

Ao relacionar esse comportamento com os trabalhos correlatos, observa-se que cada abordagem destaca, à sua maneira, a importância da consistência do dado de entrada para obtenção de bons resultados. Angonezi (2022) mostra que, em imagens de microscopia de luz visível, pequenas alterações na aparência das células podem comprometer tarefas de detecção devido ao baixo contraste e às estruturas delicadas — ponto que reforça a necessidade de *pipelines* cuidadosos. Já Santos *et al.* (2023) obtêm excelente desempenho em classificação de células sanguíneas mesmo com *data augmentation*, mas isso ocorre em um contexto com grande volume de dados, objetos bem definidos e classes amplas, diferente do cenário deste trabalho. Troullinou *et al.* (2021), embora atuem em um domínio distinto — classificando neurônios a partir de sinais temporais de cálcio, e não de imagens — também evidenciam como modelos de aprendizado profundo dependem fortemente de dados estáveis e representativos para distinguir categorias biologicamente próximas. Assim, em conjunto, os correlatos reforçam que a qualidade e a integridade do dado são determinantes para o desempenho dos modelos, independentemente da natureza da tarefa.

Essa interpretação também se alinha ao modelo desenvolvido por Krzizanowski e Carvalho (2024), cujo desempenho mais estável foi obtido em condições controladas e com imagens preservando suas características originais. Tanto o modelo baseado em Faster R-CNN quanto os experimentos conduzidos neste trabalho apontam na mesma direção: quando se trabalha com um *dataset* reduzido, objetos muito pequenos e baixa separação visual, intervenções agressivas no pré-processamento — como alterações intensas de resolução, mosaicos ou composições artificiais — tendem a introduzir ruído e dificultar a aprendizagem. Assim, a comparação entre os experimentos realizados, os trabalhos correlatos e o modelo atual evidenciam um padrão consistente: abordagens mais conservadoras são, no momento, as mais adequadas para garantir detecções estáveis. Esse conjunto de análises encerra a seção ao demonstrar que, embora diferentes metodologias e arquiteturas possam ser empregadas, o fator decisivo permanece na integridade e representatividade dos dados utilizados no treinamento.

Entretanto, algumas limitações precisam ser destacadas. O *dataset* continua sendo um fator crítico: o número de neurônios atípicos é pequeno quando comparado ao número de neurônios saudáveis presentes nas imagens, sua distribuição é dispersa e a morfologia apresenta forte variabilidade. Estas características dificultam o aprendizado do modelo e podem amplificar problemas de *overfitting*. Outra limitação relevante é a dificuldade de detecção de objetos extremamente pequenos, mesmo com camadas P2 e resoluções elevadas.

Como recomendações para trabalhos futuros, sugere-se a ampliação e diversificação da base de dados, incorporando imagens de diferentes microscópios, qualidades e resoluções. Adicionalmente, a marcação explícita dos demais neurônios presentes nas imagens como saudáveis poderia contribuir para uma classificação mais precisa, uma vez que o modelo deixaria de distinguir apenas entre neurônio atípico e fundo, passando a diferenciar neurônio atípico, neurônio saudável e fundo separadamente.

Conclui-se, portanto, que o modelo YOLOv8 é uma alternativa viável e eficiente para a identificação de neurônios atípicos em imagens microscópicas, sendo capaz de atingir desempenho semelhante ao obtido anteriormente com Faster R-CNN. Além de validar a aplicabilidade dessa arquitetura em contextos biomédicos, este estudo contribui com uma análise sistemática que pode orientar futuras pesquisas e aprimoramentos no diagnóstico assistido por visão computacional.

Referências

ALI, Bushra A. *et al*. **Effect of Electromagnetic Field (EMF) of cellular phones on Purkinje cell margins and perineuronal space in the cortex of rat cerebellum.** Pakistan Journal of Medical & Health Sciences, v. 14, n. 4, p. 958–961, 2020. Disponível em: https://www.researchgate.net/publication/355719951\_Effect\_of\_Electromagnetic\_Field\_EMF\_of\_cellular\_phones\_on\_purkinje\_cell\_margins\_and\_perineuronal\_space\_in\_the\_cortex\_of\_rat\_cerebellum. Acesso em: 30 nov. 2025.

ALJARARI, Rabab. M. **Neuroprotective effects of a combination of Boswellia papyrifera and Syzygium aromaticum on AlCl3 induced Alzheimer’s disease in male albino rat.** Brazilian Journal of Biology, v. 83, 01 jan. 2023.

AMARAL, David. G.; LAVENEX, Pierre. **Hippocampal neuroanatomy**. In: ANDERSEN, P.; MORRIS, R.; AMARAL, D. G.; BLISS, T.; O’KEEFE, J. (org.). The hippocampus book. Oxford: Oxford University Press, 2007. p. 37–114. DOI: 10.1093/acprof:oso/9780195100273.003.0003. Disponível em: https://academic.oup.com/book/25965/chapter-abstract/193768286?redirectedFrom=fulltext. Acesso em: 18 nov. 2025.

ANGONEZI, Angelo Luiz. **Aprendizado de máquina aplicado à detecção, classificação e análise de células tumorais em imagens de microscopia de luz visível**. 2022. Trabalho de Conclusão de Curso (Bacharelado em Biotecnologia – Ênfase em Biotecnologia Molecular) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2022. Disponível em: https://lume.ufrgs.br/handle/10183/270636. Acesso em: 12 out. 2025.

BOCHKOVSKIY, Alexey *et al*. YOLOv4: Optimal Speed and Accuracy of Object Detection. **arXiv preprint arXiv:2004.10934**, 2020. Disponível em: https://arxiv.org/abs/2004.10934. Acesso em: 01 dez. 2025.

CERVANTES, Evelyn Perez. **Classificação morfológica de neurônios baseada na hierarquia das arvores dendríticas**. 2019. Tese (Doutorado em Ciências). Instituto de Matematica e Estatística, Universidade de São Paulo, São Paulo.

CORRADETTI, Daniele; BERNARDI, Alessandro; CORRADETTI, Renato. Deep learning models for atypical serotonergic cells recognition. **Journal of Neuroscience Methods**, [S.l.], v. 407, 110158, 2024. DOI: 10.1016/j.jneumeth.2024.110158. Disponível em: https://www.sciencedirect.com/science/article/pii/S0165027024001031. Acesso em: 18 nov. 2025.

GBD 2021 NERVOUS SYSTEM DISORDERS COLLABORATORS. Global, regional, and national burden of disorders affecting the nervous system, 1990–2021: a systematic analysis for the Global Burden of Disease Study 2021. **The Lancet Neurology**, [S.l.], v. 23, n. 4, p. 344–381, abr. 2024. DOI: 10.1016/S1474-4422(24)00038-3. Disponível em: https://www.thelancet.com/journals/laneur/article/PIIS1474-4422(24)00038-3/fulltext. Acesso em: 26 nov. 2025.

GOEL, Parul *et al*. **Neuronal cell death mechanisms in Alzheimer’s disease**: An insight. Frontiers in Molecular Neuroscience, [S.l.], v. 15, art. 937133, 2022. DOI: 10.3389/fnmol.2022.937133. Disponível em: https://www.frontiersin.org/articles/10.3389/fnmol.2022.937133/full. Acesso em: 26 nov. 2025.

GOODFELLOW, Ian *et al*. **Deep learning**. Cambridge: MIT Press, 2016. Disponível em: https://www.deeplearningbook.org/. Acesso em: 12 nov. 2025.

HE, Kaiming *et al*. Deep residual learning for image recognition. **arXiv preprint arXiv:1512.03385**, 2015. DOI: 10.48550/arXiv.1512.03385. Disponível em: https://arxiv.org/abs/1512.03385. Acesso em: 18 nov. 2025

JOCHER, Glenn *et al*. **YOLOv8: New State-of-the-Art Real-Time Object Detector**. 2023. Zenodo. DOI: 10.5281/zenodo.7802523. Disponível em: https://www.scirp.org/journal/paperinformation?paperid=126758. Acesso em: 01 dez. 2025.

KRZIZANOWSKI, Gabriel; CARVALHO, Lucas Eduardo de. **Modelo de aprendizado de máquina para a contagem e identificação de neurônios atípicos**. 2024. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Universidade Regional de Blumenau, Blumenau, 2024.

LECUN, Yann *et al*. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998. DOI: 10.1109/5.726791. Disponível em: https://ieeexplore.ieee.org/document/726791. Acesso em: 22 out. 2025.

MA, Feiyo. *et al*. **Metallothionein 3 attenuated the apoptosis of neurons in the CA1 region of the hippocampus in the senescence-accelerated mouse/PRONE8 (SAMP8)**. Arquivos de Neuro-Psiquiatria, [S.L.], v. 69, n. 1, p. 105-111, fev. 2011. FapUNIFESP (SciELO).

MACHADO, Angelo. **Neuroanatomia funcional**. 2. ed. São Paulo: Atheneu, 2013.

MACHADO, Annelisa Pimentel Rezende; CARVALHO, Izabella Oliveira; SOBRINHO, Hermínio Maurício da Rocha. Neuroinflamação na doença de Alzheimer. **Revista Brasileira Militar de Ciências**, [S.l.], v. 6, n. 14, p. 30–38, 2020. DOI: 10.36414/rbmc.v6i14.33. Disponível em: https://rbmc.org.br/rbmc/article/view/33. Acesso em: 26 out. 2025.

MATANÓ, Beatriz Regina de Sousa; ZEMLENOI, Laryssa Greggio; PACHECO, Thaisy. Prognóstico da doença de Alzheimer a partir da identificação de biomarcadores. **Ensaios – Revista de Ciências Humanas e Sociais da Universidade São Francisco**, [S.l.], v. 27, n. 1, p. 47–56, 2023. Disponível em: https://ensaios.usf.edu.br/ensaios/article/view/361. Acesso em: 26 mar. 2025.

MISHRA, Bhabatosh. **Understanding Convolutional Neural Networks**. 2020. Disponível em: https://arxiv.org/abs/2010.13141. Acesso em: 01 dez. 2025.

MONTANARI, T.; **Histologia – Texto, Atlas e Roteiro de Aulas Práticas** – Série Graduação – 2 Ed. Rio Grande do Sul: Ufrgs, 2006. 155p. Disponível em: https://www.nucleodoconhecimento.com.br/saude/sistema-nervoso. Acesso em: 26 nov. 2025.

MOREIRA, Catarina. Neurónio. **Revista de Ciência Elementar**, [S.l.], v. 1, n. 1, art. 008, dez. 2013. DOI: 10.24927/rce2013.008. Disponível em: https://rce.casadasciencias.org/rceapp/art/2013/008/. Acesso em: 12 nov. 2025.

PHAM, Minh-Tan *et al*. **YOLO-Fine**: one-stage detector of small objects under various backgrounds in remote sensing images. Remote Sensing, [S.L.], v. 12, n. 15, p. 2501, 4 ago. 2020. MDPI AG. <http://dx.doi.org/10.3390/rs12152501>.

REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. **arXiv preprint arXiv:1804.02767**, 2018. Disponível em: https://arxiv.org/abs/1804.02767. Acesso em: 21 nov. 2025.

REDMON, Joseph *et al*. You Only Look Once: Unified, Real-Time Object Detection. **arXiv preprint arXiv:1506.02640**, 2016. DOI: 10.48550/arXiv.1506.02640. Disponível em: https://arxiv.org/abs/1506.02640. Acesso em: 18 nov. 2025.

RIBEIRO, Edivaine G. **Rede neural convolucional aplicada ao reconhecimento de passagens de nível clandestinas em Ferrovias**. 2020. 59 f. Monografia (Especialista em Sistemas Inteligentes Aplicados a Automação) - Instituto Federal do Espírito Santo, Vitória, 2020.

ROSA, João Vitor. **Redes neurais convolucionais**. 2018. Disponível em: https://repositorio.ifsc.edu.br/handle/123456789/1579. Acesso em: 20 nov. 2025.

SANTOS, Mariana Dourado X. S. *et al*. Algoritmos de aprendizado de máquina para classificação de células nucleadas do sangue periférico: uma experiência do projeto Hemovision. **Anais do ERIGO** 2022, [S.l.], p. 1–8, 2022. Disponível em: https://sol.sbc.org.br/index.php/erigo/article/view/22543. Acesso em: 26 nov. 2025.

SECRETÁRIO, José Henrique Azevedo; PIRES, Ricardo. Uso de visão computacional para contagem automática de células em imagens obtidas por microscópios. **REGRASP – Revista para Graduandos / IFSP-Câmpus São Paulo**, [S.l.], v. 3, n. 3, p. 4–17, jun. 2018. Disponível em: https://regrasp.spo.ifsp.edu.br/index.php/regrasp/article/view/234. Acesso em: 12 nov. 2025.

SIMONYAN, Karen; ZISSERMAN, Andrew. **Very Deep Convolutional Networks for Large-Scale Image Recognition**. 2015. Disponível em: https://arxiv.org/abs/1409.1556. Acesso em: 20 nov. 2025.

SOLAWETZ, Jacob; ZUPPICHINI, Francesco Saverio. **What is YOLOv8?**: The Ultimate Guide. 2023. Disponível em: https://blog.roboflow.com/whats-new-in-yolov8/. Acesso em: 01 dez. 2025.

SZEGEDY, Christian *et al*. Going Deeper with Convolutions. **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, 2015. DOI: 10.1109/CVPR.2015.7298594. Disponível em: https://www.cv-foundation.org/openaccess/content\_cvpr\_2015/html/Szegedy\_Going\_Deeper\_With\_2015\_CVPR\_paper.html. Acesso em: 20 nov. 2025.

TROULLINOU, Eirini *et al*. Artificial neural networks in action for an automated cell-type classification of biological neural networks. **IEEE Transactions on Emerging Topics in Computational Intelligence**, [S.l.], v. 5, n. 5, p. 755–767, out. 2021. DOI: 10.1109/TETCI.2020.3028581. Disponível em: https://ieeexplore.ieee.org/document/9222095. Acesso em: 12 nov. 2025.

VALESO, V. **Neurônios**: Unidade Funcional do Tecido Nervoso. Aracaju, [2021]. Disponível em: https://www.passeidireto.com/arquivo/89714279/neuronios-unidade-funcional-do-tecido-nervoso. Acesso em: 12 nov. 2025.

WANG, Chien-Yao *et al*. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. **arXiv preprint arXiv:2207.02696**, 2022. Disponível em: https://arxiv.org/abs/2207.02696. Acesso em: 22 nov. 2025.