1ª EDIÇÃO

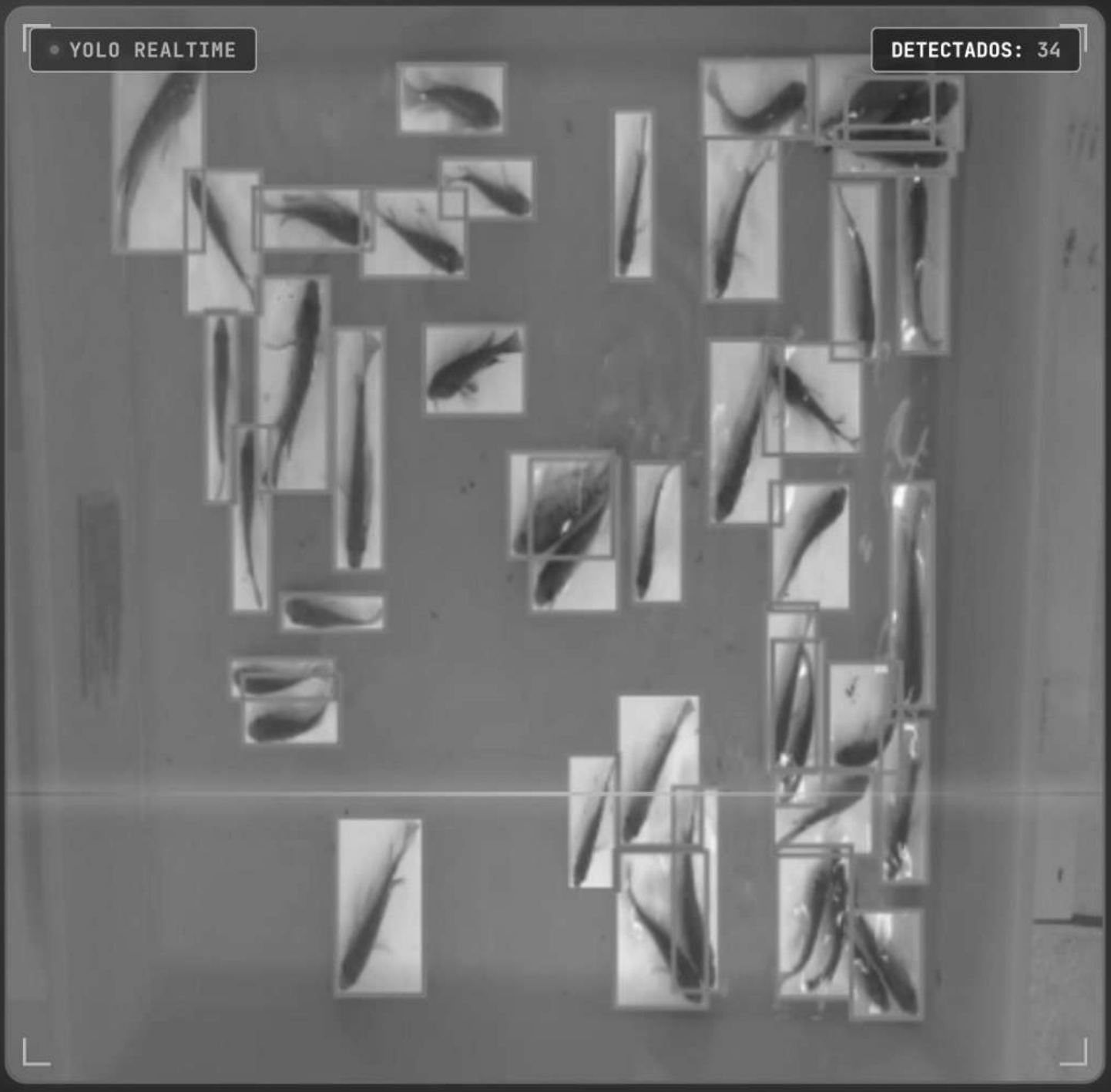
SERGIO SOUZA NOVAK
JULIANA FORTUNATO
ANDREA BARBOZA PROTO SARDI
GABRIELA ALMEIDA MARQUES
RONALDO CAMILO DOS SANTOS
LORENA SANTANA ARANTES
ADRIANO SOARES DE OLIVEIRA BAILÃO
LEANDRO MANUEL PEREIRA SANCHES
MATHEUS BARP PIEROZAN
GIDELIA ARAÚJO FERREIRA DE MELO
ADRIANO CARVALHO COSTA
GUSTAVO HENRIQUE JOAZEIRO DE ALMEIDA



DESENVOLVIMENTO DE SISTEMAS DE INTELIGÊNCIA ARTIFICIAL COM FOCO EM VISÃO COMPUTACIONAL APOIADO EM REDES NEURAIS CONVOLUCIONAIS DA FAMÍLIA YOU ONLY LOOK ONCE (YOLO)

1ª EDIÇÃO

SERGIO SOUZA NOVAK
JULIANA FORTUNATO
ANDREA BARBOZA PROTO SARDI
GABRIELA ALMEIDA MARQUES
RONALDO CAMILO DOS SANTOS
LORENA SANTANA ARANTES
ADRIANO SOARES DE OLIVEIRA BAILÃO
LEANDRO MANUEL PEREIRA SANCHES
MATHEUS BARP PIEROZAN
GIDELIA ARAÚJO FERREIRA DE MELO
ADRIANO CARVALHO COSTA
GUSTAVO HENRIQUE JOAZEIRO DE ALMEIDA



DESENVOLVIMENTO DE SISTEMAS DE INTELIGÊNCIA ARTIFICIAL COM FOCO EM VISÃO COMPUTACIONAL APOIADO EM REDES NEURAIS CONVOLUCIONAIS DA FAMÍLIA YOU ONLY LOOK ONCE (YOLO)

Editora Omnis Scientia

DESENVOLVIMENTO DE SISTEMAS DE INTELIGÊNCIA ARTIFICIAL COM FOCO EM VISÃO COMPUTACIONAL APOIADO EM REDES NEURAIS CONVOLUCIONAIS DA FAMÍLIA YOU ONLY LOOK ONCE (YOLO)

Volume 1

1ª Edição

RECIFE - PE

2026

EDITOR-CHEFE

Dr. Daniel Luís Viana Cruz

AUTORES

Sergio Souza Novak

Adriano Soares de Oliveira Bailão

Juliana Fortunato

Leandro Manuel Pereira Sanches

Andrea Barboza Proto Sardi

Matheus Barp Pierozan

Gabriela Almeida Marques

Gidelia Araújo Ferreira de Melo

Ronaldo Camilo dos Santos

Adriano Carvalho Costa

Lorena Santana Arantes

Gustavo Henrique Joazeiro de Almeida

CONSELHO EDITORIAL

Dr. Amâncio António de Sousa Carvalho - ESS-UTAD - Portugal

Dr. Cássio Brancaleone - UFFS - Brasil

Dr. Marcelo Luiz Bezerra da Silva - UEPa - Brasil

Dra. Pauliana Valéria Machado Galvão - UPE - Brasil

Dr. Plínio Pereira Gomes Júnior - UFRPE - Brasil

Dr. Walter Santos Evangelista Júnior - UFRPE - Brasil

Dr. Wendel José Teles Pontes - UFPE - Brasil

EDITORES DE ÁREA - CIÊNCIAS EXATAS E DA TERRA

Dr. André Luiz Leite Ferreira

Dra. Angela Maria Corrêa Mouzinho Santos

Dr. Filipe Mendonça de Lima

ASSISTENTE EDITORIAL

Thialla Larangeira Amorim

IMAGEM DE CAPA

Canva e OpenAI (ChatGPT)

EDIÇÃO DE ARTE

Nhatallia Laranjeira Amorim

REVISÃO

Os autores

Este trabalho está licenciado com uma Licença Creative Commons – Atribuição-NãoComercial-SemDerivações 4.0 Internacional.

O conteúdo abordado nos artigos, seus dados em sua forma, correção e confiabilidade são de responsabilidade exclusiva dos autores.

Dados Internacionais de Catalogação na Publicação (CIP)
Lumos Assessoria Editorial

D451
Desenvolvimento de sistemas de inteligência artificial com foco em visão computacional apoiado em redes neurais convolucionais da família You Only Look Once [recurso eletrônico] / Sergio Souza Novak ... [et al.]. — 1. ed. — Recife : Omnis Scientia, 2026.
Dados eletrônicos (pdf).

Inclui bibliografia.
ISBN 978-65-284-0683-8
DOI: 10.47094/978-65-284-0683-8

1. Inteligência artificial – Aplicações médicas. 2. Redes neurais (Ciência da computação) - Aplicações médicas. 3. Sistemas de detecção de objetos (YOLO). I. Novak, Sergio Souza.

CDD23: 610.285

I-0311261/2

Bibliotecária: Priscila Pena Machado – CRB-7/6971

Editora Omnis Scientia

Av. República do Líbano, nº 251, Sala 2205, Torre A,

Bairro Pina, CEP 51.110-160, Recife-PE.

Telefone: +55 (87) 9920-5762

editoraomnisscientia.com.br

contato@editoraomnisscientia.com.br



AUTORES

Sergio Souza Novak

Mestre em Ciências Agrárias - Agronomia (PPGCA - AGRO) – IF Goiano – Campus Rio Verde - e bacharel em Ciência da Computação pelo Instituto Federal Goiano. Professor na Universidade de Rio Verde (UniRV) especificamente na Faculdade de Engenharia de Software.

Adriano Soares de Oliveira Bailão

Docente do curso de Bacharelado em Ciência da Computação do IF Goiano – Campus Rio Verde. Doutor em Ciência da Computação. Atua nas áreas de Ciência da Computação, Inteligência Computacional e Engenharia de Software.

Juliana Fortunato

Doutora em Física pela Universidade Estadual de Maringá (UEM), com formação em Bacharelado e Licenciatura em Física pela mesma instituição e Licenciatura em Pedagogia pelo Centro Universitário Unifacvest. Professora da Secretaria de Estado da Educação do Paraná (SEED-PR). Atua no ensino de Física, Matemática e Química na Educação Básica e Profissional Técnica, com experiência em práticas experimentais e Ensino de Ciências.

Leandro Manuel Pereira Sanches

Mestre em Ciências da Computação pela Universidade Federal da Bahia, Salvador, Brasil. Professor na Universidade de Rio Verde (UniRV) na Faculdade de Engenharia de Software. Atua na área de Banco de Dados e Inteligência Artificial (Processamento de Linguagem Natural).

Andrea Barboza Proto Sardi

Mestre em Engenharia Elétrica pela Universidade Estadual Paulista (UNESP) e graduada em Ciência da Computação. Servidora efetiva do Instituto Federal Goiano e doutoranda no Programa de Pós-Graduação em Ciências Agrárias (PPGCA-AGRO). Atua como docente no Ensino Superior com experiência em Sistemas de Computação. Desenvolve pesquisas interdisciplinares integrando Internet das Coisas (IoT), Visão Computacional, Drones e Inteligência Artificial aplicadas ao agronegócio.

Matheus Barp Pierozan

Graduação em Medicina Veterinária pela Universidade Federal de Santa Maria (2008) e Pós-Graduação Lato Sensu em Ciências Avícolas pela Universidade Federal de Uberlândia (2010). É Mestre em Zootecnia pelo Programa de Pós-Graduação em Zootecnia (PPGZ) do Instituto Federal Goiano Campus Rio Verde (2022) e atualmente é Doutorando em Agroquímica pelo Programa de Pós-Graduação em Agroquímica (PPGAq) pelo mesmo Instituto.

Gabriela Almeida Marques

Bacharel em Zootecnia pelo Instituto Federal Goiano (Campus Rio Verde) e mestranda em Zootecnia pela mesma instituição. Atua na área de aquicultura com foco em inteligência artificial e visão computacional aplicadas à contagem de alevinos, biometria e avaliação de carcaça de tilápias, com experiência em projetos EMBRAPII e desenvolvimento de produtos tecnológicos.

Gidelia Araújo Ferreira de Melo

Licenciada em Química e mestre em Zootecnia pelo Instituto Federal Goiano, onde cursa o doutorado em Biotecnologia e Biodiversidade. Desenvolve pesquisas interdisciplinares em química, aquicultura e biotecnologia, com ênfase em sustentabilidade produtiva, aditivos alimentares e detecção e contagem automatizada de peixes por meio de visão computacional.

Ronaldo Camilo dos Santos

Bacharel em Análise de Sistemas pela PUC-Campinas e mestre em Engenharia Elétrica (Automação) pela Unicamp. Professor na Faculdade de Engenharia de Software da Universidade de Rio Verde (UniRV). Atua na área de Engenharia Elétrica e Computação, com ênfase em telecomunicações, radiocomunicação, ciência de dados e inteligência artificial.

Adriano Carvalho Costa

Doutor, mestre e graduado em Zootecnia pela Universidade Federal de Lavras (UFLA), com doutorado-sanduíche na Universidade Nacional de Rosario (Argentina). Professor do Instituto Federal Goiano (Campus Rio Verde), coordenador do Núcleo de Pesquisa Aplicada à Pesca e Aquicultura (NUPA) e bolsista de produtividade em Desenvolvimento Tecnológico e Extensão Inovadora do CNPq (DT-1D). Atua na coordenação e orientação em programas de pós-graduação, liderando projetos de inovação tecnológica, visão computacional, IoT e inteligência artificial aplicadas à aquicultura e produção animal.

Lorena Santana Arantes

Graduada em Medicina Veterinária pela Universidade de Rio Verde (UniRV). Possui experiência profissional nas áreas de clínica veterinária, gestão e desenvolvimento comercial, com atuação em projetos de tecnologia, inovação e sustentabilidade aplicadas à produção e à saúde de animais não ruminantes.

Gustavo Henrique Joazeiro de Almeida

Mestre em Produção Animal – Zootecnia (PPGZ) pela Universidade Federal da Grande Dourados e bacharel em Medicina Veterinária pelo Centro Universitário Moura Lacerda. Atua como professor universitário do curso de Medicina Veterinária pela Faculdade Unilasalle Lucas do Rio Verde e médico-veterinário na Clínica Veterinária Bonasalute em Lucas do Rio Verde.

PREFÁCIO

O campo da Inteligência Artificial, e em especial a área de Visão Computacional, passou por uma revolução sem precedentes na última década. O que antes era restrito a grandes corporações de tecnologia e complexos centros de pesquisa, hoje encontra-se acessível a qualquer pessoa munida de determinação. A família de modelos YOLO (You Only Look Once) representa o ápice dessa democratização, combinando extrema precisão técnica com velocidades de processamento em tempo real.

Foi da necessidade de construir uma ponte sólida entre a complexa teoria das redes neurais profundas e a prática aplicada que esta obra nasceu. O que apresentamos a você, leitor, não é apenas um apanhado teórico, mas sim um guia definitivo e totalmente prático, desenhado para conduzi-lo por todo o processo de preparação, treinamento e avaliação de redes YOLO.

Seja você um profissional de saúde buscando automatizar a detecção de anomalias em exames de imagem, um engenheiro agrônomo realizando a contagem e classificação de frutos e pragas, um especialista em segurança pública rastreando veículos, ou um gestor industrial monitorando defeitos em linhas de montagem: os princípios estruturados neste livro aplicam-se diretamente à sua realidade. Este guia foi construído pensando em profissionais das mais diversas áreas, provando que não é necessário possuir uma formação prévia em engenharia de software avançada para dominar e aplicar o estado da arte da Inteligência Artificial.

Ao longo das próximas páginas, caminharemos passo a passo. Desde a organização bruta dos dados e o entendimento profundo dos hiperparâmetros vitais de treinamento, passando pela interpretação correta das métricas de validação e matrizes de confusão, até o dimensionamento da infraestrutura de hardware adequada (CPU vs GPU) e a automação de backups na nuvem para garantir a segurança dos seus experimentos.

Esperamos que este livro sirva como um verdadeiro manual de bancada, capaz de impulsionar seus projetos, pesquisas e inovações profissionais. Que a leitura seja o primeiro passo para que suas ideias ganhem visão e autonomia.

Boa leitura e um excelente treinamento!

Os Autores

SUMÁRIO

CAPÍTULO 1.....12
INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL E À VISÃO COMPUTACIONAL: CONCEITOS, ARQUITETURAS E APLICAÇÕES PRÁTICAS
DOI: 10.47094/978-65-284-0683-8/12-27

CAPÍTULO 2.....28
EVOLUÇÃO E ARQUITETURA DA FAMÍLIA YOLO
DOI: 10.47094/978-65-284-0683-8/28-42

CAPÍTULO 3.....43
CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO, PADRÃO DE DATASET E TREINAMENTO PARA YOLO
DOI: 10.47094/978-65-284-0683-8/43-56

CAPÍTULO 4.....57
AVALIAÇÃO DE MODELOS YOLO
DOI: 10.47094/978-65-284-0683-8/57-74

CAPÍTULO 5.....75
GUIA DIDÁTICO DE INFRAESTRUTURA E HARDWARE PARA TREINAMENTO
DOI: 10.47094/978-65-284-0683-8/75-79

CAPÍTULO 6.....80
BACKUP AUTOMATIZADO E OTIMIZADO COM RCLONE E PYTHON
DOI: 10.47094/978-65-284-0683-8/80-87

CAPÍTULO 7.....88
CONCLUSÃO, CICLO DE VIDA E IMPLANTAÇÃO
DOI: 10.47094/978-65-284-0683-8/88-90

CAPÍTULO 1

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL E À VISÃO COMPUTACIONAL: CONCEITOS, ARQUITETURAS E APLICAÇÕES PRÁTICAS

1. INTRODUÇÃO E EVOLUÇÃO CONCEITUAL DA INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial (IA) consolidou-se como um dos pilares tecnológicos mais transformadores da sociedade contemporânea, redefinindo as fronteiras da automação e da análise de dados em diversas áreas do conhecimento humano. Conceituada formalmente na metade do século XX, a IA pode ser entendida didaticamente como o ramo da Ciência da Computação dedicado ao projeto e desenvolvimento de sistemas computacionais capazes de simular, emular ou ampliar aptidões cognitivas historicamente associadas à inteligência humana, tais como o aprendizado contínuo, a inferência lógica, a tomada de decisão em cenários de incerteza e a percepção sensorial (TURING, 1950; RUSSELL; NORVIG, 2010). Em suas primeiras décadas de desenvolvimento, o paradigma dominante da IA baseava-se em abordagens simbólicas e sistemas especialistas baseados em regras programadas manualmente. Nesses sistemas tradicionais, especialistas humanos traduziam o conhecimento de um determinado domínio em um conjunto rígido de instruções condicionais da forma se-então. Contudo, à medida que os problemas do mundo real se tornaram mais complexos e dinâmicos — exibindo grande volume de ruído, variação dimensional e imprecisão —, as limitações do paradigma simbólico tornaram-se evidentes. A transição para o Aprendizado de Máquina (Machine Learning) representou uma mudança fundamental de abordagem: em vez de construir explicitamente todas as regras de inferência, os algoritmos passaram a extrair automaticamente padrões estatísticos e generalizações a partir de grandes volumes de dados de treinamento.

2. O PARADIGMA DO APRENDIZADO SUPERVISIONADO

O Aprendizado de Máquina (Machine Learning) é um método que automatiza a construção de modelos analíticos, no qual os sistemas aprendem diretamente com os dados a identificar padrões complexos e a tomar decisões com o mínimo de intervenção humana.

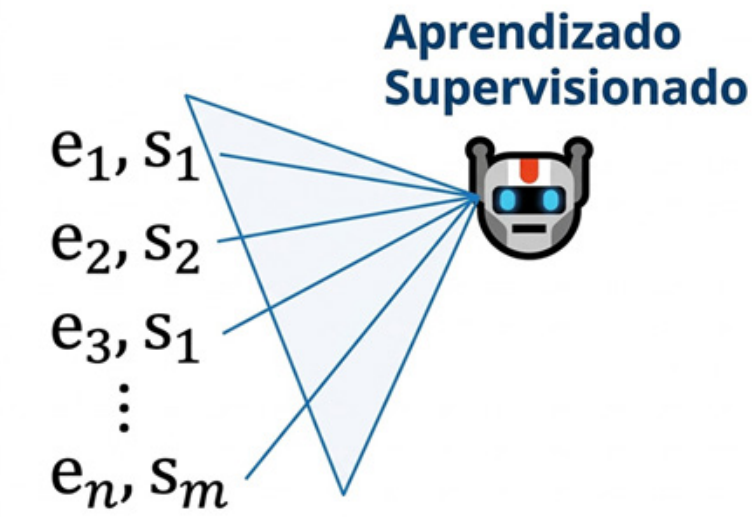
Trata-se da técnica fundamental na qual as máquinas adquirem conhecimento e passam a executar ações com base no aprendizado extraído dos dados, e não apenas a partir da entrada rígida de instruções estáticas. Dentre os diversos métodos de aprendizado computacional, os tipos de algoritmos mais bem-sucedidos são aqueles que automatizam a tomada de decisão generalizando a partir de exemplos conhecidos. No aprendizado supervisionado (Supervised Learning), tem-se uma tarefa de aprendizado de máquina que consiste em aprender uma função de mapeamento $f(x)$ que associa uma entrada x a uma saída desejada y, a partir de pares de entrada e saída (x_i, y_i) fornecidos ao

sistema. Tudo isso é possível graças à presença de um supervisor (humano ou especialista de domínio), que estabelece a relação entre as entradas e os rótulos (labels). Eles são assim chamados porque o supervisor fornece supervisão direta aos algoritmos, oferecendo as saídas desejadas para cada exemplo durante a fase de treinamento. No processo de aprendizagem, contudo, o agente computacional pode apresentar erros de generalização ao ser submetido a dados novos e não observados anteriormente. O grande desafio dos algoritmos supervisionados reside justamente em minimizar esses erros de generalização, ajustando internamente seus parâmetros para maximizar a acurácia do modelo sem cair em memorização excessiva (overfitting). Os dois exemplos e categorias fundamentais do Aprendizado Supervisionado são:

- **Classificação:** Tarefa na qual a saída prevista é um rótulo categórico ou discreto (por exemplo, identificar a espécie de um inseto ou determinar o estágio de amadurecimento de um fruto).
- **Regressão:** Tarefa na qual a saída predita é uma variável numérica contínua (como estimar as coordenadas geográficas de um objeto ou prever a biomassa agrícola).

A Figura 1 ilustra didaticamente o fluxo conceitual do Aprendizado Supervisionado, evidenciando como o par de entrada e saída (x_i, y_i) orienta a calibração do modelo preditivo. O processo de refinamento iterativo da função $f(x)$ e a avaliação dos erros de generalização durante a etapa de inferência em dados inéditos são detalhados na Figura 2. Por sua vez, a Figura 3 apresenta esquematicamente a subdivisão das tarefas supervisionadas entre problemas de Classificação (com saídas discretas/categóricas) e regressão (com saídas contínuas).

Figura 1: Representação do Aprendizado Supervisionado: o supervisor fornece pares rotulados de entrada (x_i) e saída desejada (y_i), orientando o algoritmo a encontrar a função $f(x)$ e a minimizar erros de generalização ao predizer novas entradas.



2.1 Tarefas e Algoritmos de Classificação e Regressão

No âmbito do Aprendizado Supervisionado, a estruturação da tarefa preditiva divide-se essencialmente em duas modalidades operacionais: a Classificação e a Regressão. Na Classificação, o agente de aprendizado computacional aprende segundo as características das entradas fornecidas, de forma que os rótulos (labels) representam as classes categóricas discretas possíveis para cada entrada. Nesse contexto, a missão do algoritmo é mapear os dados de entrada para uma das categorias pré-determinadas pelo supervisor. Os algoritmos mais famosos e utilizados em tarefas de classificação incluem:

- **K-Vizinhos Mais Próximos (K-Nearest Neighbors – KNN):** Classifica uma nova entrada com base na proximidade e voto majoritário dos k exemplos mais próximos no espaço de atributos.
- **Naive Bayes:** Algoritmo estatístico probabilístico fundamentado no Teorema de Bayes com suposição de independência entre as variáveis de entrada.

Figura 2: Fluxo de inferência e avaliação de generalização no Aprendizado Supervisionado: utilização da função aprendida $f(x)$ para predição de saídas em dados de teste.

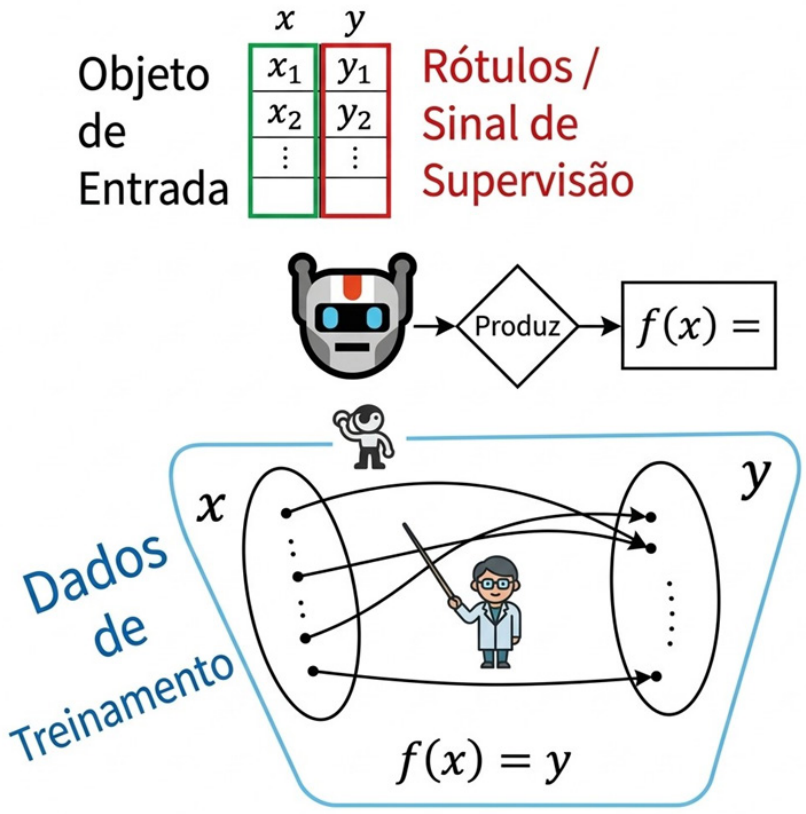
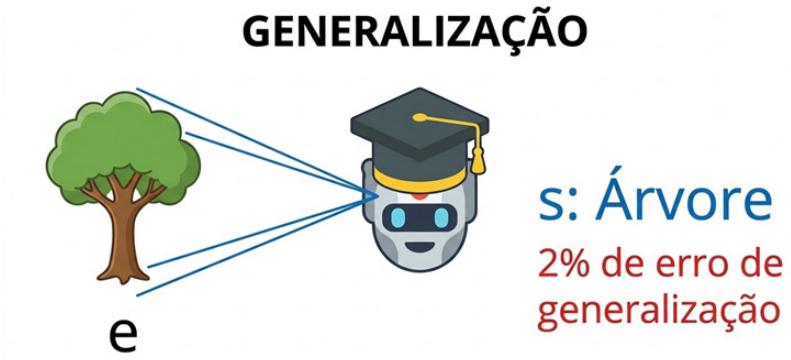


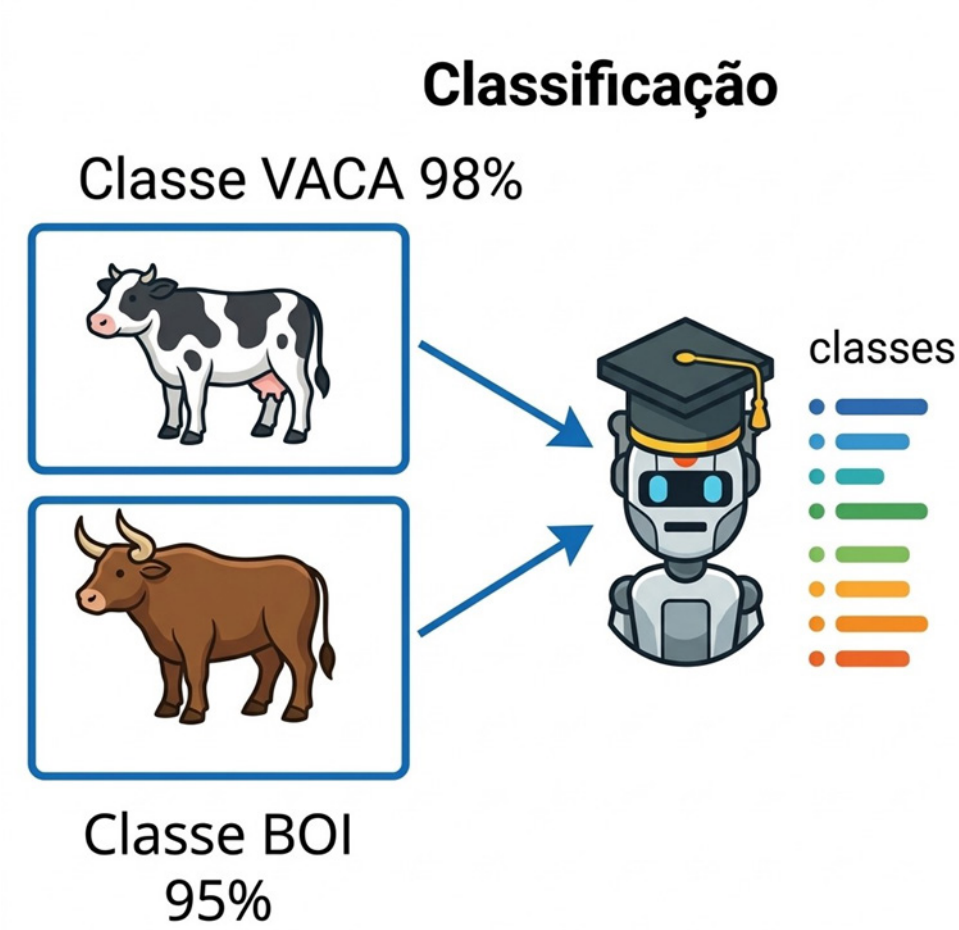
Figura 3: Subdivisão didática das tarefas de Aprendizado Supervisionado entre Classificação e Regressão.



- **Regressão Logística (Logistic Regression):** Modelo probabilístico linear adaptado para predição de variáveis categóricas discretas por meio da função sigmoide ou softmax.
- **Máquinas de Vetor de Suporte (Support Vector Machines – SVM):** Algoritmo que busca encontrar o hiperplano ótimo de separação que maximiza a margem de distância entre as diferentes classes.
- **Árvores de Decisão (Decision Trees):** Estruturas hierárquicas de decisão baseadas no particionamento recursivo dos dados a partir de regras condicionais.

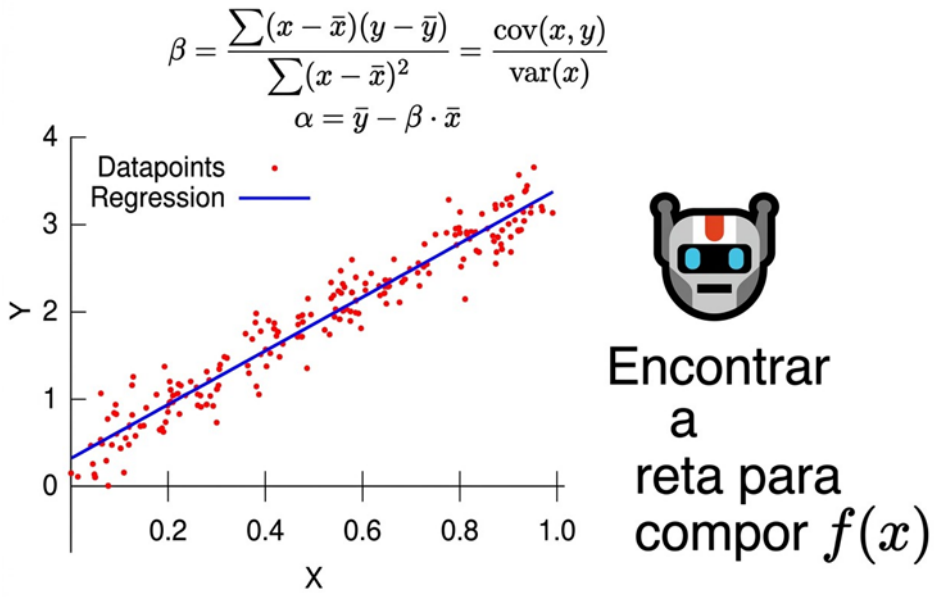
A Figura 4 ilustra graficamente o conceito de Classificação, destacando a separação e atribuição dos dados de entrada a categorias discretas pré-definidas.

Figura 4: Representação esquemática da tarefa de Classificação no Aprendizado Supervisionado: atribuição de entradas a classes categóricas discretas por meio de fronteiras de decisão.



Por sua vez, na Regressão, buscamos prever valores numéricos contínuos a partir das entradas fornecidas. Um exemplo clássico e amplamente difundido de regressão é a Regressão Linear, que busca encontrar uma função $f(x)$ representada por uma reta linear que mais se ajusta aos pontos dos dados no conjunto de treinamento, minimizando o erro quadrático entre os valores observados e as predições. A Figura 5 ilustra o princípio da Regressão Linear e o ajuste da reta $f(x)$ sobre o conjunto de dados.

Figura 5: Representação esquemática da tarefa de Regressão: ajuste da função preditiva $f(x)$ (reta de regressão) para estimar valores numéricos contínuos a partir dos dados de treinamento.

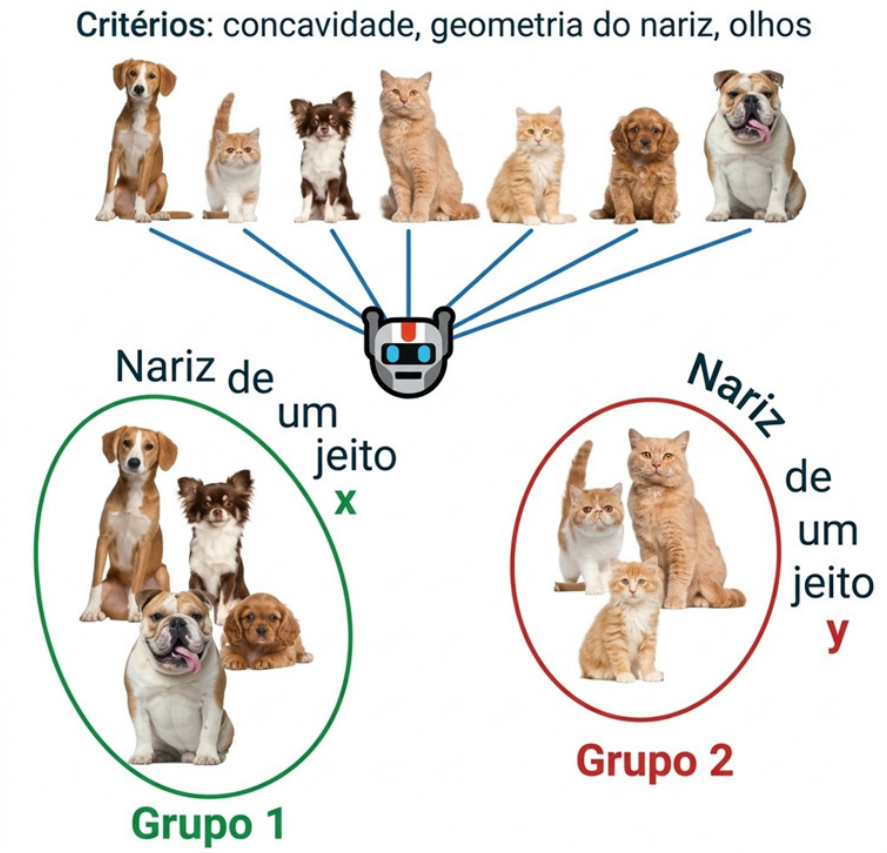


No contexto da Visão Computacional e da detecção de objetos com redes neurais YOLO, o Aprendizado Supervisionado combina harmoniosamente ambas as tarefas: cada imagem de treinamento (e_i) é acompanhada por caixas delimitadoras e rótulos de classe anotados por especialistas humanos (s_i). A rede neural aprende a função $f(x)$ executando a Classificação (Figura 4) para identificar a categoria de cada objeto e a Regressão (Figura 5) para prever com precisão as coordenadas numéricas contínuas das caixas delimitadoras.

3. O PARADIGMA DO APRENDIZADO NÃO SUPERVISIONADO

O Aprendizado Não Supervisionado (Unsupervised Learning) consiste na técnica de treinar uma máquina a partir de conjuntos de dados que não estão rotulados, ou seja, sem a presença de saídas desejadas previamente anotadas por um supervisor. Ao contrário do paradigma supervisionado, esses algoritmos buscam descobrir autonomamente informações, padrões e estruturas ocultas sobre os dados, sendo capazes de agrupá-los de acordo com suas semelhanças, correlações ou diferenças intrínsecas. Para ilustrar didaticamente esse conceito, considere um agente computacional incumbido de organizar um conjunto misto de imagens de cães e gatos via aprendizado não supervisionado. A partir de uma característica oculta dos objetos — como, por exemplo, o formato geométrico do nariz —, o algoritmo pode identificar padrões estatísticos nos pixels e agrupar esses objetos autonomamente: no Grupo X, reúne os espécimes com formato de nariz redondo, e no Grupo Y, os espécimes com formato de nariz triangular. A Figura 6 ilustra esquematicamente o fluxo conceitual do Aprendizado Não Supervisionado e o agrupamento autônomo de dados sem rótulos.

Figura 6: Representação esquemática do Aprendizado Não Supervisionado: o algoritmo analisa dados não rotulados para identificar padrões ocultos e agrupamentos por semelhança.



As técnicas mais comuns aplicadas ao Aprendizado Não Supervisionado são:

- **Agrupamento (Clustering):** Consiste em agrupar dados não rotulados com base em suas semelhanças ou diferenças, como exemplificado na separação por características anatômicas. As principais abordagens de agrupamento incluem:
  - **Agrupamentos Exclusivos:** Cada amostra pertence estritamente a um único grupo (ex.: K-Means).
  - **Agrupamentos Sobrepostos:** Uma amostra pode pertencer a múltiplos grupos simultaneamente com diferentes graus de pertinência.
  - **Agrupamentos Hierárquicos:** Estruturam os grupos em árvores hierárquicas de granularidade (dendrogramas).
  - **Agrupamentos Probabilísticos:** Modelam os dados por meio de distribuições estatísticas de probabilidade (ex.: Misturas Gaussianas).
- **Regras de Associação e Coocorrência:** Identificam relações e dependências frequentes entre variáveis em grandes volumes de dados.
- **Redução da Dimensionalidade:** Reduz o número de variáveis de entrada projetando os dados em espaços de menor dimensão enquanto retém a maior parte da variância original (ex.: PCA).

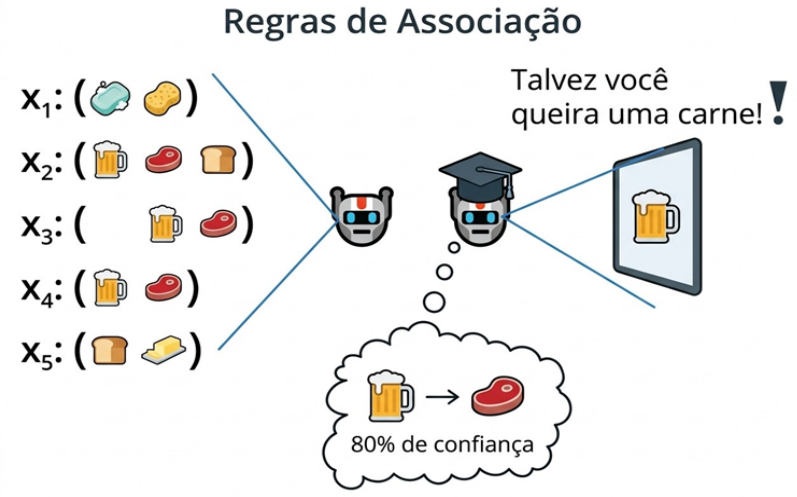
3.1 Regras de Associação e Análise de Coocorrência

Nas Regras de Associação (ou análise de coocorrência), busca-se encontrar relações, dependências e padrões frequentes entre os diferentes elementos contidos em uma base de dados. O objetivo primário dessa técnica de Aprendizado Não Supervisionado é identificar regras probabilísticas do tipo “Se {Item A} → Então {Item B}”, onde a ocorrência de determinado elemento indica uma elevada probabilidade de presença concomitante de outros objetos (coocorrência, e não causalidade estrita). Essa técnica é historicamente célebre por sua aplicação na análise de “cestas de compras” (market basket analysis) no varejo e no comércio eletrônico, permitindo descobrir, por exemplo, quais produtos são frequentemente adquiridos juntos por clientes sem que haja rótulos prévios orientando esse agrupamento. As Regras de Associação são muito utilizadas no processamento de grandes massas de dados. Elas fundamentam-se no levantamento de frequências a priori na base histórica para mapear e identificar as relações de coocorrência entre objetos. Os algoritmos mais famosos e representativos dessa categoria incluem:

- Apriori: Algoritmo pioneiro baseado na propriedade de que todos os subconjuntos não vazios de um conjunto de itens frequente também devem ser frequentes, reduzindo drasticamente a busca em grandes volumes de dados.
  - Eclat (Equivalence Class Transformation): Algoritmo que utiliza representações verticais de dados e interseção de listas de transações para acelerar a identificação de padrões de coocorrência.
  - FP-Growth (Frequent Pattern Growth): Algoritmo eficiente que compacta a base de dados em uma estrutura de árvore de padrões frequentes (FP-Tree), evitando a geração exaustiva de candidatos intermediários.

A Figura 7 ilustra esquematicamente o conceito de Regras de Associação e o mapeamento de coocorrências entre elementos.

Figura 7: Representação esquemática das Regras de Associação e coocorrência no Aprendizado Não Supervisionado: identificação de relações de dependência entre objetos.



4. O PARADIGMA DO APRENDIZADO POR REFORÇO

No Aprendizado por Reforço (Reinforcement Learning), pares pré-definidos de entradas e saídas rotuladas não são apresentados ao sistema. Em vez disso, o algoritmo busca aprender uma política (policy) de tomada de decisão que mapeia os estados do mundo às ações que o agente computacional deve executar para atingir determinado objetivo.

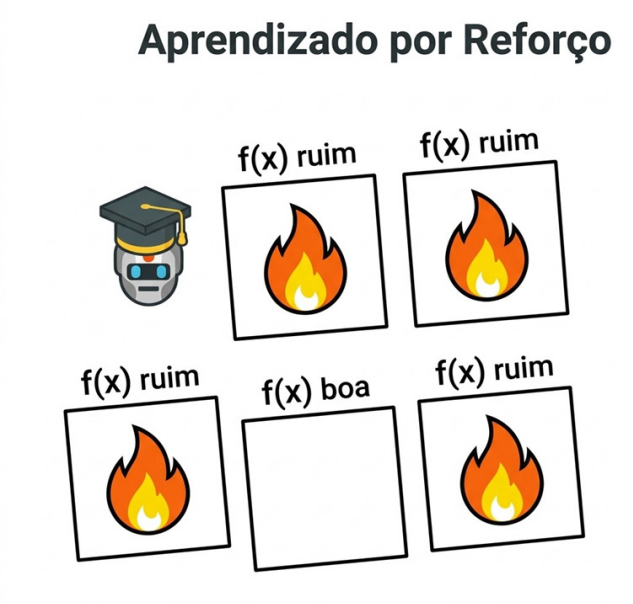
Essa técnica se preocupa diretamente em como um agente deve agir dentro de um ambiente dinâmico de forma a maximizar a recompensa acumulada ao longo do tempo. Os componentes centrais desse modelo de aprendizagem compreendem:

- **Agente:** É o elemento decisor alimentado pela inteligência artificial. O agente observa o contexto atual, realiza ações que alteram o estado do ambiente e toma decisões com base na expectativa de recompensas ou na prevenção de punições.
- **Ambiente**: É o mundo físico ou virtual no qual o agente opera, reagindo dinamicamente a cada ação tomada.
- **Estado (s):** A situação atual ou configuração do ambiente percebida pelo agente a cada instante.
- **Ação (a):** A intervenção executada pelo agente, que altera o estado do ambiente.
- **Feedback (Recompensa/Penalidade):** O sinal de retroalimentação escalar r_t emitido pelo ambiente. Ações bem-sucedidas geram recompensas positivas que impulsionam a tomada de decisão no sentido de favorecer comportamentos desejados; ações inadequadas geram penalidades (recompensas negativas).

Um bom exemplo intuitivo dessa técnica é a aprendizagem em um jogo eletrônico: a cada vez que o personagem morre ou cai em uma armadilha, o agente recebe uma penalidade (uma recompensa r_t negativa). Por outro lado, quando o agente avança para um quadrado seguro sem fogo ou atinge um objetivo, recebe um sinal r_t favorável (recompensa positiva). Dessa forma, o Aprendizado por Reforço aprende por tentativa e erro o que é benéfico (“bom”) e o que é prejudicial (“ruim”). A Figura 8 ilustra o ciclo contínuo de interação entre o agente e o ambiente no Aprendizado por Reforço.

No Aprendizado por Reforço, o sistema de Inteligência Artificial enfrenta situações dinâmicas sem gabarito prévio. Cabe ao modelo descobrir autonomamente como executar a tarefa para maximizar a recompensa total acumulada. Além de jogos eletrônicos e robótica, um exemplo proeminente dessa tecnologia são os **veículos autônomos**, que navegam em tempo real ajustando aceleração, frenagem e esterçamento com base nas condições do tráfego e nas regras de segurança do trânsito.

Figura 8: Representação do Aprendizado por Reforço: interação contínua entre o agente e o ambiente por meio de estados, ações e sinal de feedback (recompensas e penalidades).



5. VISÃO COMPUTACIONAL: DANDO PERCEPÇÃO VISUAL ÀS MÁQUINAS

Dentre as subáreas mais proeminentes da Inteligência Artificial, a Visão Computacional (VC) ocupa papel de destaque ao buscar dotar os computadores da capacidade de interpretar, analisar e compreender o mundo visual de maneira análoga ao sistema visual humano (SZELISKI, 2022). Enquanto o olho humano capta a luz refletida pelos objetos e a retino-córtex traduz os estímulos em percepção de formas, cores e distâncias, a Visão Computacional utiliza sensores ópticos (como câmeras digitais, sensores multiespectrais e sistemas Lidar) para converter a radiação eletromagnética em arranjos numéricos discretos. Didaticamente, para um sistema digital, uma imagem colorida não é percebida intuitivamente como uma cena visual, mas sim como uma matriz tridimensional de números. Em uma representação RGB padrão, a imagem é codificada nas dimensões $H \times W \times C$, onde H representa a altura em pixels, W a largura e C os três canais de cor (Vermelho, Verde e Azul). Cada elemento da matriz, denominado pixel (picture element), possui uma intensidade luminosa quantificada tipicamente em uma escala discreta de 0 a 255. O grande desafio da Visão Computacional reside em transpor o chamado “abismo semântico” (semantic gap) — isto é, a lacuna conceitual entre os valores numéricos brutos dos pixels e o significado abstrato e categórico da cena visual. Historicamente, as etapas de interpretação de imagens dividem-se em diferentes níveis de abstração e complexidade operacional:

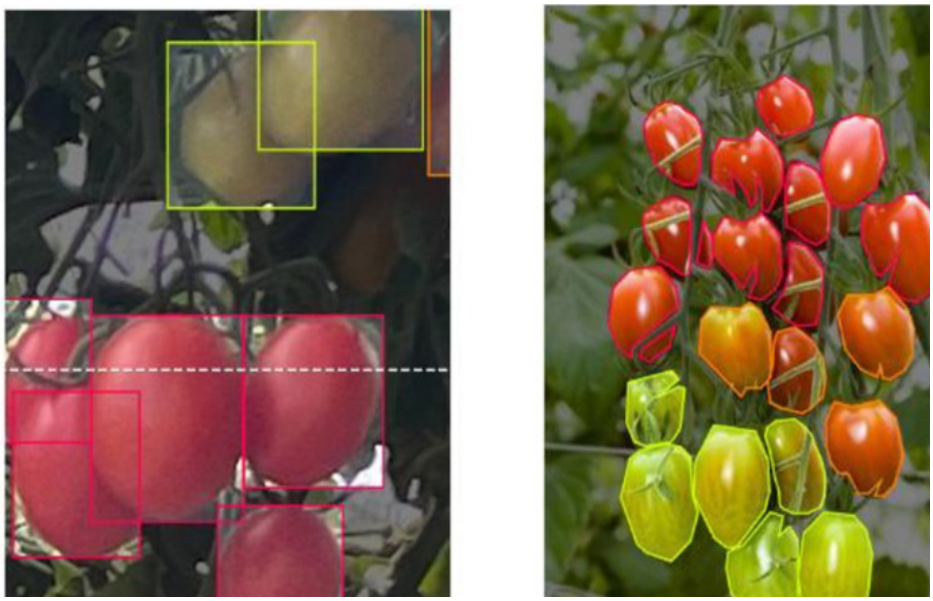
- **Processamento de Imagens (Baixo Nível):** Foco no aprimoramento da qualidade da imagem visual por meio de operações matriciais diretas, incluindo filtragem de ruído, ajuste de contraste, binarização por limiarização e detecção de bordas baseada em gradientes (como os operadores de Sobel e Canny).

- **Análise e Extração de Atributos (Médio Nível):** Identificação e agrupamento de primitivas geométricas, contornos, texturas e descritores locais pré-projetados (como SIFT, SURF e HOG) para identificação de pontos de interesse.
- **Compreensão Semântica (Alto Nível):** Atribuição de sentido e interpretação categórica aos elementos da cena. É neste patamar que atuam os algoritmos de Aprendizado Profundo. No âmbito do reconhecimento visual de alto nível, destacam-se três tarefas fundamentais: a Classificação de Imagens, que atribui um rótulo global a uma imagem inteira; a Detecção de Objetos, que identifica a presença de múltiplos alvos em uma cena, delimitando suas posições exatas por meio de caixas delimitadoras (bounding boxes) associadas a classes categóricas; e a Segmentação por Instância, que delineia o contorno exato de cada objeto ao nível de pixel.

6. MODALIDADES E TIPOS DE ROTULAÇÃO EM VISÃO COMPUTACIONAL

Para que algoritmos de Aprendizado Profundo consigam extrair atributos relevantes e aprender a reconhecer alvos visuais com elevada acurácia, o processo de anotação de dados (labeling ou rotulação) desempenha um papel fundamental. A rotulação consiste em associar informações semânticas estruturadas (rótulos de classe, coordenadas geométricas ou máscaras de pixels) às imagens brutas fornecidas aos modelos de aprendizado supervisionado. Dependendo da complexidade da aplicação e do nível de granularidade exigido pela tarefa computacional, a anotação visual pode ser categorizada em diferentes tipos de rotulação. A Figura 9, apresentada no estudo de Araújo et al. (2025), sintetiza didaticamente as principais modalidades de rotulação utilizadas na construção de conjuntos de dados para Visão Computacional.

Figura 9: Principais modalidades e tipos de rotulação de dados em Visão Computacional: Classificação de Imagem, Detecção de Objetos com Caixas Delimitadoras (Bounding Boxes), Segmentação Semântica e Segmentação por Instância.



Fonte: Araújo et al. (2025).

As quatro modalidades fundamentais de rotulação destacadas na literatura e sistematizadas por Araújo et al. (2025) compreendem:

1. **Classificação de Imagem (Rotulação Global):** A anotação atribui uma única classe categórica para a imagem inteira (por exemplo, classificar uma imagem como “Tomate Maduro” ou “Tomate Verde”), sem indicar a posição espacial ou a quantidade de objetos presentes na cena.
2. **Detecção de Objetos com Caixas Delimitadoras (Bounding Boxes):** A rotulação é realizada desenhando retângulos envolventes ortogonais em torno de cada objeto de interesse. Cada caixa é representada por coordenadas (x_{centro}, y_{centro}, w, h) associadas a uma classe. No trabalho de Araújo et al. (2025), essa técnica foi empregada para delimitar individualmente cada tomate cereja no tomateiro e classificar seus estágios de maturação utilizando modelos da família YOLO.
3. **Segmentação Semântica (Rotulação por Pixels):** Todos os pixels da imagem pertencentes a uma mesma categoria recebem a mesma etiqueta de classe (por exemplo, delimitar a área ocupada pela folhagem ou pelo solo). Não diferencia instâncias individuais pertencentes à mesma classe.
4. **Segmentação por Instância (Rotulação por Polígonos/Máscaras):** Combina a detecção de objetos com o detalhamento ao nível de pixel. Cada objeto é delimitado por um polígono exato que acompanha seus contornos geométricos, permitindo separar individualmente diferentes instâncias da mesma categoria.

A escolha do tipo de rotulação equilibra a precisão requerida e o esforço computacional e humano de anotação. Na detecção de objetos em tempo real com a arquitetura YOLO, o uso de caixas delimitadoras oferece a relação ideal entre riqueza espacial e eficiência de treinamento.

7. A REVOLUÇÃO DAS REDES NEURAIS CONVOLUCIONAIS E DO FRAMEWORK YOLO

A emergência das Redes Neurais Convolucionais (Convolutional Neural Networks – CNNs) revolucionou a Visão Computacional moderna ao substituir a extração artesanal de atributos pelo aprendizado automático de características visuais (LECUN; BENGIO; HINTON, 2015). As CNNs utilizam camadas de convolução compostas por filtros matemáticos deslizantes que preservam a topologia bidimensional da imagem. Nas camadas iniciais, os filtros aprendem a detectar atributos de baixa complexidade, como bordas horizontais e verticais. Nas camadas intermediárias e profundas, tais atributos são combinados para formar conceitos visuais refinados, como texturas, partes de objetos e geometrias complexas. Apesar da alta precisão atingida pelas primeiras arquiteturas de detecção baseadas em regiões (como a família R-CNN, Fast R-CNN e Faster R-CNN), esses modelos operavam em duas etapas distintas: primeiramente propunham regiões de

interesse na imagem e, em seguida, classificavam cada região isoladamente. Esse processo em dois estágios resultava em alto custo computacional, inviabilizando o processamento de vídeos em tempo real. Essa limitação foi superada com a introdução da família de modelos YOLO (You Only Look Once), concebida por Redmon et al. (2016). O framework YOLO redefiniu o problema de detecção de objetos ao formulá-lo não como uma busca exaustiva por regiões, mas como um único problema de regressão direta. Em uma única passagem direta (forward pass) da imagem pela rede neural, o algoritmo divide a imagem de entrada em uma grade regular e prevê simultaneamente as coordenadas das caixas delimitadoras, os índices de confiança e as probabilidades de classe para todos os objetos presentes. Essa arquitetura unificada proporcionou um ganho inédito de velocidade sem comprometer substancialmente a acurácia, viabilizando o uso da Visão Computacional em tempo real em dispositivos de borda e ambientes de produção.

8. O PROCESSO DE TREINAMENTO E INFERÊNCIA EM REDES NEURAIS CONVOLUCIONAIS

O ciclo de vida de uma Rede Neural Convolucional no contexto da Visão Computacional divide-se didaticamente em duas fases essenciais e complementares: o Treinamento (Training) e a Inferência (Inference). O Treinamento é a etapa de aprendizado supervisionado na qual o modelo computacional ajusta internamente seus milhões de parâmetros livres — denominados pesos (weights) e viéses (biases) — a partir da apresentação repetida de um conjunto de dados rotulado. Esse processo segue um fluxo iterativo composto por quatro etapas encadeadas:

1. **Passagem Direta (Forward Pass):** A imagem rotulada de treinamento é alimentada na rede, que realiza sucessivas operações de convolução, ativação e reduçãoespacial, gerando uma predição inicial de caixas delimitadoras e classes.
2. **Cálculo da Função de Perda (Loss Function):** Uma função matemática quantifica a discrepância (erro) entre a resposta prevista pela rede e o gabarito real (ground truth) anotado pelo supervisor humano.
3. **Retropropagação do Erro (Backpropagation):** Utilizando a regra da cadeia do cálculo diferencial, a rede calcula a contribuição de cada peso e viés interno no erro total, propagando os gradientes de trás para frente a partir da camada de saída até as camadas iniciais (LECUN; BENGIO; HINTON, 2015).
4. **Otimização de Parâmetros (Optimizer):** Algoritmos de otimização (como SGD, Adam ou AdamW) atualizam suavemente o valor de cada peso e viés na direção oposta ao gradiente do erro, buscando minimizar a função de perda ao longo de múltiplas épocas (epochs) de treinamento.

Por sua vez, a Inferência é a fase operacional prática na qual a rede neural — com seus parâmetros internos otimizados e congelados — é implantada no ambiente de produção

para realizar predições em imagens inéditas e vídeos em tempo real. Na inferência, o algoritmo executa apenas a passagem direta (forward pass), sem realizar cálculo de perda ou atualização de parâmetros, resultando em um tempo de resposta extremamente veloz (frequentemente medido em milissegundos por quadro). A Figura 10, oriunda do trabalho empírico desenvolvido por Araújo et al. (2025), exemplifica o resultado da fase de inferência de uma rede neural YOLO pré-treinada para a detecção e contagem de tomates cereja. Na inferência ilustrada na Figura 10, a rede neural treinada por Araújo et al. (2025) recebe uma imagem do tomateiro em ambiente agrícola e desenha instantaneamente as caixas delimitadoras (bounding boxes) sobre cada fruto individual, atribuindo simultaneamente a classe correspondente ao seu estágio de amadurecimento (verde, alaranjado/virado ou maduro) e a porcentagem de confiança da predição.

9. APLICAÇÕES CIENTÍFICAS E CONTRIBUIÇÕES EMPÍRICAS NA FRONTEIRA DO CONHECIMENTO

A maturidade alcançada pelos algoritmos de Visão Computacional baseados na família YOLO abriu novos horizontes para a solução de desafios complexos e interdisciplinares, com relevância direta para a agricultura de precisão, a ecologia e a conservação da biodiversidade. As pesquisas conduzidas por Sergio Souza Novak e colaboradores exemplificam a versatilidade e o impacto científico da aplicação de redes convolucionais YOLO em cenários reais de monitoramento automatizado. Na proteção fitossanitária e na agricultura 4.0, a identificação precoce de pragas em lavouras é essencial para mitigar perdas produtivas e otimizar a aplicação de defensivos agrícolas. Demonstrando a viabilidade do processamento inteligente na borda do campo, Almeida et al. (2024) desenvolveram o sistema AgroInsect, uma arquitetura híbrida que integra redes neurais de Visão Computacional executadas em dispositivos de borda (edge computing) com processamento em nuvem. A solução combina a detecção automatizada de insetos-praga em tempo real com métodos de interpolação geoestatística, permitindo mapear espacialmente a infestação no campo e fundamentar tomadas de decisão agrícolas precisas e sustentáveis. No campo do monitoramento faunístico e da conservação da biodiversidade, a Visão Computacional atua como uma ferramenta não invasiva de elevada eficácia. Ao avaliar o desempenho de diferentes versões do modelo YOLO na fauna silvestre, Barros et al. (2024) realizaram uma análise comparativa abrangente de culminou no desenvolvimento e na implantação do software TortoiseVision Detect, demonstrando como algoritmos de detecção em tempo real podem ser embarcados em ferramentas computacionais aplicadas à biologia da conservação e ao manejo de répteis. A automação do controle de qualidade e a estimativa de maturação em produtos hortícolas também se beneficiam diretamente dessas tecnologias. No cultivo de tomateiro cereja, Araújo et al. (2025) conduziram uma avaliação comparativa de modelos YOLO aplicados à classificação e quantificação automatizada dos estágios de amadurecimento dos frutos. Os resultados evidenciaram a capacidade das redes convolucionais em discriminar nuances sutis de coloração e morfológicas no fruto, permitindo automatizar processos pós-colheita que tradicionalmente exigiam triagem manual morosa

e sujeita à subjetividade humana. Por fim, no estudo da biodiversidade e dos bioindicadores ambientais, a detecção de organismos de pequeno porte e movimento rápido representa um desafio extremo para sistemas visuais. Enfrentando essa complexidade, França et al. (2025) aplicaram a rede neural convolucional YOLOv8 para a detecção automatizada de abelhas nativas do Cerrado brasileiro. O trabalho comprovou que mesmo em fundos vegetais complexos e sob variações de iluminação natural, a arquitetura YOLOv8 apresenta elevada capacidade de generalização para identificar polinizadores fundamentais para a manutenção dos ecossistemas neotropicais.

Figura 10: Exemplo prático do resultado da fase de inferência de um modelo YOLO: detecção e classificação automatizada dos estágios de amadurecimento em tomates cereja em tempo real.



Fonte: Araújo et al. (2025).

Arquiteturas neurais para a identificação e contagem automatizada de jabutis.

10. SÍNTESE DO CAPÍTULO E ESTRUTURAÇÃO DO LIVRO

Este primeiro capítulo contextualizou didática e cientificamente os fundamentos da Inteligência Artificial, do Aprendizado Supervisionado, Não Supervisionado, por Reforço e da Visão Computacional, demonstrando como a evolução dos algoritmos — desde as abordagens de processamento de imagem, rotulação, treinamento e inferência até a arquitetura YOLO — viabilizou soluções automatizadas de alto impacto no mundo real. As pesquisas empíricas citadas comprovam que a integração entre a teoria da computação e os problemas práticos de campo é o caminho para o desenvolvimento de soluções agrícolas e ambientais de ponta. Ao longo das próximas seções e capítulos deste livro, o leitor será introduzido passo a passo aos detalhes práticos e teóricos necessários para dominar a família YOLO: desde a coleta e anotação de conjuntos de dados (datasets), passando pelo pré-processamento de imagens, arquiteturas de redes convolucionais, métodos de

treinamento supervisionado, métricas de avaliação de desempenho (como mAP, Precisão e Revocação), até o embarque e implantação de sistemas de Visão Computacional em produção.

REFERÊNCIAS

ALMEIDA, Guilherme Pires Silva de et al. AgroInsect: A hybrid AI-driven edge-cloud architecture for precision pest detection and geostatistical interpolation in agriculture. Computers and Electronics in Agriculture, v. 256, p. 112386, 2024.

ARAÚJO, Amanda Beatriz Martins et al. Avaliação comparativa de modelos YOLO para classificação e quantificação automatizadas de estágios de amadurecimento de tomates cereja. Brazilian Journal of Development, v. 11, e81856, 2025.

BARROS, Gustavo Moura et al. Avaliação comparativa de arquiteturas YOLO para identificação de jabutis e implementação do software TortoiseVision Detect. Revista de Derecho y Cambio Social, v. 23, e5475, 2024.

FRANÇA, Elzi Leida do Carmo et al. Detecção de abelhas nativas do Cerrado Brasileiro com Rede Neural Convolucional YOLOv8. Caderno Pedagógico, v. 22, n. 11, e19970, 2025. DOI: 10.54033/cadpedv22n11-207.

LECUN, Yann; BENGIO, Yoshua; HINTON, Geoffrey. Deep Learning. Nature, v. 521, n. 7553, p. 436–444, 2015.

REDMON, Joseph et al. You Only Look Once: Unified, Real-Time Object Detection. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2016. P. 779–788.

RUSSELL, Stuart; NORVIG, Peter. Artificial Intelligence: A Modern Approach. 3rd. Upper Saddle River: Pearson, 2010.

SZELISKI, Richard. Computer Vision: Algorithms and Applications. 2nd. Cham: Springer Nature, 2022.

TURING, Alan M. Computing Machinery and Intelligence. Mind, v. 59, n. 236, p. 433–460, 1950.

CAPÍTULO 2

EVOLUÇÃO E ARQUITETURA DA FAMÍLIA YOLO

1. CONTEXTO HISTÓRICO DA VISÃO COMPUTACIONAL: DO INVERNO DA IA À REVOLUÇÃO DO DEEP LEARNING

1.1 Os Primórdios das Redes Convolucionais e as Limitações da Era Inicial

Nas primeiras décadas de desenvolvimento da Inteligência Artificial, o treinamento de redes neurais artificiais com múltiplas camadas profundas enfrentava severas barreiras matemáticas e computacionais. O ajuste iterativo dos parâmetros por meio do algoritmo de retropropagação do erro (backpropagation) demonstrava forte instabilidade em arquiteturas complexas, frequentemente colapsando devido ao fenômeno do desvanecimento do gradiente (vanishing gradient). Nesse processo, os sinais de correção calculados a partir da camada de saída tornavam-se infinitesimalmente pequenos à medida que retropropagavam pelas camadas iniciais, impedindo o aprendizado efetivo. Somada à extrema escassez de dados anotados e ao reduzido poder de processamento dos computadores da época, essa limitação conduziu a comunidade científica a um período de relativo ceticismo e estagnação de investimentos conhecido como o “Inverno da IA” (AI Winter). Contudo, as Redes Neurais Convolucionais (Convolutional Neural Networks – CNNs) não surgiram subitamente com as grandes competições da década de 2010; seus fundamentos teóricos e estruturais foram estabelecidos ainda na década de 1980. Inspirado nas descobertas neurofisiológicas sobre a organização do córtex visual biológico de mamíferos (HUBEL; WIESEL, 1959, 1962), Fukushima (1980) introduziu a arquitetura do Neocognitron. Esse modelo pioneiro formalizou os conceitos de campos receptivos locais — nos quais cada unidade de processamento analisa apenas um fragmento espacial restrito da imagem de entrada — e de compartilhamento de pesos entre neurônios, reduzindo drasticamente a quantidade de parâmetros a serem ajustados em comparação com redes completamente conectadas. O salto prático subsequente ocorreu em 1989, quando Yann LeCun e seus colaboradores demonstraram de forma inédita que o algoritmo de backpropagation podia ser utilizado para aprender autonomamente os coeficientes dos filtros convolucionais diretamente a partir dos pixels brutos de imagens de dígitos manuscritos (LECUN; BOSER et al., 1989). Essa contribuição evoluiu na década de 1990 para o desenvolvimento da famosa arquitetura LeNet-5 (LECUN; BOTTOU et al., 1998), que obteve sucesso em aplicações comerciais de automação bancária e de triagem postal nos Estados Unidos, realizando a leitura automatizada de códigos de endereçamento (ZIP codes) e numerações de cheques. A despeito dessa demonstração bem-sucedida em tarefas de domínio restrito, a abordagem baseada em convoluções estagnou temporariamente ao final dos anos 1990. O grande obstáculo da época residia na impossibilidade de escalar a metodologia para a percepção de imagens naturais complexas e de alta resolução. A capacidade de processamento das unidades centrais de processamento (CPUs) existentes e a ausência de grandes acervos

de dados anotados inviabilizavam o treinamento de redes convolucionais com capacidade suficiente para atuar no mundo real.

1.2 O Desafio da Visão Real e o Paradigma da Engenharia Manual de Atributos

Diante da inviabilidade computacional de treinar redes neurais profundas ao longo dos anos 2000, a pesquisa em Visão Computacional voltou-se para a busca de métodos alternativos capazes de responder à indagação central do domínio: como reconhecer e localizar categorias genéricas de objetos em imagens reais do mundo cotidiano? Para contornar as limitações de aprendizado direto da imagem bruta, estabeleceu-se o paradigma da engenharia manual de características visuais (handcrafted feature engineering). Nessa abordagem tradicional, a interpretação visual dependia obrigatoriamente da intervenção de especialistas humanos, que projetavam matematicamente algoritmos dedicados à extração de descritores invariantes a mudanças de iluminação, escala e orientação. O fluxo operacional dividia-se rigidamente em duas etapas desconectadas: a imagem de entrada era submetida a algoritmos de extração manual de atributos — tais como o Scale-Invariant Feature Transform (SIFT), o Histogram of Oriented Gradients (HOG), o Local Binary Patterns (LBP) ou a representação por Fisher Vectors — e os vetores numéricos resultantes eram fornecidos a um classificador estatístico independente, tipicamente uma Máquina de Vetor de Suporte (Support Vector Machines – SVM). O avanço desse paradigma foi impulsionado pelo estabelecimento do desafio PASCAL VOC (Visual Object Classes Challenge) a partir de 2005 (EVERINGHAM et al., 2010). O PASCAL VOC desempenhou um papel divisor ao consolidar um benchmark padronizado para a comunidade internacional, integrando um acervo de imagens rotuladas, métricas formais de avaliação — como a Precisão Média (mean Average Precision – mAP) — e uma competição científica anual para tarefas de classificação, localização e detecção de objetos. Entretanto, como a extração de atributos dependia da intuição e do projeto humano, os sistemas baseados em descritores manuais encontravam dificuldades para generalizar o aprendizado quando submetidos a variações de oclusão, fundo dinâmico e deformação morfológica dos objetos.

1.3 A Infraestrutura do Big Data Visual: ImageNet e ILSVRC

À medida que a capacidade computacional evoluía no final da década de 2000, a escassez de dados padronizados em grande escala manteve-se como o gargalo principal para o avanço dos modelos de aprendizado. Em 2009, uma equipe de pesquisadores liderada pela professora Fei-Fei Li apresentou o projeto ImageNet (DENG et al., 2009), propondo a criação de um acervo visual massivo estruturado de acordo com a hierarquia taxonômica do banco léxico WordNet. O ImageNet superou as dimensões de todos os conjuntos de dados pré-existentes, reunindo milhões de imagens cuidadosamente rotuladas em milhares de categorias conceituais. Com base nessa infraestrutura de dados, foi instituído em 2010 o ImageNet Large Scale Visual Recognition Challenge (ILSVRC), que se consagrou como a competição de referência global na área de Visão Computacional.

O ILSVRC estabeleceu um ambiente competitivo anual em escala inédita, avaliando algoritmos sob mais de 1.000 categorias distintas de objetos e ultrapassando 1,4 milhão de imagens de treinamento. O desafio estruturava-se em três modalidades principais de avaliação: a classificação global da imagem, a localização da caixa delimitadora do objeto principal e a detecção de múltiplos objetos presentes na cena.

1.4 A Ruptura Tecnológica da AlexNet em 2012

Até o ano de 2011, as edições do ILSVRC eram amplamente dominadas por modelos baseados na evolução incremental da engenharia de características tradicionais, combinando descritores SIFT refinados, codificação por Fisher Vectors e classificadores SVM. Embora precisos para a época, os ganhos anuais de desempenho tornavam-se progressivamente menores, indicando a saturação dos métodos de extração manual. A ruptura definitiva desse cenário ocorreu no desafio ILSVRC de 2012, quando uma equipe da Universidade de Toronto — integrada por Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton — inscreveu uma Rede Neural Convolucional profunda de oito camadas, denominada AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Os resultados da competição revelaram um salto de desempenho sem precedentes na história do processamento de imagens: enquanto o segundo colocado no desafio — utilizando as técnicas mais avançadas de engenharia manual de atributos — obteve uma taxa de erro top-5 de aproximadamente 26,2%, a AlexNet alcançou a marca de 15,3%. A margem de superação de 10,8 pontos percentuais não representou uma simples otimização incremental, mas a demonstração empírica da superioridade do aprendizado profundo de representações sobre os descritores manuais.

1.5 Fundamentos da AlexNet e a Transição Conceitual para o Aprendizado End-to-End

O feito extraordinário da AlexNet viabilizou-se pela convergência síncrona de três elementos indispensáveis: a formulação de uma arquitetura convolucional profunda com cerca de 60 milhões de parâmetros, o volume massivo de amostragem fornecido pelo ImageNet e o emprego pioneiro de unidades de processamento gráfico (GPUs NVIDIA) programadas via CUDA. A aceleração por GPU permitiu realizar o treinamento das operações matriciais em poucos dias, tarefa que exigiria meses em processadores convencionais (CPUs). Mais do que o ganho estatístico, a AlexNet promoveu uma mudança conceitual profunda no desenvolvimento da inteligência visual. No paradigma anterior a 2012, o fluxo analítico dependia da seleção prévia de características pelo pesquisador, seguida pelo treinamento isolado do classificador. Com o advento das redes convolucionais profundas, o processo passou a ser operado de ponta a ponta (end-to-end), cabendo à própria rede extrair automaticamente as representações relevantes diretamente da matriz de pixels da imagem. Esse aprendizado ocorre de maneira hierárquica ao longo da profundidade da rede: nas camadas convolucionais iniciais, os filtros aprendem de forma autônoma a

detectar primitivas visuais simples, tais como bordas direcionadas, gradientes de cor e texturas locais; nas camadas intermediárias, essas primitivas são combinadas para compor geometrias, contornos e partes reconhecíveis de objetos; e nas camadas finais e mais profundas, a rede sintetiza estruturas semânticas complexas e altamente abstratas, como rostos, estruturas corporais, componentes mecânicos ou silhuetas de animais. Dessa forma, a necessidade de especificação manual de atributos visuais foi definitivamente eliminada.

1.6 A Linha do Tempo da Visão Computacional e a Evolução rumo à Detecção em Tempo Real

Embora o impacto inicial da AlexNet tenha se tornado célebre na classificação de imagens, a competição ILSVRC contemplava de forma integrada o desafio de localização e detecção de objetos. A retrospectiva oficial da competição evidencia que 2012 marcou o ponto de inflexão a partir do qual as redes neurais profundas foram expandidas da mera atribuição de rótulos globais para a delimitação espacial de múltiplos objetos em cenas complexas. Essa evolução histórica estende-se em um contínuo que interliga marcos teóricos e práticos: iniciando na proposição dos campos receptivos do Neocognitron por Fukushima em 1980; passando pelas primeiras aplicações de backpropagation convolucional por LeCun em 1989 e 1998; avançando pela padronização de métricas do PASCAL VOC em 2005 e pela estruturação do ImageNet em 2009 e ILSVRC em 2010; perpassando o ápice dos métodos manuais em 2011 e a ruptura da AlexNet em 2012; culminando no surgimento das arquiteturas de detecção por regiões de interesse, como a R-CNN de Girshick et al. (2014) em 2014, e na introdução das redes residuais profundas (ResNet) por He et al. (2016) em 2015. Esse percurso culminou na reformulação das três indagações fundamentais que acompanharam a história da disciplina:

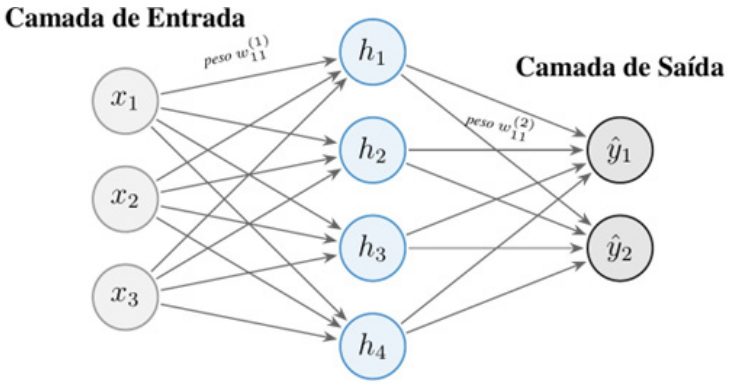
1. Na Era Tradicional (pré-2012), a questão central era: como projetar manualmente uma função matemática capaz de descrever as características de um objeto?
2. Na Era do Aprendizado Profundo em dois estágios (2012–2015), a indagação evoluiu para: como fazer a rede neural aprender autonomamente a extrair atributos para localizar e classificar regiões de interesse na imagem?
3. Na Era do Processamento em Tempo Real (2016–presente), a busca concentrou-se em: como realizar a extração de características, a localização espacial e a classificação categórica simultaneamente em uma única passagem direta, atingindo velocidade suficiente para operação em tempo real?

É precisamente para responder a essa terceira questão que a família de arquiteturas YOLO (You Only Look Once), concebida por Redmon, Divvala et al. (2016), foi desenvolvida e passou por sucessivas evoluções tecnológicas, conforme detalhado nas seções a seguir.

2. O QUE É UMA REDE NEURAL ARTIFICIAL: ESTRUTURA, GRAFOS, PESOS E APRENDIZADO

Para compreender a fundo o funcionamento de redes convolucionais complexas como as da família YOLO, é fundamental estabelecer primeiramente o conceito básico de uma **Rede Neural Artificial** (Artificial Neural Network – ANN). Uma Rede Neural Artificial é um modelo matemático e computacional inspirado na estrutura biológica do cérebro humano, projetado para reconhecer padrões, aprender relações complexas e realizar inferências a partir de dados de entrada. Do ponto de vista da teoria dos grafos, uma rede neural é representada como um **grafo direcionado ponderado**, composto por vértices (denominados neurônios ou nós) e arestas direcionadas que conectam esses neurônios em uma topologia organizada por camadas. A Figura 11 ilustra esquematicamente a arquitetura em grafo de uma rede neural artificial simples do tipo Perceptron Multicamadas (Multilayer Perceptron – MLP).

Figura 11: Representação de uma Rede Neural Artificial como um grafo direcionado ponderado: a informação flui da Camada de Entrada através da Camada Oculta até a Camada de Saída pelas arestas associadas a pesos computacionais (w_{ij}).



2.1 Anatomia do Grafo Neural: Camadas, Arestas, Pesos e Viéses

Conforme apresentado no grafo da Figura 11, a arquitetura de uma rede neural é constituída por três elementos matemáticos primordiais:

1. **Camadas (Layers):**
   - Camada de Entrada (Input Layer): É a primeira camada do grafo, responsável por receber os valores numéricos brutos não processados (x_1, x_2, . . . , x_n), tais como a intensidade de pixels de uma imagem ou características quantitativas.
   - Camadas Ocultas (Hidden Layers): São as camadas intermediárias localizadas entre a entrada e a saída. Nessas camadas, cada neurônio combina as informações recebidas das arestas anteriores e aplica transformações não lineares, permitindo extrair representações abstratas e padrões de ordem superior a partir dos dados.
   - Camada de Saída (Output Layer): É a camada final que entrega as respostas

ou predições calculadas pelo modelo ($\hat{y}_1$, $\hat{y}_2$), tais como as probabilidades de pertencimento a uma determinada classe categórica ou as coordenadas de uma caixa delimitadora.

2. Arestas Conectivas e Pesos (Weights – w_{ij}): As arestas são os canais direcionados que conectam os neurônios de uma camada aos neurônios da camada subsequente. Cada aresta possui associado a si um parâmetro numérico escalar denominado **peso** (w_{ij}). O peso quantifica a intensidade, força ou grau de relevância da conexão entre o neurônio ***i*** da camada anterior e o neurônio ***j*** da camada posterior. Se o peso for um valor positivo elevado, o sinal de entrada é fortemente amplificado; se for próximo de zero, o sinal é atenuado ou desconsiderado; se for negativo, a conexão exerce efeito inibitório.

3. Viéses (Biases – b_j): Além dos pesos, cada neurônio das camadas ocultas e de saída possui um termo ajustável independente denominado **viés** (b). O viés atua como uma constante de deslocamento que permite alterar o limiar de disparo do neurônio, garantindo flexibilidade para que ele consiga ser ativado mesmo quando todas as entradas numéricas resultarem em zero. Matematicamente, cada neurônio ***j*** em uma camada oculta calcula primeiramente o somatório ponderado de suas entradas somado ao seu viés:

$$z_j = \sum_{i=1}^{n} w_{ij} x_i + b_j \qquad (1)$$

Em seguida, o valor escalar resultante z_j é submetido a uma **função de ativação não linear** $\sigma(z)$ (como a função ReLU, Sigmoide ou Tanh), produzindo o sinal de saída do neurônio $a_j = \sigma(z_j)$ que será transmitido pelas arestas da camada seguinte.

2.2 Por que os Pesos Precisam ser Atualizados?

Quando uma rede neural é criada e inicializada, seus parâmetros internos — isto é, todos os pesos w_{ij} de suas arestas e os viéses b_j de seus neurônios — recebem valores numéricos aleatórios (ou pseudo-aleatórios pequenos). Consequentemente, durante as primeiras passagens do modelo, as respostas produzidas na camada de saída ($\hat{y}$) são completamente incorretas e distantes do gabarito real (y). A essência do **aprendizado de máquina** reside no processo iterativo de ajuste e calibração desses parâmetros. Os pesos precisam ser continuamente atualizados durante a fase de treinamento pelos seguintes motivos centrais:

- **Minimização do Erro de Predição:** Os pesos iniciais geram um erro elevado, quantificado matematicamente pela **Função de Perda** (L). Atualizar os pesos é o único meio de alterar a sensibilidade dos neurônios para fazer a resposta prevista $\hat{y}$ se aproximar do valor real y.
- **Aprendizado de Padrões e Generalização:** Cada peso ajustado reflete a importância atribuída a uma característica visual específica. À medida que os pesos são refinados

ao longo de múltiplas épocas de treinamento, a rede deixa de emitir palpites aleatórios e passa a "memorizar" a combinação ideal de atributos (como bordas, contornos e geometrias) necessária para classificar e localizar objetos em imagens inéditas.

- **Mecanismo da Retropropagação (Backpropagation):** A atualização é orientada pelo cálculo dos gradientes do erro em relação a cada peso ($\partial L/\partial w_{ij}$) via regra da cadeia. Um algoritmo de otimização (como o Gradiente Descendente / SGD ou Adam) modifica suavemente o valor de cada peso na direção oposta ao gradiente:

$$w_{ij}^{(novo)} = w_{ij}^{(antigo)} - \eta \cdot \frac{\partial L}{\partial w_{ij}} \qquad (2)$$

onde η representa a taxa de aprendizado (learning rate). Sem essa atualização sistemática dos pesos e viéses, a rede neural seria apenas um grafo estático sem capacidade adaptativa. É precisamente por meio da modificação progressiva desses pesos que a rede adquire inteligência computacional, permitindo que arquiteturas avançadas — como os modelos da família YOLO — aprendam a detectar objetos com elevada precisão em tempo real.

3. CONCEITO FUNDAMENTAL E A REVOLUÇÃO DO YOLO

O desenvolvimento da visão computacional moderna foi profundamente transformado pela introdução da arquitetura YOLO (You Only Look Once), concebida originalmente por Redmon, Divvala et al. (2016). Até o surgimento dessa abordagem, a tarefa de detecção de objetos em imagens dependia majoritariamente de sistemas em duas etapas (two-stage detectors), como a família R-CNN (Region-based Convolutional Neural Networks). Nesses modelos tradicionais, um primeiro módulo gerava propostas de regiões de interesse (region proposals) e, em seguida, uma rede convolucional dedicada classificava e ajustava as caixas de cada região proposta individualmente. Embora precisos, tais métodos apresentavam alto custo computacional e latência elevada, inviabilizando aplicações em tempo real. O YOLO revolucionou esse cenário ao formular a detecção de objetos como um único problema de regressão direta, operando de ponta a ponta (end-to-end). Em vez de segmentar a imagem em múltiplos sub-processos, uma única rede neural convolucional analisa a imagem inteira de uma só vez. Nessa passagem única, o modelo divide a imagem de entrada em uma grade (grid) e prevê simultaneamente as coordenadas das caixas delimitadoras (bounding boxes), as pontuações de confiança (confidence scores) e as probabilidades de cada classe para todos os objetos presentes (REDMON; DIVVALA et al., 2016). Essa arquitetura integrada proporcionou um ganho drástico de velocidade de processamento, permitindo que a detecção ocorresse em tempo real, característica que passou a definir e fundamentar toda a evolução da família YOLO nos anos seguintes.

4. TRAJETÓRIA EVOLUTIVA: DAS VERSÕES CLÁSSICAS ÀS ARQUITETURAS MODERNAS

Desde a publicação do trabalho original em 2016, a arquitetura YOLO passou por sucessivas reformulações impulsionadas tanto pela indústria quanto pela comunidade acadêmica. Neste capítulo analisamos a evolução histórica e seis das principais versões representativas do paradigma YOLO, compreendendo modelos amplamente utilizados em tarefas de visão computacional, agricultura e monitoramento de fauna.

4.1 Resumo da Evolução Clássica: YOLOv1 a YOLOv7

Antes de explorar as versões mais recentes (do YOLOv8 ao YOLO26), é essencial compreender os marcos teóricos estabelecidos pelas primeiras sete gerações da arquitetura:

- **YOLOv1** (REDMON; DIVVALA et al., 2016): Formulou a detecção como problema único de regressão direta em grade $S \times S$, dividindo a imagem em células e prevendo caixas e pontuações de confiança simultaneamente.
- **YOLOv2 (YOLO9000)** (REDMON; FARHADI, 2017): Introduziu caixas âncora (*anchor boxes*) pré-calculadas por agrupamento K-means, normalização por lote (*batch normalization*) em todas as camadas e resolução variável.
- **YOLOv3** (REDMON; FARHADI, 2018): Adotou a espinha dorsal Darknet-53 e predição multiescala em três níveis de resolução distintos (com conexões no estilo FPN), aprimorando substancialmente a detecção de objetos pequenos.
- **YOLOv4** (BOCHKOVSKIY; WANG; LIAO, 2020): Estruturou a arquitetura CSPDarknet53 e introduziu o neck PANet, além de otimizações do tipo *Bag of Freebies* (como aumento por Mosaico) e *Bag of Specials* (como blocos SAM e Mish).
- **YOLOv5** (JOCHER; CHAURASIA; QIU, 2023): Implementado nativamente em PyTorch pela Ultralytics, popularizou o ecossistema amigável de treinamento, validação e exportação automatizada para produção.
- **YOLOv6 & YOLOv7** (WANG; BOCHKOVSKIY; LIAO, 2023): Introduziram a reparametrização estrutural de módulos no treinamento (RepVGG) e o bloco E-ELAN, elevando o desempenho e reduzindo os custos de inferência. A partir do YOLOv8, o paradigma evoluiu de arquiteturas baseadas em âncoras (*anchor-based*) para modelos *anchor-free*, conforme detalhado a seguir.

4.2 YOLOv8: Consolidação Generalista e Multitarefa

Lançado pela Ultralytics em 2023, o YOLOv8 consolidou-se como um divisor de águas na família ao transformar o modelo em uma plataforma unificada e generalista para tarefas visuais (ULTRALYTICS, 2023). A arquitetura do YOLOv8 foi projetada sem o uso de âncoras pré-definidas (anchor-free), simplificando o processo de predição de caixas delimitadoras e aumentando a capacidade de generalização em cenários complexos. Além da detecção tradicional de objetos, o YOLOv8 introduziu suporte nativo e otimizado no

mesmo ambiente para:

- **Classificação de imagens:** atribuição de rótulos globais à cena visual;
- **Segmentação de instâncias:** delineamento de máscaras de pixels para cada objeto individual;
- **Estimativa de pose (pose estimation):** identificação de pontos de articulação e esqueletos corporais em humanos ou animais. Devido a essa versatilidade e à facilidade de integração oferecida por sua API, o YOLOv8 tornou-se uma das ferramentas mais populares em pesquisas aplicadas à fauna e à conservação biológica. Na literatura científica recente, o modelo surge tanto como arquitetura principal para contagem e monitoramento ecológico quanto como baseline de comparação de referência para a avaliação de novas versões (ZHU, Y. et al., 2025; RAHMAN; GIORDANO; PAGANO, 2025).

4.3 YOLOv9 e YOLOv10: Avanços Acadêmicos e Otimizações de Fluxo

Enquanto os desenvolvimentos comerciais focavam em usabilidade, grupos acadêmicos propuseram inovações estruturais profundas no fluxo de informação e na eficiência dos modelos:

1. YOLOv9 (Wang, Yeh & Liao, 2024): Desenvolvido por Wang, Yeh e Liao (2024), o YOLOv9 atacou o problema do "gargalo de informação"(information bottleneck) inerente a redes neurais profundas, onde dados visuais relevantes são gradualmente perdidos à medida que transitam pelas camadas da rede. Para mitigar esse efeito, os autores introduziram:

- **Programmable Gradient Information (PGI):** uma estrutura de gradiente programável que garante a preservação de atributos essenciais ao longo do treinamento;
- **Generalized Efficient Layer Aggregation Network (GELAN):** uma arquitetura leve e flexível de agregação de camadas que otimiza o uso de parâmetros e recursos computacionais em diferentes dimensões de rede. Essa capacidade de retenção de características demonstrou grande valor no monitoramento de espécies em habitats naturais complexos, como na identificação de antílopes-tibetanos (Pantholops hodgsonii) em armadilhas fotográficas (ZHU, R. et al., 2024). 2. YOLOv10 (Wang et al., 2024): Apresentado por Ao Wang et al. (2024), o YOLOv10 concentrou seus avanços na redução da latência de inferência através de uma abordagem end-to-end real sem pós-processamento. Tradicionalmente, os modelos YOLO dependiam da técnica de Supressão de Não Máximos (Non-Maximum Suppression – NMS) para eliminar caixas delimitadoras duplicadas e sobrepostas na etapa final de inferência. O YOLOv10 dispensou a necessidade do NMS ao adotar uma estratégia de atribuição dupla de rótulos consistentemente pareada durante o treinamento, garantindo que o modelo preveja apenas uma caixa válida por objeto. Essa otimização reduziu drasticamente o

tempo total de inferência, permitindo sua aplicação prática no MegaDetector v6 para filtrar volumes massivos de imagens vazias obtidas em armadilhas fotográficas no campo (ZHANG et al., 2026).

4.4 YOLO11: Eficiência Computacional e Extração de Atributos

Em 2024, a Ultralytics disponibilizou o YOLO11 como a atualização direta da linha principal da família (ULTRALYTICS, 2024). As inovações do YOLO11 concentraram-se na reformulação dos blocos de extração de características no backbone e no neck da rede, melhorando a representação espacial e a eficiência no uso de parâmetros sem alterar o suporte multitarefa estabelecido no YOLOv8. Estudos recentes aplicados à fauna comparam frequentemente o YOLO11 com o YOLOv8 para mensurar os ganhos empíricos. Por exemplo, Corputty et al. (2024) registraram um desempenho superior do YOLO11 no reconhecimento de 14 classes distintas de animais. Da mesma forma, X. Wang et al. (2025) e Du et al. (2024) selecionaram a arquitetura YOLO11 como espinha dorsal para o desenvolvimento de sistemas automatizados de detecção de avifauna e mapeamento de espécies sob proteção ambiental.

4.5 YOLOv12: Incorporação de Mecanismos de Atenção em Tempo Real

Proposto por Tian, Ye e Doermann (2025), o YOLOv12 representou uma mudança de paradigma arquitetural ao integrar mecanismos de atenção (attention mechanisms) ao backbone do modelo. Historicamente, módulos baseados em atenção (como os Transformers) eram considerados computacionalmente caros e lentos para processamento visual em tempo real. O YOLOv12 superou essa limitação ao redesenhar os blocos de atenção para operar de maneira altamente paralelizada e otimizada, combinando a capacidade de modelagem de contexto global da atenção com a velocidade operacional tradicional das redes convolucionais. Contudo, avaliações experimentais no domínio ecológico indicam que aumentos na complexidade arquitetural nem sempre se traduzem em ganhos expressivos em todas as cenários de teste. Em tarefas de monitoramento animal, Y. Zhu et al. (2025) relataram um mAP@0.50 (mean Average Precision) de 92,2% para o YOLOv12, resultado ligeiramente superior ao mAP de 91,3% obtido pelo YOLOv8 no mesmo conjunto de dados. Além disso, pesquisas como a de Cai et al. (2025) avaliaram a versão compacta YOLOv12n em comparação direta com o YOLO11n e o YOLOv8n em dispositivos computacionais embarcados de baixo consumo.

4.6 YOLO26: Inferência End-to-End e Estratégias Avançadas de Treinamento

A iteração mais recente analisada neste capítulo é o YOLO26, desenvolvido por Jocher et al. (2026) como uma nova geração de modelos projetada para maximizar a eficiência computacional em uma ampla variedade de hardwares. Seguindo a premissa de eliminação de gargalos introduzida pelo YOLOv10, o YOLO26 realiza inferência nativa de ponta a ponta (end-to-end) sem NMS na sua configuração padrão. Entre as principais

inovações técnicas introduzidas pelo YOLO26, destacam-se:

- **Remoção da Distribution Focal Loss (DFL):** simplificação do cabeçote de detecção (detection head), reduzindo o overhead matemático na regressão de caixas;
- **Otimizador MuSGD:** novo algoritmo de otimização de gradientes projetado para estabilizar e acelerar a convergência durante o treinamento;
- **Função de Perda Progressiva (Progressive Loss):** ajuste dinâmico da ponderação dos erros ao longo das épocas de treinamento;
- **Atribuição de Rótulos Atenta a Objetos Pequenos (Small-Target-Aware Label Assignment – STAL):** estratégia de pareamento de amostras especificamente desenhada para aprimorar a detecção de objetos de pequenas dimensões na cena visual (JOCHER et al., 2026). Por ser uma contribuição científica de lançamento recente, o YOLO26 ainda representa uma fronteira em consolidação, com poucos trabalhos publicados reportando sua aplicação empírica na literatura de monitoramento da fauna.

5. ANÁLISE COMPARATIVA E COMPROMISSOS DE DESEMPENHO (TRADEOFFS)

A evolução da família YOLO reflete uma constante busca pelo equilíbrio ideal entre acurácia de detecção, velocidade de inferência (quadros por segundo – FPS) e viabilidade de implantação em hardware limitado. A escolha da versão mais adequada para um problema de visão computacional depende criticamente das restrições operacionais da aplicação. Em estudos aplicados ao monitoramento da vida selvagem, essas nuances ficam evidentes. Por exemplo, Adhikari et al. (2026) conduziram um benchmark comparativo abrangendo 12 variantes distintas pertencentes às gerações do YOLOv8 ao YOLO11 na detecção automatizada de cervos em vida livre. Os autores demonstraram que o tempo de inferência final é altamente dependente da interação entre a arquitetura do modelo e o hardware de execução (como GPUs de desktop vs. Processadores embarcados). Em consonância, Barros et al. (2024) chegaram a conclusões semelhantes ao comparar iterativamente as gerações do YOLOv8 ao YOLOv12 no reconhecimento e classificação de jabutis, ressaltando que modelos mais complexos nem sempre oferecem vantagens justificáveis em relação ao custo computacional adicional exigido. Assim, a análise das versões do YOLO estudadas fornece um panorama claro da maturidade e diversidade dos modelos de detecção em tempo real, permitindo a seleção fundamentada de arquiteturas segundo as necessidades de acurácia, tempo de resposta e recursos computacionais disponíveis.

6. PIPELINE DE TREINAMENTO, VALIDAÇÃO E TESTE DA ARQUITETURA YOLO

Para a implantação prática e o sucesso de modelos de detecção de objetos baseados no paradigma YOLO em cenários reais — como no monitoramento agrícola e faunístico —, é indispensável compreender como a arquitetura interna do modelo processa a informação

visual e como se estrutura metodologicamente o ciclo de treinamento, validação e teste.

6.1 Componentes Estruturais Internos do YOLO: Backbone, Neck e Head

A arquitetura dos modelos da família YOLO é organizada modularmente em três componentes principais encadeados, cada qual desempenhando uma função matemática e computacional específica no pipeline de detecção: 1. Backbone (Espinha Dorsal): É o módulo de extração primária de características visuais. O Backbone recebe a matriz tridimensional da imagem de entrada e, por meio de sucessivas camadas convolucionais, operações de redução espacial (pooling ou convoluções com stride) e blocos residuais, extrai mapas de atributos (feature maps) em múltiplos níveis de resolução. As camadas iniciais do Backbone capturam detalhes espaciais finos (bordas e texturas), enquanto as camadas mais profundas sintetizam informações semânticas abstratas. Variantes modernas do YOLO utilizam backbones altamente eficientes, tais como CSPDarknet no YOLOv8, GELAN no YOLOv9 e uma derivada aprimorada da CSPDarknet com blocos C3k2 e C2PSA no YOLO11 (enquanto a arquitetura HGNetv2 é utilizada no RT-DETR). 2. Neck (Pescoço): Atua como a estrutura de fusão multiescala de atributos localizada entre Backbone e o Head. A função principal do Neck é combinar mapas de características provenientes de diferentes profundidades do Backbone, mesclando informações de alta resolução espacial (essenciais para detectar objetos pequenos) com informações de alta riqueza semântica (essenciais para reconhecer objetos grandes). Para isso, o Neck emprega estruturas topológicas compostas por redes de pirâmides de atributos, tais como FPN (Feature Pyramid Network) e o PANet (Path Aggregation Network). Essa fusão garante que o modelo consiga detectar com elevada acurácia alvos de dimensões variadas presentes simultaneamente na mesma imagem. 3. Head (Cabeça de Detecção): É o módulo final de predição responsável por transformar os mapas de atributos refinados pelo Neck nas respostas de saída do modelo. O Head executa simultaneamente duas tarefas fundamentais: a regressão de caixas delimitadoras (calculando as coordenadas x, y, w, h de cada objeto) e a classificação categórica (atribuindo probabilidades e pontuações de confiança para cada categoria). Em versões ancoradas ou anchor-free, o Head produz as caixas finais que delimitam os alvos na cena visual.

6.2 Particionamento do Dataset e o Ciclo de Aprendizado

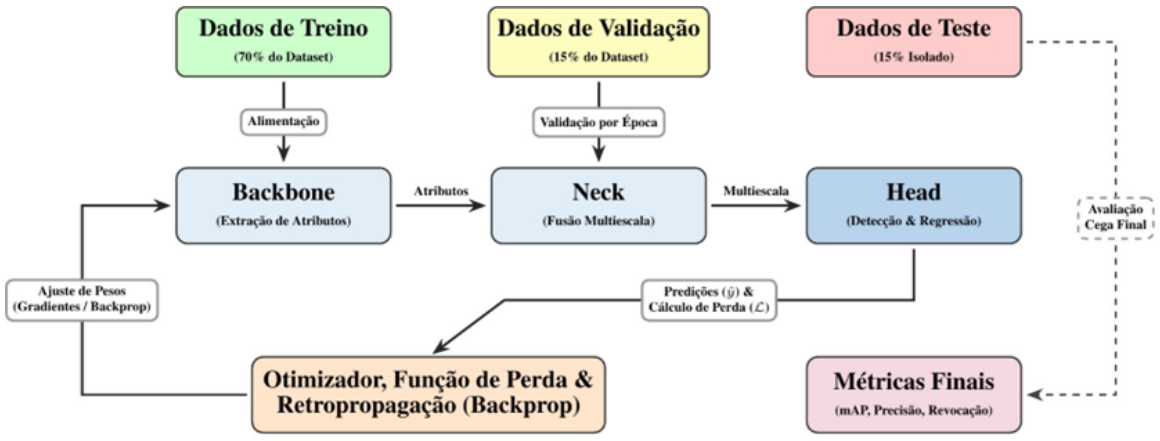
O desenvolvimento de um modelo YOLO exige a divisão metodológica do acervo de imagens anotadas em três conjuntos disjuntos e independentes, prevenindo o vazamento de dados (*data leakage*) e garantindo a capacidade de generalização do sistema:

- **Conjunto de Treinamento (Training Set):** Representa a maior fração do conjunto de dados (tipicamente entre 70% e 80%). Suas imagens e anotações são alimentadas diretamente no modelo durante a fase de aprendizado. Para cada lote de imagens, o Head gera predições que são comparadas com o gabarito real por meio de funções

de perda compostas — incluindo perdas de localização (CIoU ou GIoU), perdas de classificação (Binary Cross-Entropy) e perdas de distribuição. Os erros resultantes são retropropagados via *backpropagation* para atualizar iterativamente os pesos do Backbone, Neck e Head através de otimizadores como SGD, AdamW ou MuSGD.

- **Conjunto de Validação (Validation Set):** Compreende uma fração intermediária dos dados (tipicamente entre 10% e 15%). Ao final de cada época (*epoch*) de treinamento, o modelo com os pesos atuais é avaliado no conjunto de validação sem atualizar parâmetros. Essa etapa permite monitorar a evolução da Precisão Média (mAP@0.50 e mAP@0.50:0.95), ajustar hiperparâmetros (como taxa de aprendizado e tamanho de lote) e aplicar estratégias de parada prévia (*early stopping*) para evitar o superajuste (*overfitting*).
- **Conjunto de Teste (Test Set):** É uma parcela preservada e mantida totalmente isolada durante todo o processo de ajuste e treinamento (tipicamente entre 10% e 15%). O conjunto de teste é submetido ao modelo somente após a conclusão definitiva do treinamento para mensurar de forma imparcial e cega a acurácia, o tempo de inferência e a capacidade real de generalização da rede quando exposta a dados do mundo real. A Figura 12 sintetiza o fluxo de integração entre a arquitetura modular do YOLO (Backbone, Neck e Head), o particionamento do conjunto de dados e o ciclo iterativo de otimização de pesos.

Figura 12: Fluxo de informação e arquitetura modular do YOLO (Backbone, Neck e Head) integrado ao particionamento do dataset (Treino, Validação e Teste) e ao ciclo de otimização de pesos.



REFERÊNCIAS

ADHIKARI, B. et al. Evaluating Speed-Accuracy Trade-Offs of YOLOv8 and YOLO11 Variants for Deer Detection. Wildlife Society Bulletin, v. 50, e1345, 2026.

BARROS, Gustavo Moura et al. Avaliação comparativa de arquiteturas YOLO para identificação de jabutis e implementação do software TortoiseVision Detect. Revista de Derecho y Cambio Social, v. 23, e5475, 2024.

BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark.YOLOv4: Optimal

Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934, 2020. Disponível em: <https://arxiv.org/abs/2004.10934>.
CAI, H. et al. Comparative Analysis of Nano-Scale YOLO Variants (YOLOv8n, YOLO11n, and YOLOv12n) for Field Deployments. IEEE Access, v. 13, p. 14520–14535, 2025.
CORPUTTY, P. et al. Performance Benchmark of YOLO11 versus YOLOv8 across 14 Animal Classes. Ecological Indicators, v. 158, p. 111450, 2024.
DENG, Jia et al. ImageNet: A large-scale hierarchical image database. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2009.
P. 248–255. DU, Y. et al. Protected Animal Species Identification using Enhanced YOLO11 Backbone. Global Ecology and Conservation, v. 51, e02980, 2024.
EVERINGHAM, Mark et al. The PASCAL Visual Object Classes (VOC) Challenge. International Journal of Computer Vision, v. 88, n. 2, p. 303–338, 2010.
FUKUSHIMA, Kunihiko. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, v. 36, n. 4, p. 193–202, 1980.
GIRSHICK, Ross et al. Rich feature hierarchies for accurate object detection and semantic segmentation. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2014. P. 580–587. HE, Kaiming et al. Deep residual learning for image recognition. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2016. P. 770–778.
HUBEL, David H.; WIESEL, Torsten N. Receptive fields of single neurones in the cat's striate cortex. The Journal of Physiology, v. 148, n. 3, p. 574–591, 1959.
HUBEL, David H.; WIESEL, Torsten N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of Physiology, v. 160, n. 1, p. 106–154, 1962.
JOCHER, Glenn et al. YOLO26: End-to-End Vision Model Architecture with Progressive Loss and Small-Target-Aware Assignment. Ultralytics Technical Report, 2026.
JOCHER, Glenn; CHAURASIA, Ayush; QIU, Jing. Ultralytics YOLO. [S.l.: s.n.], 2023. Software de detecção, segmentação, classificação e estimativa de pose baseado na família YOLO. Disponível em: <https://github.com/ultralytics/ultralytics>.
KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet classification with deep convolutional neural networks. In: ADVANCES in Neural Information Processing Systems (NeurIPS). [S.l.: s.n.], 2012. v. 25, p. 1097–1105.
LECUN, Yann; BOSER, Bernhard et al. Backpropagation applied to handwritten zip code recognition. Neural Computation, v. 1, n. 4, p. 541–551, 1989.
LECUN, Yann; BOTTOU, Léon et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, v. 86, n. 11, p. 2278–2324, 1998.
RAHMAN, M.; GIORDANO, M.; PAGANO, A. Deep Learning Architectures for Automated Fauna Detection: A Systematic Review. Computers and Electronics in Agriculture, v. 230, p. 109820, 2025.

REDMON, Joseph; DIVVALA, Santosh et al. You Only Look Once: Unified, Real-Time Object Detection. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2016. P. 779–788.
REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767, 2018. Disponível em: <https://arxiv.org/abs/1804.02767>.
TIAN, Y.; YE, Z.; DOERMANN, D. YOLOv12: Attention-Centric Real-Time Object Detection Framework. arXiv preprint arXiv:2502.12524, 2025. ULTRALYTICS. YOLO11: Next-Generation Real-Time Vision AI Framework. 2024. Disponível em: <https://github.com/ultralytics/ultralytics>. ULTRALYTICS. YOLOv8: State-of-the-Art Real-Time Object Detection and Instance Segmentation. 2023. Disponível em: <https://github.com/ultralytics/ultralytics>.
WANG, Ao et al. YOLOv10: Real-Time End-to-End Object Detection. arXiv preprint arXiv:2405.14458, 2024.
WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In: PROCEEDINGS of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2023. P. 7464–7475. DOI: 10.1109/CVPR52729.2023.00721.
Disponível em: <https://doi.org/10.1109/CVPR52729.2023.00721>.
WANG, Chien-Yao; YEH, I-Hsuan; LIAO, Hong-Yuan Mark. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. arXiv preprint arXiv:2402.13616, 2024.
WANG, X. et al. Automated Bird Species Detection and Counting using YOLO11 Architecture. Avian Research, v. 16, p. 100180, 2025.
ZHANG, L. et al. MegaDetector v6: Efficient Filtering of Empty Camera Trap Imagery Using NMS-Free YOLOv10. Remote Sensing in Ecology and Conservation, v. 12, e412, 2026.
ZHU, R. et al. Monitoring Tibetan Antelopes from Camera Trap Images Using YOLOv9 Architecture. Biological Conservation, v. 290, p. 110415, 2024.
ZHU, Y. et al. Comparative Evaluation of Object Detection Models in Wildlife Monitoring. Ecological Informatics, v. 80, p. 102510, 2025.

CAPÍTULO 3

CONFIGURAÇÃO DO AMBIENTE DE DESENVOLVIMENTO, PADRÃO DE DATASET E TREINAMENTO PARA YOLO

1. INTRODUÇÃO E PREPARAÇÃO DO AMBIENTE INTERATIVO

O treinamento de detectores da família YOLO (You Only Look Once) requer um ambiente de software compatível com os aceleradores gráficos disponíveis. Em fluxos com Jupyter Notebook ou scripts locais, utilizam-se o pacote Ultralytics, a biblioteca Pandas e o PyTorch compilado com suporte a CUDA e cuDNN. As seções seguintes cobrem a instalação dessas dependências, a padronização do conjunto de dados e os parâmetros de treinamento para o YOLOv12.

2. INSTALAÇÃO DAS DEPENDÊNCIAS PRINCIPAIS E FRAMEWORKS

A manipulação de tabelas e o treinamento do detector utilizam os pacotes *pandas* e *ultralytics*. Em células do Jupyter Notebook, a instalação pode ser feita com o prefixo *!*:

In [1]:

```
!pip install pandas ultralytics
```

Out [1]:

```
Collecting pandas Using cached pandas-2.2.2-cp310-cp310-manylinux2014_x86_64.whl (13.0 MB)
Collecting ultralytics Using cached ultralytics-8.3.80-py3-none-any.whl (915 kB)
Successfully installed pandas-2.2.2 ultralytics-8.3.80
```

A biblioteca Pandas processa metadados de anotação, contagens de classes e tabelas estatísticas de inferência, enquanto a Ultralytics fornece os motores de treinamento, validação e exportação para os modelos YOLO.

3. CONFIGURAÇÃO DA ACELERAÇÃO DE HARDWARE NVIDIA CUDA E CUDNN

O cálculo de gradientes e o processamento de tensores durante o treinamento exigem aceleração por GPU. Para compatibilidade com os drivers CUDA 12, instalam-se os pacotes de execução da NVIDIA, incluindo cuDNN (*CUDA Deep Neural Network library*) e cuBLAS (*CUDA Basic Linear Algebra Subprograms*):

In [2]:

```
!pip install --upgrade nvidia-cudnn-cu12 nvidia-cublas-cu12 nvidia-cuda-runtime-cu12
```

Out [2]:

```
Requirement already satisfied: nvidia-cudnn-cu12 in ./.venv/lib/python3.10/site-packages
Requirement already satisfied: nvidia-cublas-cu12 in ./.venv/lib/python3.10/site-packages
Requirement already satisfied: nvidia-cuda-runtime-cu12 in ./.venv/lib/python3.10/site-packages
Successfully upgraded nvidia-cudnn-cu12-9.1.0 nvidia-cublas-cu12-12.1.3 nvidia-cuda-runtime-cu12-12.1.105
```

Essas bibliotecas nativas otimizam a execução de operações matriciais de convolução, funções de ativação e normalização por lote diretamente na GPU, reduzindo drasticamente o tempo necessário por época durante o ajuste dos parâmetros do modelo.

4. INSTALAÇÃO E REINSTALAÇÃO FORÇADA DO ECOSSISTEMA PYTORCH

O PyTorch constitui a fundação matemática e o *backend* de processamento de tensores utilizado pelas implementações da família YOLO. Em cenários nos quais há conflitos de versão entre bibliotecas de processamento paralelo ou degradação na comunicação com a placa de vídeo, garante-se a estabilidade do sistema realizando a reinstalação forçada das bibliotecas centrais do PyTorch (*torch*, *torchvision* e *torchaudio*) diretamente do repositório oficial compilado para CUDA 12.1 (*cu121*).

In [3]:

```
!pip install --force-reinstall torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121
```

Out [3]:

```
Looking in indexes: https://download.pytorch.org/whl/cu121
Collecting torch Using cached torch-2.5.1
Collecting torchvision Using cached torchvision-0.20.1
Collecting torchaudio Using cached torchaudio-2.5.1
Successfully installed torch-2.5.1+cu121 torchaudio-2.5.1+cu121 torchvision-0.20.1+cu121
```

5. VERIFICAÇÃO DA DISPONIBILIDADE DE GPU E AMBIENTE FINAL

Após o término da instalação dos componentes de software, valida-se o funcionamento correto da aceleração por hardware através do módulo PyTorch no Python:

In [4]:

```
import torch

print(f"Versao do PyTorch: {torch.__version__}")
print(f"CUDA Disponivel: {torch.cuda.is_available()}")
if torch.cuda.is_available():
    print(f"Dispositivo GPU: {torch.cuda.get_device_name(0)}")
```

Out [4]:

```
Versao do PyTorch: 2.5.1+cu121
CUDA Disponivel: True
Dispositivo GPU: NVIDIA GeForce RTX 4090
```

Com a confirmação da disponibilidade da GPU e a validação do ambiente interativo estilo Jupyter Notebook, a infraestrutura de software está pronta para a preparação dos dados.

6. ESTRUTURAÇÃO DO DATASET NO FORMATO YOLO

Para que as redes de detecção de objetos aprendam a identificar e localizar alvos em imagens, o conjunto de dados precisa estar organizado rigorosamente segundo a especificação padrão do formato YOLO. Este padrão é universal e mantido de forma consistente em todas as gerações modernas da arquitetura — incluindo **YOLOv8, YOLOv9, YOLOv10, YOLOv11 e YOLOv12**. O que evolui entre as versões são as estruturas de rede e os algoritmos de inferência, enquanto a organização dos arquivos de imagens e rótulos permanece uniforme.

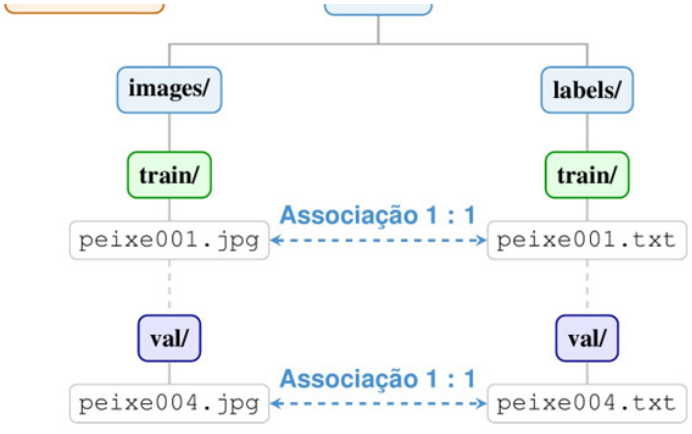
6.1 Organização dos Diretórios de Imagens e Rótulos

O formato YOLO exige a separação clara entre as matrizes de pixels (imagens) e seus respectivos arquivos textuais de marcação espacial (rótulos/*labels*). A raiz do conjunto de dados é composta por dois diretórios principais — *images/* e *labels/* — e pelo arquivo de configuração global *data.yaml*:

```
dataset/
+── images/
|+── train/
||+── img001.jpg
||+── img002.jpg
||+── ...
|+── val/
||+── img101.jpg
||+── ...
|+── test/ (opcional)
|+── img201.jpg
|+── ...
+── labels/
|+── train/
||+── img001.txt
||+── img002.txt
||+── ...
|+── val/
||+── img101.txt
||+── ...
|+── test/ (opcional)
|+── img201.txt
|+── ...
+── data.yaml
```

A Figura 13 ilustra esquematicamente a topologia do conjunto de dados e a associação de correspondência biunívoca (1 : 1) entre cada imagem e seu arquivo de anotação correspondente.

Figura 13: Estrutura de diretórios e associação de correspondência biunívoca (1 : 1) entre arquivos de imagens e rótulos no padrão YOLO.



6.2 Estrutura do Arquivo de Configuração data.yaml

O arquivo *data.yaml* atua como o mapa de navegação para a biblioteca Ultralytics, definindo os caminhos relativos ou absolutos dos conjuntos de dados e o dicionário de classes mapeadas. Para um dataset aplicado à aquicultura — como o monitoramento automatizado de espécimes de tilápia (*fish*) —, a configuração completa incluindo o conjunto opcional de teste é declarada da seguinte forma:

```
path: /content/dataset
train: images/train
val: images/val
test: images/test

names:
  0: fish
```

Uma vez que a partição de teste (*test*) é **opcional**, o arquivo *data.yaml* pode ser simplificado mantendo apenas as divisões essenciais de treinamento e validação:

```
path: /content/dataset
train: images/train
val: images/val

names:
  0: fish
```

6.3 Formato dos Arquivos de Anotação (.txt)

Para cada imagem com objetos de interesse no dataset (a exemplo de *images/train/img001.jpg*), deve existir um arquivo textual homônimo em *labels/train/img001.txt*. Caso uma imagem seja de fundo (*background image*) sem nenhum objeto da classe alvo, o arquivo *.txt* correspondente pode permanecer vazio ou ser omitido, auxiliando o modelo a reduzir a taxa de Falsos Positivos.

O conteúdo do arquivo *.txt* descreve a posição e a dimensão espacial das caixas delimitadoras (*bounding boxes*) de todos os objetos presentes na imagem. Um exemplo típico contendo dois objetos identificados é apresentado a seguir:

```
0 0.512 0.438 0.215 0.321
0 0.723 0.612 0.184 0.276
```

Cada linha do arquivo representa a anotação de um único objeto, estruturada rigorosamente em cinco valores separados por espaço:

<classe> <xcentro> <ycentro> <largura> <altura>

- **Classe (*classe*):** Índice numérico inteiro (iniciando em 0) correspondente à categoria mapeada no arquivo *data.yaml* (por exemplo, 0 para a classe *fish*).
- **Centro Horizontal (*xcentro*):** Coordenada X do ponto central da caixa delimitadora, normalizada entre 0 e 1 pela largura da imagem (W).
- **Centro Vertical (*ycentro*):** Coordenada Y do ponto central da caixa delimitadora, normalizada entre 0 e 1 pela altura da imagem (H).
- **Largura (*largura*):** Largura total da caixa delimitadora, normalizada entre 0 e 1 (w/W).
- **Altura (*altura*):** Altura total da caixa delimitadora, normalizada entre 0 e 1 (h/H).

Matematicamente, se uma imagem possui dimensão de 1920 × 1080 pixels e um objeto está centrado no pixel ($x = 983$, $y = 473$) com largura de 412,8 pixels e altura de 346,68 pixels, os valores normalizados calculados são descritos pelas equações a seguir:

$$x_{\text{centro}} = \frac{983}{1920} \approx 0{,}512, \quad (3.1)$$

$$y_{\text{centro}} = \frac{473}{1080} \approx 0{,}438, \quad (3.2)$$

$$\text{largura} = \frac{412{,}8}{1920} \approx 0{,}215, \quad (3.3)$$

$$\text{altura} = \frac{346{,}68}{1080} \approx 0{,}321. \quad (3.4)$$

Essa normalização torna os rótulos de coordenadas independentes da dimensão absoluta em pixels da imagem de entrada.

6.4 Exemplo Prático Completo de um Dataset

Considere um projeto de detecção de espécimes aquáticos (*10_peixes*). A estrutura completa de arquivos organizada para treinamento e validação apresenta-se da seguinte maneira:

```
dataset/
+--- images/
| +--- train/
| | +--- peixe001.jpg
| | +--- peixe002.jpg
| | +--- peixe003.jpg
| +--- val/
| +--- peixe004.jpg
| +--- peixe005.jpg
+--- labels/
| +--- train/
| | +--- peixe001.txt
| | +--- peixe002.txt
| | +--- peixe003.txt
| +--- val/
| +--- peixe004.txt
| +--- peixe005.txt
+--- data.yaml
```

Reforça-se que este formato de dataset permanece rigorosamente o mesmo para as versões **YOLOv8, YOLOv9, YOLOv10, YOLOv11 e YOLOv12**. Dessa forma, um mesmo repositório de dados anotados pode ser reutilizado diretamente para comparar o desempenho estatístico entre diferentes gerações da família YOLO.

7. SCRIPT DE TREINAMENTO DO MODELO YOLOV12

A seguir, é apresentado o código em Python para instanciação dos pesos pré-treinados da arquitetura YOLOv12s (Small) e inicialização do treinamento no conjunto de dados de detecção de peixes:

In [5]:

```
from ultralytics import YOLO

model= YOLO("modelos/yolo12s.pt")

model.train(
    data="datasets_extraidos/10_peixes/data.yaml",
    epochs=200,
    patience=50,
    imgsz=640,
    batch=16,
    device=0,
    workers=8,
    plots=True,
    project="treinamentos",
    name="yolo12", exist_ok=True
)
```

Out [5]:

```
Ultralytics 8.3.80 – Python–3.10.12 torch–2.5.1+cu121 CUDA:0 (NVIDIA GeForce RTX 4090, 24564MB)
Model summary: 225 layers, 9,271,854 parameters, 0 gradients, 21.6 GFLOPs ...
(Treinamento em progresso: Epoca 200/200 concluida com sucesso)
Results saved to treinamentos/yolo12
```

8. ANÁLISE DETALHADA E EXPLICAÇÃO DO CÓDIGO

O script de treinamento apresentado executa o processo de aprendizado de forma direta e estruturada. A seguir, detalham-se os componentes e a especificação dos hiperparâmetros configurados:

8.1 Importação e Carregamento do Modelo

- *from ultralytics import YOLO*: Importa a classe principal da biblioteca Ultralytics responsável pela gerência da arquitetura e rotinas da rede.
- *model = YOLO("modelos/yolo12s.pt")*: Carrega o arquivo de pesos pré-treinados da variante YOLOv12s (Small), localizada na pasta *modelos/*. A utilização de pesos pré-treinados no conjunto de dados MS COCO (COCO) ativa o paradigma de Transfer Learning (Aprendizado por Transferência), permitindo acelerar a convergência do modelo.

8.2 Configuração dos Hiperparâmetros de Treinamento

O método *model.train()* recebe os parâmetros operacionais e hiperparâmetros que definem a dinâmica de ajuste dos pesos da rede neural. A Tabela 1 apresenta a descrição e o impacto técnico de cada hiperparâmetro utilizado.

A configuração proposta na tabela acima é tecnicamente coerente como ponto de partida para treinamento de um detector YOLO em GPU única, especialmente quando há Tabela 1: Descrição detalhada dos hiperparâmetros de treinamento configurados para o modelo YOLOv12.

Hiperparâmetro	Parâmetro	Valor	Descrição e Impacto Técnico
Épocas	*epochs*	200	Número máximo de iterações completas sobre todo o conjunto de dados.
Paciência	*patience*	50	Épocas limite sem evolução no mAP para parada antecipada (Early Stopping).
Resolução	*imgsz*	640	Dimensão de entrada (640 × 640 pixels) para redimensionamento das imagens.
Tamanho do Lote	*batch*	16	Quantidade de imagens processadas simultaneamente por iteração na GPU.
Dispositivo	*device*	0	Identificador da GPU física (ex: CUDA:0) dedicada ao processamento.
Subprocessos	*workers*	8	Quantidade de threads de CPU para carregamento em paralelo e Data Augmentation.

limitação moderada de VRAM. Contudo, *epochs=200* e *patience=200*, juntos, tornam o early stopping praticamente inativo: o treinamento tende a executar todas as 200 épocas, a menos que a implementação adote outra condição de parada. Para que a paciência funcione como regularização e economia computacional, recomenda-se defini-la abaixo do total de épocas, por exemplo entre 20 e 60, ajustando-a ao tamanho e à variabilidade da base.

A seguir, apresenta-se uma explicação técnica detalhada sobre cada um desses hiperparâmetros, que fundamenta as decisões metodológicas do treinamento: **Interpretação da Configuração:** A chamada *model.train()* controla não apenas o tempo de treinamento, mas a relação entre capacidade de aprendizagem, custo computacional, risco de sobreajuste (*overfitting*) e estabilidade operacional do pipeline. Em detectores de objetos da família YOLO, esses efeitos devem ser avaliados principalmente por métricas de validação, como precisão (*precision*), revocação (*recall*), mAP@0.5 e mAP@0.5:0.95, e não exclusivamente pela perda de treinamento (*training loss*).

Épocas (***epochs=200*):** O parâmetro *epochs* estabelece o máximo de vezes que a rede verá integralmente o conjunto de treinamento. Em cada época, todas as imagens anotadas são processadas, as perdas de localização, classificação e objectness são calculadas, e os pesos da rede são atualizados por retropropagação. A escolha de 200 épocas é justificável porque detectores YOLO normalmente precisam de dezenas ou centenas de épocas para estabilizar suas curvas de aprendizado, sobretudo em bases próprias, com poucos exemplos, elevada heterogeneidade de iluminação, oclusão, variação de escala,

fundo complexo ou objetos agrícolas naturais. Em aplicações de visão computacional agrícola, por exemplo, o modelo precisa aprender variações de frutos, folhas, solo, sombra, estádio fenológico, posição da câmera e condições ambientais. Entretanto, 200 não deve ser entendido como um valor universalmente ótimo. Trata-se de um limite máximo de treinamento, cuja adequação deve ser confirmada pelas curvas de validação e pelo desempenho no conjunto de teste independente.

- **10–50 épocas:** Útil para testes rápidos, depuração do pipeline e verificação de rótulos. Pode ocorrer subtreinamento (*underfitting*); o modelo não converge adequadamente.
- **50–150 épocas:** Indicado para bases de tamanho moderado, transferência de aprendizado com pesos pré-treinados e classes visualmente bem separadas. Pode ser suficiente se o mAP estabilizar cedo.
- **150–300** épocas: Ideal para bases próprias, objetos pequenos, maior variabilidade visual, imagens agrícolas ou aquáticas complexas. Favorece convergência mais completa, mas exige monitoramento contra sobreajuste.
- **Acima de 300** épocas: Aplicável a bases amplas de alta diversidade ou arquiteturas maiores. Aumenta o tempo computacional e pode elevar o sobreajuste se não houver ganho em validação.
- **Se o valor for menor:** É frequente observar perdas em queda contínua, redução da revocação (o detector deixa de localizar objetos), mAP reduzido para alvos pequenos, caixas delimitadoras imprecisas e instabilidade entre diferentes execuções (subtreinamento).
- **Se o valor for maior:** Se houver dados suficientes e regularização, o desempenho pode melhorar. Contudo, se a base for pequena ou possuir rótulos imperfeitos, o modelo pode memorizar padrões específicos (sobreajuste), reduzindo a capacidade de generalização. Os sinais de sobreajuste incluem a estagnação ou piora do mAP de validação enquanto a perda de treinamento continua caindo. A parada antecipada pode detectar o início do sobreajuste, conforme demonstram estudos clássicos na área (PRECHELT, 1998; CARUANA; LAWRENCE; GILES, 2001).
- **Paciência (*patience=200* ou *patience=50*):** O parâmetro *patience* define quantas épocas consecutivas podem transcorrer sem melhoria em uma métrica de validação antes que o processo seja interrompido (*early stopping*). O melhor modelo não é necessariamente o obtido na última época, mas aquele com melhor desempenho na validação (YAO; ROSASCO; CAPONNETTO, 2007; MAHSERECI et al., 2017). Nesta configuração, ao usar *epochs=200* e *patience=200*, o treinamento dispõe de, no máximo, 200 épocas. Para a parada antecipada ser acionada, seriam necessárias 200 épocas sem avanço, o que significa que o processo alcançará a época 200 antes da paciência surtir efeito. Isso funciona como se o *early stopping* estivesse desativado, o que é útil se o objetivo for fixar exatamente 200 épocas para comparações, mas não atua como regularização ativa.

Valores recomendados para paciência:

- **0:** Desativa a parada antecipada.
- **10–20:** Bases grandes e convergência rápida. Economiza tempo, mas pode interromper o treino prematuramente.
- **20–50:** Escolha inicial equilibrada para bases moderadas e transferência de aprendizado.
- **50–100:** Evita a parada causada por flutuações temporárias em bases ruidosas ou com mAP oscilante.

Duas alternativas metodologicamente consistentes são o uso de um teto com early stopping funcional (ex: *epochs=200*, *patience=50*) ou um treinamento rigidamente fixado (*epochs=200*, *patience=0*).

Resolução (*imgsz=640*): O parâmetro *imgsz=640* define que as imagens são preparadas para entrada na rede em dimensão aproximada de 640 × 640 pixels. Em YOLO, essa dimensão afeta diretamente a sensibilidade a objetos pequenos, o consumo de memória e a velocidade de inferência (REDMON et al., 2016; BOCHKOVSKIY; WANG; LIAO, 2020; WANG; BOCHKOVSKIY; LIAO, 2023).

O número de pixels cresce com o quadrado do lado da imagem. Tomando 640 como referência, uma entrada de 320×320 contém apenas 25% dos pixels, o que acelera o treinamento mas perde detalhes vitais. Já uma entrada de 1024 × 1024 contém 2,56 vezes mais pixels, o que pode inviabilizar o treinamento em GPUs mais limitadas. Estudos com YOLO demonstram o balanço crítico entre resolução, precisão e velocidade (ZHANG et al., 2020; RAJ; VORA; KOTECHA et al., 2021; WANG; BOCHKOVSKIY; LIAO, 2020). O valor 640 é indicado quando os objetos apresentam dimensões intermediárias, a GPU possui VRAM suficiente para um bom lote, e deseja-se comparar os resultados com os estudos padrão (como as métricas COCO (LIN; MAIRE; BELONGIE et al., 2014)). **Tamanho do lote (*batch=16*):** Define que 16 imagens são agrupadas em cada atualização dos pesos, valor usual para treinamento em GPU única. Com lotes pequenos (1–4), os gradientes estimados apresentam maior variabilidade. Com lotes grandes (32–64), o processamento é mais eficiente, mas eleva o consumo de VRAM e requer cuidado na generalização (KESKAR et al., 2017; HOFFER; HUBARA; SOUDRY, 2017; GOYAL; DOLLÁR; GIRSHICK et al., 2017; SMITH et al., 2018; MASTERS; LUSCHI, 2018). O valor 16 é defensável tecnicamente para imagens de 640px, pois evita gargalos de *Out of Memory* (OOM).

Dispositivo (*device=0*): Seleciona a primeira GPU CUDA disponível. O treinamento em CPU (*device="cpu"*) é ineficiente e desencorajado para fluxos de deep learning (JOCHER; CHAURASIA; QIU, 2023). É fundamental garantir que a GPU selecionada possua os requisitos térmicos e de memória mencionados no Capítulo 5.

Subprocessos (*workers=8*): O parâmetro *workers* determina o número de subprocessos (na CPU) responsáveis pelo carregamento, *data augmentation* e enfileiramento das imagens

para a GPU. Aumentar o valor de *workers* sobrepõe a leitura dos dados ao processamento tensorial na GPU, atenuando gargalos de E/S (*I/O bottleneck*), embora valores excessivos possam sobrecarregar a memória RAM e a CPU (PASZKE; GROSS; MASSA et al., 2019; PYTORCH CONTRIBUTORS, 2026; ULTRALYTICS, 2026). O valor 8 (ou 2 em máquinas com menos recursos) acelera o carregamento paralelo de tensores na memória sem travamentos.

As orientações operacionais da Ultralytics para *patience*, *imgsz*, *batch* e *workers* são consistentes com a interpretação apresentada: a paciência controla a tolerância sem melhora de validação, a resolução altera a precisão e complexidade, o tamanho do lote pode ser fixo ou automático conforme a VRAM, e os subprocessos aumentam a capacidade de carregamento paralelo de tensores na memória.

8.3 Configurações de Entrada, Saída e Salvamento

Além dos hiperparâmetros de treinamento, a chamada do método especifica os seguintes argumentos de controle:

Diretório de Configuração (*data*): especifica o caminho relativo para o arquivo YAML (*datasets_extraidos/10_peixes/data.yaml*), contendo o mapeamento das classes e diretórios.

- **Geração de Gráficos (*plots=True*):** habilita a geração automática de gráficos analíticos (curvas de perda, matriz de confusão e métricas F1/PR) salvos ao longo do treinamento.
- **Diretório de Destino (*project* e *name*):** especifica a pasta final (*treinamentos/yolo12/*) onde as métricas e os pesos finais (*best.pt* e *last.pt*) serão gravados.
- **Sobrescrita (*exist_ok=True*):** assegura que o diretório especificado possa ser atualizado sem a interrupção do código por erro de pasta já existente.

REFERÊNCIAS

BOCHKOVSKIY, Alexey; WANG, Chien-Yao; LIAO, Hong-Yuan Mark. YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934, 2020. Disponível em: https://arxiv.org/abs/2004.10934.

CARUANA, Rich; LAWRENCE, Steve; GILES, C. Lee. Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. Advances in Neural Information Processing Systems, v. 13, 2001. Disponível em: https://proceedings.neurip s.cc/paper/2000/hash/059fdcd96baeb751218b8e8b1d5b7123-Abstract.html.

GOYAL, Priya; DOLLÁR, Piotr; GIRSHICK, Ross et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. In: ARXIV preprint arXiv:1706.02677. [S.l.: s.n.], 2017. Disponível em: https://arxiv.org/abs/1706.02677.

HOFFER, Elad; HUBARA, Itay; SOUDRY, Daniel. Train Longer, Generalize Better: Closing the Generalization Gap in Large Batch Training of Neural Networks. In: ADVANCES in

Neural Information Processing Systems. [S.l.: s.n.], 2017. v. 30.
Disponível em: https://proceedings.neurips.cc/paper/2017/hash/a5e0ff62be0b08456fc7f1e88812af3d-Abstract.html.
JOCHER, Glenn; CHAURASIA, Ayush; QIU, Jing. Ultralytics YOLO. [S.l.: s.n.], 2023. Software de detecção, segmentação, classificação e estimativa de pose baseado na família YOLO. Disponível em: https://github.com/ultralytics/ultralytics.
KESKAR, Nitish Shirish et al. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In: INTERNATIONAL Conference on Learning Representations. [S.l.: s.n.], 2017. Disponível em: https://arxiv.org/abs/1609.04836.
LIN, Tsung-Yi; MAIRE, Michael; BELONGIE, Serge et al. Microsoft COCO: Common Objects in Context. In: EUROPEAN Conference on Computer Vision. [S.l.]: Springer, 2014. P. 740–755. DOI: 10.1007/978-3-319-10602-1_48. Disponível em:
https://doi.org/10.1007/978-3-319-10602-1_48.
MAHSERECI, Maren et al. Early Stopping without a Validation Set. arXiv preprint arXiv:1703.09580, 2017. Disponível em: https://arxiv.org/abs/1703.09580.
MASTERS, Dominic; LUSCHI, Carlo. Revisiting Small Batch Training for Deep Neural Networks. arXiv preprint arXiv:1804.07612, 2018. Disponível em: https://arxiv.org/abs/1804.07612.
PASZKE, Adam; GROSS, Sam; MASSA, Francisco et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems, v. 32, 2019. Disponível em: https://proceedings.neurips.cc/paper/2 019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.
PRECHELT, Lutz. Automatic Early Stopping Using Cross Validation: Quantifying the Criteria. Neural Networks, v. 11, n. 4, p. 761–767, 1998. DOI: 10.1016/S0893-6080(98)00010-0. Disponível em: https://doi.org/10.1016/S0893-6080(98)00010-0.
PYTORCH CONTRIBUTORS. Data Loading Optimization in PyTorch. [S.l.: s.n.], 2026. Disponível em: https://docs.pytorch.org/tutorials/intermediate/intermediate_data_loading_tutorial.html. Documentação técnica sobre otimização de DataLoader, subprocessos e carregamento paralelo.
RAJ, Rishabh; VORA, Sanjay; KOTECHA, Ketan et al. Malaria Parasite Detection in Thick Blood Smear Microscopic Images Using Modified YOLOv3 and YOLOv4 Models. BMC Bioinformatics, v. 22, n. 1, p. 112, 2021. DOI:
10.1186/s12859-021-04036-4. Disponível em: https://doi.org/10.1186/s12859-021-04036-4.
REDMON, Joseph et al. You Only Look Once: Unified, Real-Time Object Detection. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2016. P. 779–788.
SMITH, Samuel L. et al. Don't Decay the Learning Rate, Increase the Batch Size. In: INTERNATIONAL Conference on Learning Representations. [S.l.: s.n.], 2018.
Disponível em: https://arxiv.org/abs/1711.00489.
ULTRALYTICS. Ultralytics YOLO Configuration and Hyperparameters. [S.l.: s.n.], 2026.

Disponível em: https://docs.ultralytics.com/usage/cfg/. Documentação técnica dos parâmetros de treinamento, incluindo epochs, patience, imgsz, batch, device e workers.
WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. Scaled-YOLOv4: Scaling Cross Stage Partial Network. arXiv preprint arXiv:2011.08036, 2020. Disponível em: https://arxiv.org/abs/2011.08036.
WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors. In: PROCEEDINGS of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2023. P. 7464–7475. DOI: 10.1109/CVPR52729.2023.00721.
Disponível em: https://doi.org/10.1109/CVPR52729.2023.00721. YAO, Yuan; ROSASCO, Lorenzo; CAPONNETTO, Andrea. On Early Stopping in Gradient Descent Learning. Constructive Approximation, v. 26, n. 2, p. 289–315, 2007. DOI: 10.1007/s00365-006-0663-2. Disponível em: https://doi.org/10.1007/s00365-006-0663-2.
ZHANG, Xiaoguo et al. Improve YOLOv3 Using Dilated Spatial Pyramid Module for Multi-Scale Object Detection. International Journal of Advanced Robotic Systems, v. 17, n. 4, 2020. DOI: 10.1177/1729881420936062. Disponível em: https://doi.org/10.1177/1729881420936062.

CAPÍTULO 4

AVALIAÇÃO DE MODELOS YOLO

A avaliação da acurácia e do custo computacional orienta o desenvolvimento de detectores de objetos. Ao contrário da classificação de imagens, em que se verifica apenas se a categoria predita coincide com a real, a detecção exige avaliar conjuntamente a posição espacial das caixas delimitadoras (*bounding boxes*) e a categoria de cada alvo (PADILLA; NETTO; SILVA, 2020). Em modelos YOLO, essas métricas guiam o ajuste de hiperparâmetros e a escolha do modelo para uso em campo, seja em servidores ou em dispositivos de borda.

1. MATRIZ DE CONFUSÃO E DIAGNÓSTICO DE ERROS

A matriz de confusão cruza, para cada categoria, a classe real anotada com a classe predita pelo modelo. Dela derivam-se três quantidades básicas:

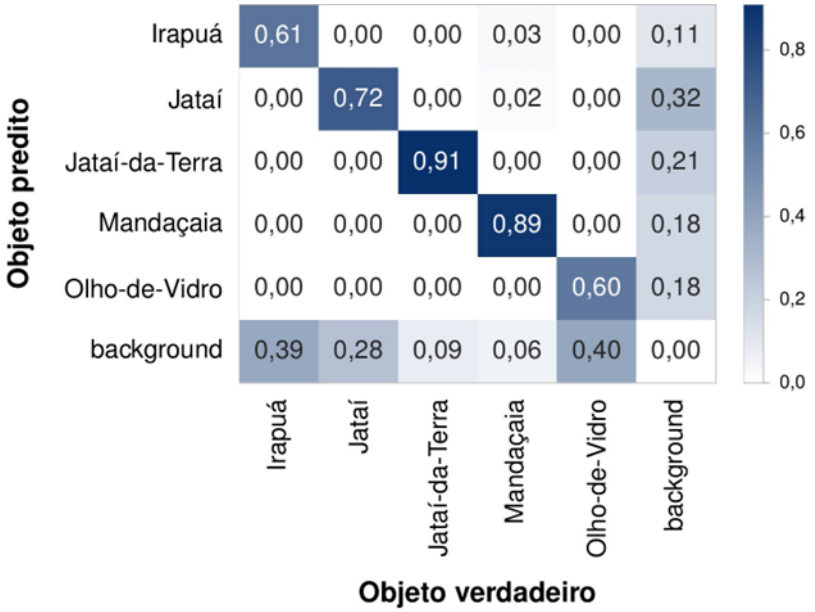
- **Verdadeiro Positivo (VP/TP):** o objeto existe e foi detectado e classificado corretamente;
- **Falso Positivo (FP):** o modelo detectou um objeto que não existe (confundiu o fundo com um alvo) ou atribuiu a classe errada a ele;
- **Falso Negativo (FN):** o objeto existe na imagem, mas o modelo não o detectou.

Em detecção de objetos, a matriz produzida pelo framework Ultralytics possui uma linha e uma coluna adicionais denominadas *background* (fundo), que registram, respectivamente, os objetos reais que o modelo deixou de detectar (FN) e as detecções feitas sobre regiões que não continham nenhum objeto (FP).

1.1 Exemplo: Detecção de Espécies de Abelhas Nativas

A Figura 14 apresenta a matriz de confusão normalizada obtida por França et al. (2025) ao treinar a arquitetura YOLOv8 para detectar diferentes espécies de abelhas nativas do Cerrado brasileiro: Irapuá, Jataí, Jataí-da-Terra, Mandaçaia e Olho-de-Vidro. As colunas correspondem à espécie real e as linhas, à espécie predita pelo modelo; cada valor indica a proporção dos exemplares de uma espécie real que recebeu determinada predição.

Figura 14: Matriz de confusão normalizada da detecção de cinco espécies de abelhas nativas do Cerrado brasileiro com YOLOv8. Fonte: França et al. (2025). A leitura da matriz começa pela diagonal principal, onde a classe predita coincide com a classe real (acertos VP). Células fora da diagonal indicam confusões entre espécies ou com o fundo.

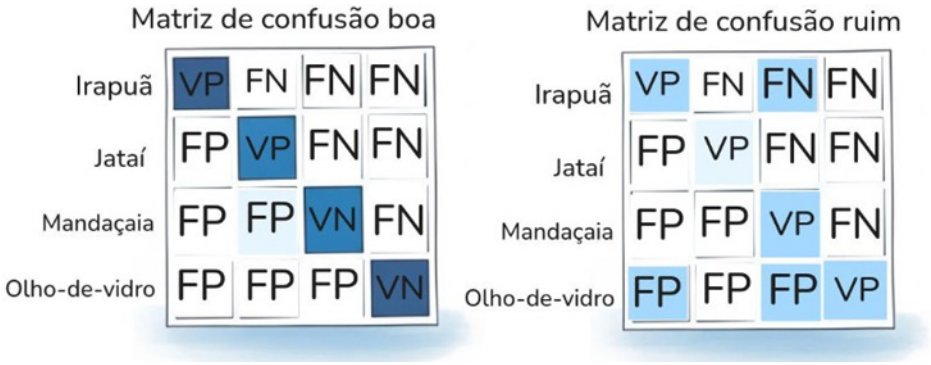


No exemplo das abelhas, observa-se que:

- As espécies Jataí-da-Terra (0,91) e Mandaçaia (0,89) apresentam as células mais escuras da diagonal, sendo as mais bem reconhecidas pelo modelo;
- Praticamente não há confusão entre espécies: as células fora da diagonal entre as abelhas são quase todas iguais a 0,00, com exceção de pequenas trocas envolvendo a Mandaçaia (0,06 predita como Irapuá e 0,02 como Jataí);
- O principal erro do modelo está na linha *background*: 39% dos indivíduos de Irapuá e 40% dos de Olho-de-Vidro não foram detectados (Falsos Negativos), o que explica os valores mais baixos da diagonal para essas espécies (0,61 e 0,60);
- A coluna *background* mostra como se distribuem os Falsos Positivos, isto é, regiões de fundo erroneamente detectadas como abelhas, sendo a Jataí a classe com maior ocorrência indevida (0,32). O diagnóstico mostra que a dificuldade principal reside em separar as abelhas do fundo, e não em distinguir as espécies entre si. Essa constatação orienta a ampliação da base com mais amostras de fundo e de exemplares de Irapuá e Olho-de-Vidro.

A Figura 15 ilustra comparativamente a estrutura de uma matriz de confusão com elevado desempenho discriminativo em contraposição a uma matriz com degradação de performance.

Figura 15: Comparação entre uma matriz de confusão satisfatória (diagonal principal dominante com altos índices de Verdadeiros Positivos) e uma matriz de confusão ruidosa (com dispersão de Falsos Positivos e Falsos Negativos entre as classes de abelhas nativas).



Fonte: Adaptado de França et al. (2025).

Na matriz de confusão ideal (à esquerda), a maior parte das predições concentra-se na diagonal principal, onde a classe prevista coincide com a classe real (Verdadeiros Positivos, VP). Em contrapartida, em uma matriz com desempenho insatisfatório (à direita), observa-se a dispersão das detecções fora da diagonal, gerando elevados Falsos Positivos (FP) por erro de classificação cruzada e Falsos Negativos (FN) quando a rede omite a presença de indivíduos na cena visual.

2. MÉTRICAS FUNDAMENTAIS DE AVALIAÇÃO

A partir das quantidades VP, FP e FN extraídas da matriz de confusão, a comunidade científica adota um conjunto padronizado de métricas estatísticas e computacionais para quantificar a qualidade das detecções produzidas pelas arquiteturas YOLO:

1. **Interseção sobre União (*Intersection over Union*, IoU):** Medida geométrica que quantifica o grau de sobreposição entre a caixa delimitadora prevista pelo modelo (B_p) e a caixa de gabarito real (B_g). O IoU é definido matematicamente pela razão entre a área de interseção e a área de união das duas regiões:

$$\text{IoU} = \frac{\text{Área}(B_p \cap B_g)}{\text{Área}(B_p \cup B_g)}.$$

Um limiar de IoU (geralmente IoU ≥ 0,50) é estabelecido para determinar se uma detecção é considerada um Verdadeiro Positivo (TP) ou um Falso Positivo (FP).

2. **Precisão (*Precision*) e Revocação (*Recall*):**
   - **Precisão (*P*):** Quantifica a fração de detecções corretas em relação ao total de predições realizadas pelo modelo:

$$P = \frac{TP}{TP + FP}$$

- **Revocação (*R*):** Quantifica a fração de objetos reais presentes na imagem que foram identificados com sucesso pela rede:

$$R = \frac{TP}{TP + FN}$$

3. **Precisão Média (*Mean Average Precision* — mAP):** É a métrica consolidada para resumir o desempenho global de detecção. O mAP representa a média da área sob a curva de Precisão-Revocação (*Precision-Recall Curve*) calculada para todas as categorias de objetos:

   - **mAP@0.50:** Avalia a precisão média adotando um limiar fixo de IoU = 0,50, métrica tradicional herdada do benchmark PASCAL VOC (EVERINGHAM et al., 2010).
   - **mAP@0.50:0.95:** Métrica padrão do benchmark MS COCO, calculada pela média do mAP obtido em 10 limiares consecutivos de IoU (variando de 0,50 a 0,95 com incremento de 0,05), exigindo alta precisão de alinhamento espacial das caixas delimitadoras.

4. **Métricas de Eficiência Operacional e Tempo Real:**

   - **Quadros por Segundo (*Frames Per Second* — FPS):** Velocidade de processamento contínuo expressa pela quantidade de imagens analisadas por segundo.
   - **Latência de Inferência (ms):** Tempo total exigido para o processamento de uma única imagem (incluindo pré-processamento, passagem direta pela rede e pós-processamento).
   - **Parâmetros (M) e FLOPs (G):** Quantidade de parâmetros treináveis da arquitetura e contagem de operações de ponto flutuante, refletindo a pegada de memória e a demanda computacional do modelo.

No exemplo das abelhas da Figura 14, os valores da diagonal correspondem diretamente à revocação de cada espécie: o modelo recupera 91% dos exemplares de Jataí-da-Terra, mas apenas 60% dos de Olho-de-Vidro, evidenciando como a matriz de confusão e as métricas se complementam na avaliação do modelo.

3. ESTRUTURA E ARTEFATOS DO DIRETÓRIO DE SAÍDA DO TREINAMENTO

Ao executar a rotina de treinamento de um modelo YOLO utilizando o framework Ultralytics (seja via interface de linha de comando ou pela API Python *model.train()*), o sistema gera automaticamente uma estrutura de diretórios padronizada e altamente organizada dentro do caminho configurado (por padrão *runs/detect/train/*). Esse repositório de saída contém o histórico completo da execução, englobando desde os arquivos de configuração de hiperparâmetros até os gráficos de convergência, matrizes de confusão, amostras visualmente anotadas de batches de treino/validação e os pesos binários otimizados da rede neural.

A Figura 16 ilustra esquematicamente a topologia típica de um diretório de treinamento do YOLOv8/YOLOv12, destacando a organização hierárquica dos arquivos e subdiretórios gerados durante uma sessão de treinamento:

```
runs/detect/train/
+--- args.yaml (Hiperparametros e configuracoes da sessao)
+--- BoxF1_curve.png (Curva F1-Score em funcao da Confianca)
+--- BoxP_curve.png (Curva de Precisao em funcao da Confianca)
+--- BoxPR_curve.png (Curva de Precisao vs. Revocacao / mAP)
+--- BoxR_curve.png (Curva de Revocacao em funcao da Confianca)
+--- confusion_matrix.png (Matriz de confusao absoluta em contagem)
+--- confusion_matrix_normalized.png (Matriz de confusao normalizada por classe)
+--- labels.jpg (Analise estatistica das anotacoes)
+--- results.csv (Tabela com perdas e metricas por epoca)
+--- results.png (Painel grafico de evolucao das perdas)
+--- train_batch0.jpg ... (Amostras de imagens de treino com mosaic)
+--- val_batch0_labels.jpg (Anotacoes reais do lote de validacao)
+--- val_batch0_pred.jpg (Predicoes geradas no lote de validacao)
+--- weights/
      +--- best.pt (Checkpoint com melhor metrica fitness)
      +--- last.pt (Checkpoint final da ultima epoca)
```

Figura 16: Representação hierárquica dos arquivos e artefatos de diagnóstico produzidos ao final de um treinamento YOLO.

Estrutura de Arquivos Gerada pelo Treinamento do YOLO (*runs/detect/train/*)

```
.
+--- args.yaml  (Hiperparâmetros e configurações da execução)
+--- BoxF1_curve.png  (Curva F1-Score em função da Confiança)
+--- BoxP_curve.png ..... (Curva de Precisão em função da
Confiança)
+--- BoxPR_curve.png ..... (Curva de Precisão vs.  Revocação /
mAP)
+--- BoxR_curve.png ..... (Curva de Revocação em função da
Confiança)
+--- confusion_matrix.png .... (Matriz de confusão absoluta em
contagem de objetos)
+--- confusion_matrix_normalized.png ..... (Matriz de confusão
normalizada por classe)
+--- labels.jpg ..... (Análise estatística e espacial das
anotações do dataset)
+--- results.csv .. (Tabela com perdas e métricas registradas
época a época)
+--- results.png .... (Painel gráfico de evolução das perdas e
métricas)
+--- train_batch0.jpg ...train_batch2.jpg ..... (Amostras de
imagens de treino com data augmentation)
+--- train_batch11800.jpg ...train_batch11802.jpg ..... (Lotes
finais de treino processados pelo modelo)
+--- val_batch0_labels.jpg . (Anotações reais/ground-truth do
lote de validação)
+--- val_batch0_pred.jpg .. (Predições geradas pelo modelo no
lote de validação)
+--- weights/
+--- best.pt ..... (Checkpoint do modelo com a melhor métrica
fitness)
+--- last.pt ..... (Checkpoint do estado final da última época
treinada)
```

A Tabela 2 categoriza os principais artefatos produzidos segundo a sua funcionalidade técnica no fluxo de desenvolvimento e auditoria de modelos de visão computacional.

Tabela 2: Classificação e descrição funcional dos artefatos do diretório de saída do YOLO.

Categoria	Arquivos	Função e Utilidade Diagnóstica
Configuração e Reprodutibilidade	*args.yaml*	Armazena o registro imutável de todos os hiperparâmetros, caminhos de dados e arquitetura utilizados na sessão.
Checkpoints de Pesos	*weights/best.pt*, *weights/last.pt*	Arquivos binários PyTorch contendo a arquitetura e os parâmetros otimizados para implantação ou retomada.
Logs Numéricos	*results.csv*	Tabela tabular detalhada com o valor numérico exato de cada perda e métrica a cada época de treino.
Gráficos de Convergência	*results.png*	Painel visual com 10 subgráficos monitorando as perdas de treino/validação e a evolução do mAP, Precisão e Revocação.
Curvas de Desempenho	*BoxP_curve.png*, *BoxR_curve.png*, *BoxF1_curve.png*, *BoxPR_curve.png*	Curvas paramétricas em função do limiar de confiança, essenciais para a calibração operacional do modelo em produção.
Diagnóstico Matricial	*confusion_matrix.png*, *confusion_matrix_normalized.png*	Matrizes de confusão absoluta e relativa para identificação de erros de classificação cruzada e confusão com o fundo.
Inspeção de Dados e Batches	*labels.jpg*, *train_batch*.jpg*, *val_batch*_*.jpg*	Imagens compostas para auditoria visual da qualidade das anotações, aumentos de dados (mosaic) e predições.

4. REGISTRO DE CONFIGURAÇÃO E CHECKPOINTS DE PESOS

4.1 O Arquivo de Hiperparâmetros (args.yaml)

A reprodutibilidade científica exige que qualquer experimento de aprendizado profundo possa ser reconstruído exatamente com os mesmos parâmetros operacionais. No ecossistema YOLO, o arquivo *args.yaml* desempenha essa função ao gravar a totalidade dos hiperparâmetros aplicados durante a chamada de treinamento. Entre os parâmetros gravados, destacam-se:

- **Hiperparâmetros de Otimização:** taxa de aprendizado inicial (*lr0*), taxa de aprendizado final (*lrf*), momento (*momentum*), decaimento de pesos (*weight_decay*) e épocas de aquecimento (*warmup_epochs*);
- **Pesos das Funções de Perda:** coeficientes de ponderação para a perda de caixa delimitadora (*box*), perda de classificação (*cls*) e perda focal de distribuição (*dfl*);
- **Aumento de Dados (*Data Augmentation*):** probabilidades e amplitudes de variações no espaço de cores HSV (*hsv_h*, *hsv_s*, *hsv_v*), rotações (*degrees*), translações (*translate*), escala (*scale*), espelhamento horizontal (*fliplr*), corte mosaico (*mosaic*) e mistura de imagens (*mixup*).

O registro em formato YAML permite que frameworks de inferência ou scripts de acompanhamento recarreguem a configuração exata do experimento para posterior ajuste fino (*fine-tuning*).

4.2 Checkpoints de Pesos: best.pt e last.pt

Dentro do subdiretório *weights/*, o YOLO mantém continuamente atualizados dois arquivos binários no formato PyTorch (*.pt*):

1. ***best.pt* (Melhor Modelo):** É o arquivo de pesos correspondente à época em que a rede atingiu o valor máximo da métrica composta de qualidade (*fitness score*). O Ultralytics calcula o *fitness* como uma combinação linear ponderada do mAP@0.50 e do mAP@0.50:0.95:

$$\text{Fitness} = 0{,}1 \cdot \text{mAP@0.50} + 0{,}9 \cdot \text{mAP@0.50:0.95}.$$

A ponderação de 90% atribuída ao mAP@0.50:0.95 prioriza modelos que demonstram alta precisão geométrica na localização das caixas delimitadoras. O arquivo *best.pt* é o artefato indicado para implantação direta em ambiente de produção.

2. ***last.pt* (Último Estado):** Preserva os pesos e os estados do otimizador referentes à última época processada (por exemplo, a época 200 de 200). Este arquivo é fundamental por duas razões:
   - **Retomada de Treinamento (*Resume*):** Permite reiniciar o treinamento exatamente a partir do ponto onde foi interrompido (devido a falhas de energia, estouro de tempo

em clusters ou interrupções voluntárias), utilizando o comando *YOLO("last.pt"). train(resume=True)*;

- **Análise de Sobreajuste (*Overfitting*):** Permite comparar o comportamento do modelo no final da execução em relação ao seu pico de desempenho mantido no *best.pt*.

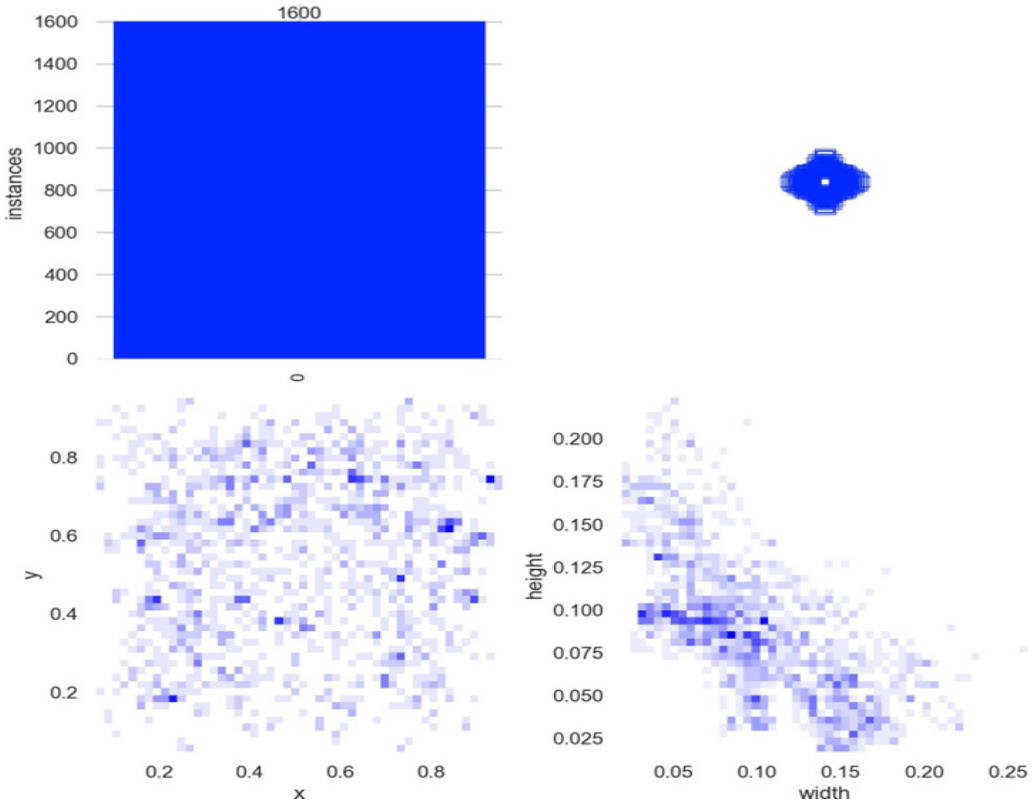
5. INSPEÇÃO QUALITATIVA DOS DADOS E BATCHES (TREINO E VALIDAÇÃO)

Uma premissa fundamental da visão computacional aplicada é a verificação da qualidade dos dados antes e durante o processo de otimização. O YOLO produz automaticamente representações visuais dos dados de entrada e dos lotes (*batches*) processados pela rede.

5.1 Estatísticas Espaciais e Categorias do Dataset (labels.jpg)

O arquivo *labels.jpg* reúne quatro painéis estatísticos que resumem a distribuição categórica e a geometria das anotações contidas no conjunto de treinamento, conforme demonstrado na Figura 17.

Figura 17: Painel estatístico de rótulos do conjunto de dados (labels.jpg). Superior esquerdo: Histograma de contagem por classe. Superior direito: Distribuição espacial e dimensões das caixas delimitadoras. Inferior esquerdo: Mapeamento dos centros (x, y) das caixas. Inferior direito: Relação entre largura (w) e altura (h) dos objetos.



A análise detalhada de cada quadrante do *labels.jpg* fornece diagnósticos valiosos:

- **Histograma de Classes (Superior Esquerdo):** Apresenta a contagem absoluta de instâncias por categoria. Permite identificar o grau de desbalanceamento de classes no dataset, alertando o projetista para a necessidade de técnicas de reponderação ou amostragem balanceada.
- **Distribuição de Caixas por Imagem (Superior Direito):** Mostra a sobreposição e densidade espacial das caixas delimitadoras anotadas no conjunto de dados.
- **Centros das Caixas Delimitadoras (Inferior Esquerdo):** Mapeia as posições relativas dos centros das caixas (*x, y*) normalizadas no intervalo [0, 1]. Concentrações excessivas no centro do gráfico indicam viés de enquadramento (*center bias*), comum quando fotógrafos tendem a centralizar o objeto principal.
- **Dimensões das Caixas (Inferior Direito):** Plota a largura (*w*) versus a altura (*h*) de todas as marcações. Permite verificar a variação de escala dos alvos e a prevalência de objetos pequenos, médios ou grandes, orientando a escolha dos tamanhos de *anchors* ou da resolução de entrada do modelo (*imgsz*).

5.2 Batches de Treinamento e Data Augmentation (train_batch*.jpg)

Durante o treinamento, o gerador de dados do YOLO aplica transformações estocásticas nas imagens de cada lote para aumentar a capacidade de generalização da rede. Os arquivos *train_batch0.jpg*, *train_batch1.jpg* e *train_batch2.jpg* exibem as primeiras amostras do dataset já processadas pelas técnicas de aumento de dados, com as caixas de gabarito sobrepostas. A Figura 18 ilustra um lote inicial de treinamento (*train_batch0.jpg*).

Entre as transformações visíveis em *train_batch0.jpg*, destaca-se o Aumento por Mosaico (*Mosaic Augmentation*), no qual quatro imagens distintas são recortadas, redimensionadas e combinadas em um único painel composto. Essa técnica força o modelo a detectar objetos em contextos visuais variados, em escalas reduzidas e próximos às bordas da imagem, reduzindo drasticamente a dependência da rede quanto ao fundo original das fotografias. A Figura 19 apresenta o lote subsequente (*train_batch1.jpg*), confirmando a variabilidade das transformações aplicadas entre lotes consecutivos.

Figura 18: Visualização do lote de treinamento train_batch0.jpg, demonstrando a aplicação do aumento de dados por colagem Mosaico 4-way (Mosaic Augmentation) e ajuste de espaço de cor HSV.

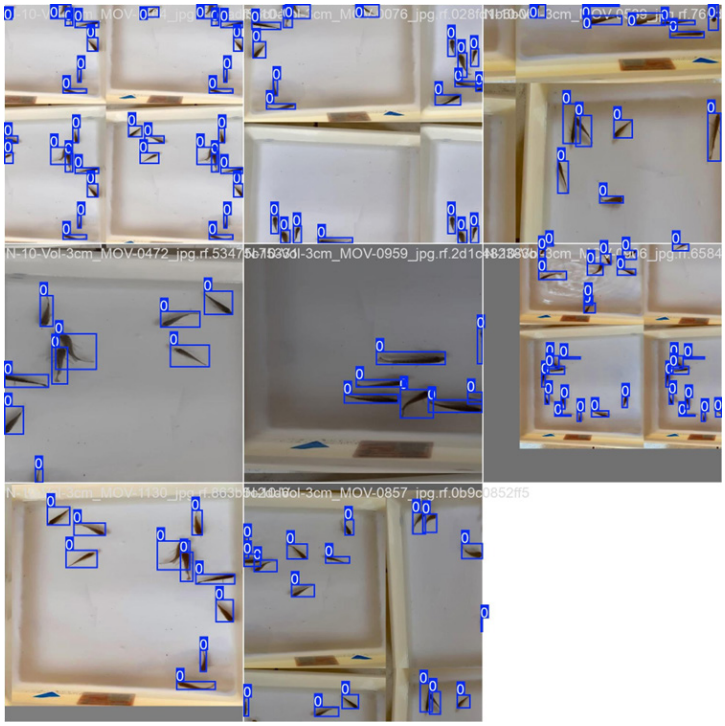
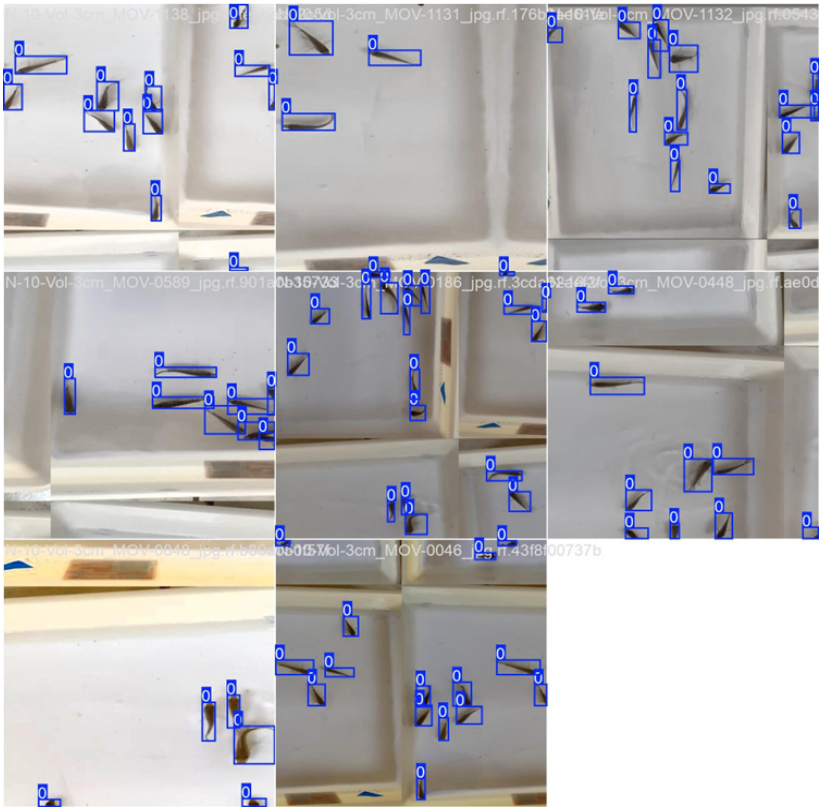


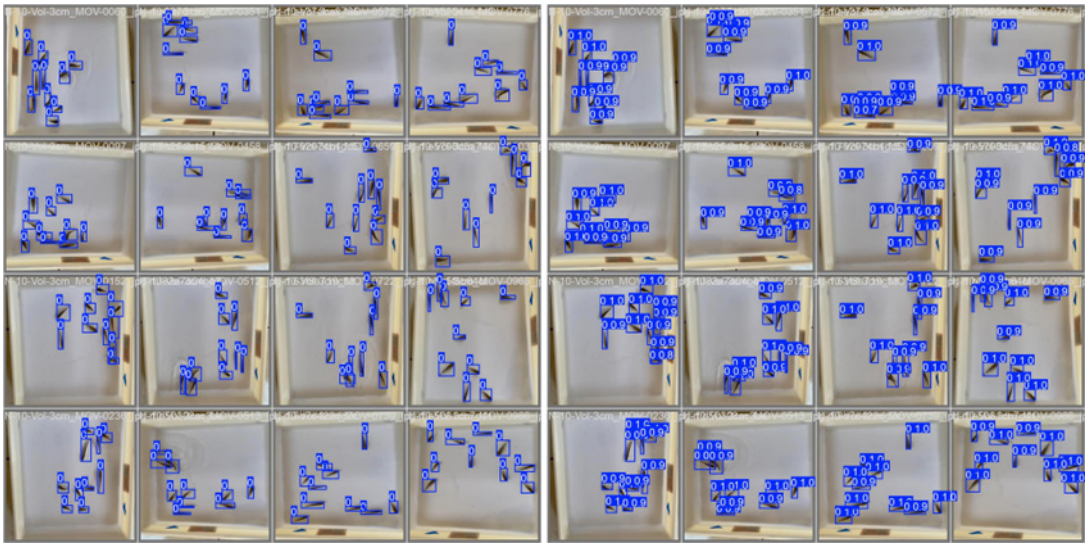
Figura 19: Visualização do lote de treinamento train_batch1.jpg, exibindo diferentes composições de mosaico e variação de iluminação.



5.3 Validação Qualitativa: Gabarito vs. Predições do Modelo

Ao final de cada época ou etapa de validação, o YOLO seleciona lotes do conjunto de validação para gerar duas imagens comparativas: *val_batch0_labels.jpg* (caixas de gabarito reais anotadas por especialistas humanos) e *val_batch0_pred.jpg* (caixas delimitadoras e rótulos previstos pela rede neural, acompanhados das pontuações de confiança). A Figura 20 apresenta a comparação lado a lado entre o gabarito real e a predição gerada pela rede no primeiro lote de validação.

Figura 20: Inspeção qualitativa de validação no lote 0: comparação entre as anotações reais de referência (val_batch0_labels.jpg, à esquerda) e as detecções inferidas pelo modelo YOLO (val_batch0_pred.jpg, à direita).



Essa inspeção visual comparativa é uma ferramenta indispensável para identificar padrões sistemáticos de erro:

1. **Falsos Positivos Visualizáveis:** Caixas presentes na predição que não existem no gabarito revelam regiões do cenário que enganam o modelo;
2. **Falsos Negativos (Omissões):** Objetos marcados no gabarito que não receberam caixa predita indicam alvos não detectados;
3. **Precisão do Ajuste da Caixa:** Permite avaliar se os limites das caixas previstas pela rede se ajustam perfeitamente ao contorno dos objetos ou se apresentam folgas ou cortes significativos.

6. ANÁLISE DETALHADA DAS CURVAS DE DESEMPENHO E LIMIARES DE CONFIANÇA

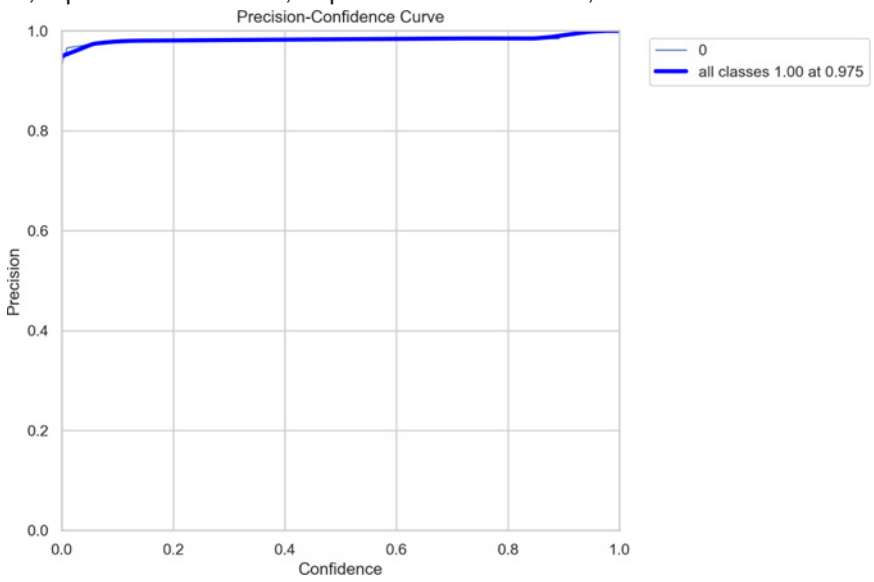
Durante o processo de inferência, o YOLO atribui a cada detecção uma pontuação de confiança conf $\in [0, 1]$, que expressa a probabilidade estimada de a caixa conter um objeto multiplicada pelo IoU previsto. A escolha do limiar de corte de confiança ($conf_{thresh}$) altera radicalmente o equilíbrio entre Precisão e Revocação do sistema. Para orientar essa

calibração, o YOLO gera quatro curvas paramétricas fundamentais.

6.1 Curva Precisão-Confiança (BoxP_curve.png)

A Curva Precisão-Confiança (Figura 21) ilustra a evolução da Precisão (P) à medida que o limiar de confiança mínimo exige predições cada vez mais rigorosas.

Figura 21: Curva Precisão-Confiança (BoxP_curve.png). À medida que o limiar de confiança se aproxima de 1,00, a precisão tende a 1,00 para todas as classes, eliminando Falsos Positivos.

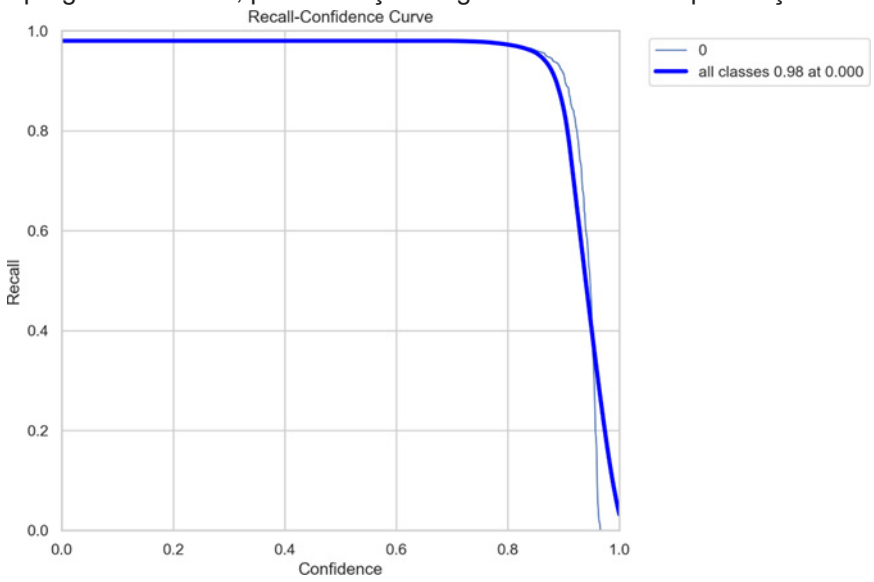


Como regra geral, aumentando o limiar $conf_{thresh}$, o modelo descarta predições incertas, reduzindo o número de Falsos Positivos (FP) e fazendo a Precisão se aproximar de 1,00 (100%). Em aplicações industriais que não toleram alarmes falsos (como controle de qualidade automatizado), seleciona-se um limiar elevado nesta curva.

6.2 Curva Revocação-Confiança (BoxR_curve.png)

Em contrapartida, a Curva Revocação-Confiança (Figura 22) demonstra a variação da Revocação (R) em função do limiar de confiança. Ao elevar o limiar de confiança, o sistema torna-se extremamente seletivo e deixa de detectar objetos que apresentem menor grau de certeza, aumentando os Falsos Negativos (FN) e reduzindo a Revocação. Em sistemas de segurança ou diagnóstico médico, onde falhar na detecção de um alvo é crítico, escolhe-se um limiar de confiança mais baixo nesta curva.

Figura 22: Curva Revocação-Confiança (BoxR_curve.png). Conforme o limiar de confiança aumenta, a revocação cai progressivamente, pois detecções legítimas com menor pontuação são descartadas.



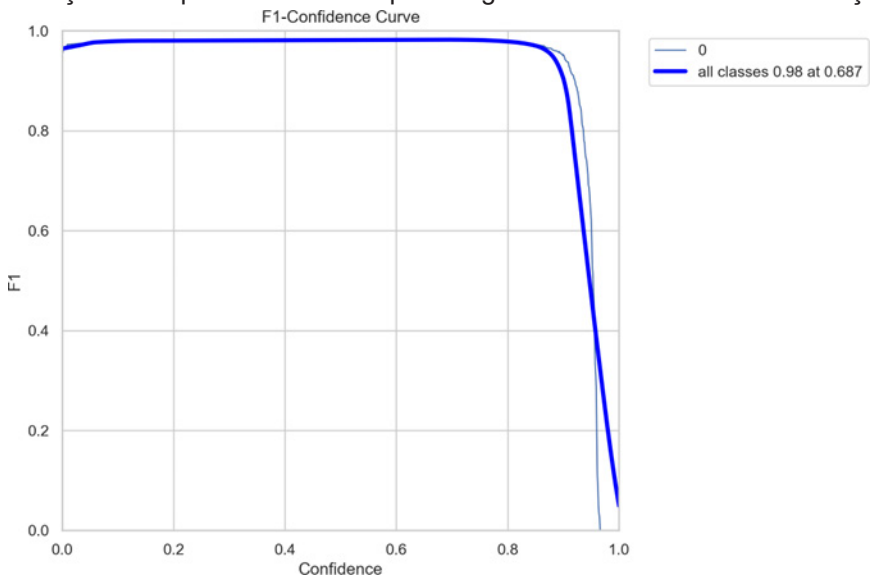
6.3 Curva F1-Score-Confiança (BoxF1_curve.png)

Para encontrar o ponto de equilíbrio ótimo entre Precisão e Revocação sem priorizar arbitrariamente nenhuma das duas métricas, adota-se o F_1-Score, definido pela média harmônica entre P e R:

$$F_1(\text{conf}) = 2 \cdot \frac{P(\text{conf}) \cdot R(\text{conf})}{P(\text{conf}) + R(\text{conf})}$$

A Figura 23 apresenta a curva F_1-Score em função da Confiança (*BoxF1_curve.png*). O gráfico destaca o valor de confiança que maximiza o F_1-score específico desse experimento de exemplo (neste caso, conf = 0,38 com F_1 = 0,89). Ressalta-se que esse valor de limiar de corte não é um padrão fixo universal, devendo ser calibrado dinamicamente para cada aplicação conforme o custo operacional de Falsos Positivos versus Falsos Negativos.

Figura 23: Curva F1-Score-Confiança (BoxF1_curve.png). O ponto de pico da curva indica o limiar de confiança ótimo que maximiza o equilíbrio global entre Precisão e Revocação.



6.4 Curva Precisão-Revocação (BoxPR_curve.png)

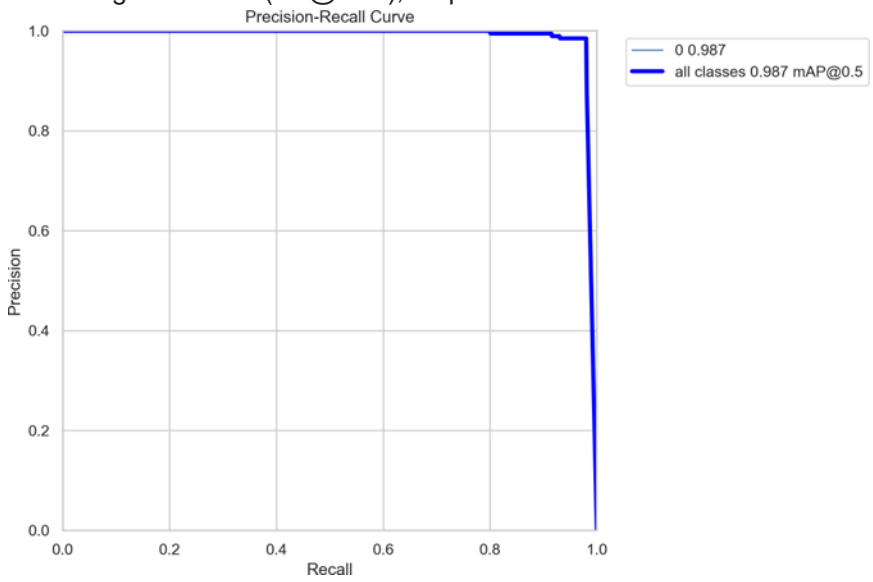
A Curva Precisão-Revocação (Figura 24) sintetiza o *trade-off* direto entre Precisão (eixo Y) e Revocação (eixo X) para todas as categorias de objetos, independentemente do limiar de confiança explícito. Em uma curva PR ideal, a linha estende-se até o canto superior direito (coordenada (1.0, 1.0)), indicando que o modelo alcança alta precisão mantendo alta revocação. A área delimitada abaixo de cada curva individual representa o AP daquela classe específica; o valor global exibido na legenda (mAP@0.50) é a média aritmética das áreas de todas as categorias.

7. EVOLUÇÃO TEMPORAL DAS FUNÇÕES DE PERDA E MÉTRICAS (RESULTS.PNG E RESULTS.CSV)

7.1 Painel Visual de Desempenho (results.png)

O arquivo *results.png* agrega 10 gráficos de linha dispostos em duas fileiras principais, permitindo o acompanhamento temporal simultâneo da convergência das perdas e da ascensão das métricas ao longo de todas as épocas de treinamento, conforme ilustrado na Figura 25.

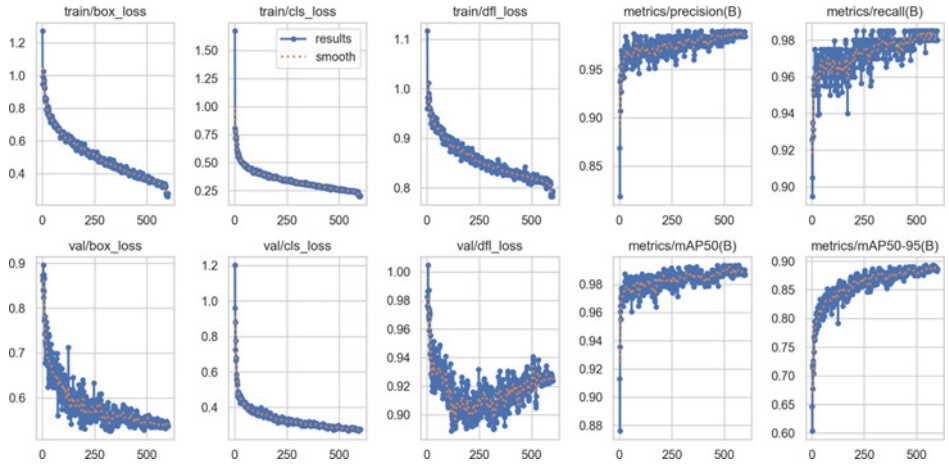
Figura 24: Curva Precisão-Revocação (BoxPR_curve.png). A área sob a curva (AUC) de cada classe corresponde ao seu Average Precision (AP@0.50), enquanto a média entre as classes define o mAP@0.50.



As três funções de perda monitoradas pelo YOLO possuem papéis matemáticos distintos na otimização da rede:

1. **Perda de Caixa (*Box Loss* — L_{box}):** Mensura o erro de alinhamento espacial entre as caixas delimitadoras previstas e os gabaritos reais. Utiliza a métrica Complete IoU (CIoU), que penaliza diferenças de posição central, sobreposição de área e proporção de aspecto (*aspect ratio*);
2. **Perda de Classificação (*Class Loss* — L_{cls}):** Avalia a acurácia na identificação das categorias dos objetos utilizando a perda de Entropia Cruzada Binária (*Binary Cross-Entropy Loss* — BCE);
3. **Perda Focal de Distribuição (*Distribution Focal Loss* — L_{dfl}):** Utilizada nas arquiteturas YOLO de última geração (YOLOv8/YOLOv12) para tratar a incerteza nos limites exatos das bordas dos objetos quando a imagem apresenta oclusão ou contornos difusos, otimizando uma distribuição de probabilidade contínua sobre as posições dos limites.

Figura 25: Painel completo de evolução temporal do treinamento (results.png). Linha 1: Perdas de treinamento (train/box_loss, train/cls_loss, train/dfl_loss) e métricas (metrics/precision, metrics/recall). Linha 2: Perdas de validação (val/box_loss, val/cls_loss, val/dfl_loss) e métricas de mAP (metrics/mAP50, metrics/mAP50-95).



7.2 Diagnóstico de Convergência e Sobreajuste (Overfitting)

A comparação direta entre a fileira de perdas de treino (*train/*) e a fileira de perdas de validação (*val/*) no gráfico *results.png* fornece o diagnóstico definitivo sobre a dinâmica do aprendizado:

- **Aprendizado Saudável (Convergência Estável):** As perdas de treino e de validação decrescem de forma suave e paralela ao longo das épocas, enquanto o mAP50 e o mAP50-95 sobem continuamente até atingirem um patamar de estabilização (*plateau*);
- **Sintoma de Sobreajuste (*Overfitting*):** A perda de treino continua caindo, mas a perda de validação estabiliza e começa a subir a partir de determinada época, acompanhada da estagnação ou queda do mAP de validação. Nesse cenário, o modelo memorizou os padrões do treino mas perdeu a capacidade de generalizar;
- **Sintoma de Subajuste (*Underfitting*):** Todas as perdas mantêm valores elevados ao final das épocas programadas, indicando que o modelo necessita de mais épocas de treinamento, maior capacidade arquitetural ou taxas de aprendizado ajustadas.

7.3 Registro Numérico Estruturado (results.csv)

Enquanto o arquivo *results.png* destina-se à inspeção visual rápida, o arquivo *results.csv* armazena os valores numéricos exatos de todas as variáveis monitoradas em cada época no formato tabular CSV (*Comma-Separated Values*). Cada linha do *results.csv* contém as seguintes colunas padronizadas:

epoch, train/box_loss, train/cls_loss, train/dfl_loss, metrics/precision, metrics/recall, metrics/mAP50, metrics/mAP50-95, val/box_loss, val/cls_loss, val/dfl_loss, lr/pg0, lr/pg1, lr/pg2

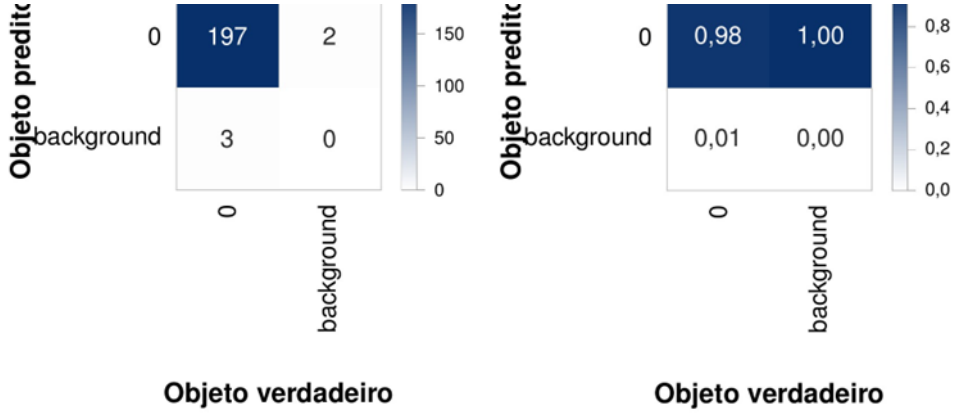
Esse registro tabular permite que pesquisadores importem os dados em bibliotecas de análise científica (como Pandas, NumPy e Seaborn) para gerar gráficos customizados, realizar análises estatísticas comparativas entre diferentes arquiteturas ou aplicar algoritmos de decisão automatizada para parada antecipada (*early stopping*).

8. MATRIZ DE CONFUSÃO ABSOLUTA VS. NORMALIZADA

O diretório de saída do treinamento gera duas versões suplementares da matriz de confusão: *confusion_matrix.png* (contagens absolutas) e *confusion_matrix_normalized.png* (proporções relativas).

A Figura 26 apresenta a comparação visual entre essas duas modalidades de diagnóstico matricial.

Figura 26: Comparação entre a matriz de confusão em contagens brutas de instâncias (confusion_matrix.png, à esquerda) e a matriz normalizada por classe real (confusion_matrix_normalized.png, à direita).



(a) Matriz de Confusão Absoluta (*confusion_matrix.png*)

(b) Matriz Normalizada (*confusion_matrix_normalized.png*)

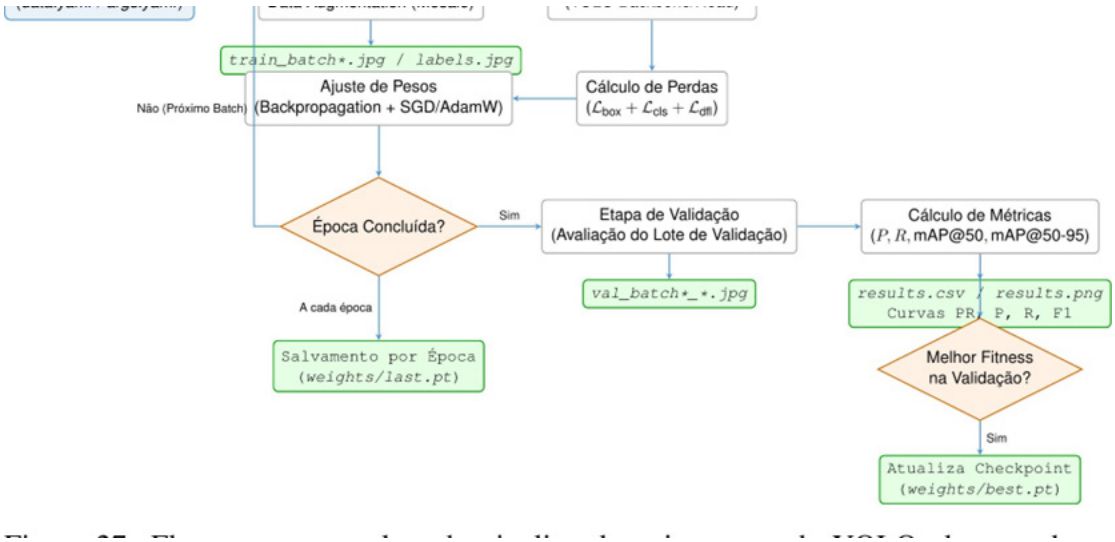
A utilidade complementar de cada versão reside na escala de análise:

1. **Matriz de Contagens Absolutas (*confusion_matrix.png*):** Exibe o número exato de instâncias físicas detectadas. É indispensável para avaliar a escala do dataset e identificar o volume absoluto de erros em categorias críticas;
2. **Matriz Normalizada (*confusion_matrix_normalized.png*):** Divide o valor de cada célula pela soma total de instâncias pertencentes àquela classe real, expressando a taxa de acertos e erros no intervalo [0, 1]. Esta versão é fundamental para conjuntos de dados desbalanceados, pois evita que a classe com maior número de exemplares domine a percepção visual do gráfico, permitindo avaliar a verdadeira taxa de sensibilidade de classes minoritárias.

9. FLUXOGRAMA INTEGRADO DO PIPELINE DE TREINAMENTO E DIAGNÓSTICO

Para sintetizar o ciclo de vida operacional da avaliação de modelos YOLO, a Figura 27 apresenta o fluxograma completo do pipeline de treinamento, validação época a época, geração de artefatos de saída e tomada de decisão para implantação final.

Figura 27: Fluxograma completo do pipeline de treinamento do YOLO, destacando as etapas de aumentos de dados, otimização de perdas, validação época a época e exportação dos artefatos de diagnóstico e checkpoints.



Em suma, a interpretação sistemática do conjunto de artefatos gerados pelo treinamento do YOLO permite transformar o processo de desenvolvimento de redes profundas de uma abordagem empírica de "caixa preta" em um fluxo de engenharia auditável, rigoroso e previsível.

REFERÊNCIAS

EVERINGHAM, Mark et al. The PASCAL Visual Object Classes (VOC) Challenge. International Journal of Computer Vision, v. 88, n. 2, p. 303–338, 2010.

FRANÇA, Elzi Leida do Carmo et al. Detecção de abelhas nativas do Cerrado Brasileiro com Rede Neural Convolucional YOLOv8. Caderno Pedagógico, v. 22, n. 11, e19970, 2025. DOI: https://doi.org/10.54033/cadpedv22n11-207.

PADILLA, Rafael; NETTO, Sergio L.; SILVA, Eduardo A. B. da. A Survey on Performance Metrics for Object-Detection Algorithms. In: INTERNATIONAL Conference on Systems, Signals and Image Processing (IWSSIP). [S.l.: s.n.], 2020. P. 237–242.

CAPÍTULO 5

GUIA DIDÁTICO DE INFRAESTRUTURA E HARDWARE PARA TREINAMENTO

HARDWARE PARA TREINAMENTO

1. INTRODUÇÃO: POR QUE PRECISAMOS DE HARDWARE ESPECIALIZADO?

No desenvolvimento de sistemas de Visão Computacional apoiados em Redes Neurais Convolucionais profundas (como as arquiteturas da família YOLO), o dimensionamento adequado da infraestrutura de computação é um fator determinante para a viabilidade dos projetos. O processo de treinamento envolve o ajuste iterativo de milhões de parâmetros (pesos e viéses) por meio do algoritmo de retropropagação do erro (backpropagation), exigindo o processamento repetido de milhares de imagens em alta resolução ao longo de dezenas ou centenas de épocas. Se esse treinamento for executado utilizando exclusivamente a Unidade Central de Processamento (CPU) de um computador convencional, o tempo de execução torna-se proibitivo para a prática analítica. Essa limitação decorre das diferenças arquiteturais entre os processadores: A CPU (como os processadores Intel Core i7/i9 ou AMD Ryzen 7/9) é otimizada para a execução sequencial de rotinas de alta complexidade lógica e controle de fluxo, operando com poucos núcleos de alto clock. Por outro lado, o treinamento de redes convolucionais baseia-se na execução massiva e paralela de operações matriciais simples (multiplicações de matrizes e somas de tensores). É nesse domínio que a Unidade de Processamento Gráfico (GPU) sobressai. Enquanto uma CPU de alto desempenho possui tipicamente entre 8 e 24 núcleos físicos de processamento, uma GPU moderna conta com milhares de pequenos núcleos de processamento paralelo (os chamados CUDA Cores ou Tensor Cores na arquitetura NVIDIA). Como ilustração do impacto de desempenho, o treinamento de uma rede YOLOv8 sobre um conjunto de 10.000 imagens que demandaria cerca de 5 a 7 dias ininterruptos em CPU pode ser concluído em aproximadamente 2 a 3 horas quando executado em uma GPU de alto desempenho dedicada. Assim, o emprego de GPUs não se limita a um ganho incremental de velocidade, mas constitui o requisito tecnológico fundamental para viabilizar o ciclo de desenvolvimento e experimentação em Aprendizado Profundo. Neste capítulo, apresenta-se um guia técnico para o dimensionamento e montagem de estações de trabalho (workstations) e servidores dedicados ao treinamento de redes neurais, prevenindo gargalos de transferência e otimizando o fluxo de trabalho.

2. FINALIDADES E O IMPACTO DA INFRAESTRUTURA DE PROCESSAMENTO

O dimensionamento equilibrado da infraestrutura computacional traz benefícios diretos em três frentes principais:

2.1 Aceleração da Pesquisa Científica

Na pesquisa aplicada ao Aprendizado de Máquina, a validação de hipóteses exige ciclos frequentes de experimentação empírica. O pesquisador necessita avaliar diferentes variantes arquiteturais, ajustar taxas de aprendizado e testar estratégias de aumento de dados (data augmentation). Uma infraestrutura dedicada permite realizar múltiplos experimentos diários, acelerando a obtenção de resultados auditáveis e a consolidação de publicações científicas.

2.2 Desenvolvimento de Aplicações Industriais e Produtos (MVP)

No desenvolvimento de soluções comerciais (como monitoramento agrícola, inspeção industrial ou controle de tráfego), o processamento acelerado viabiliza a rápida prototipagem de Produtos Mínimos Viáveis (MVPs). O ciclo de melhoria contínua é favorecido, pois novos lotes de dados coletados em campo podem ser incorporados e re-treinados rapidamente.

2.3 Capacitação Acadêmica e Ensino Prático

No ensino de Engenharia de Software e Ciência da Computação, o acesso a ambientes computacionais de alto desempenho permite que os estudantes vivenciem a construção prática de modelos profundos com conjuntos de dados reais, superando abordagens puramente teóricas ou limitadas a pequenos exemplos didáticos.

3.ESPECIFICAÇÕES TÉCNICAS E PREVENÇÃO DE GARGALOS (I/OBOTTLENECK)

Um erro comum no projeto de sistemas para inteligência artificial é alocar a maior parte do orçamento na aquisição de uma GPU potente e negligenciar os demais componentes do sistema. Se o armazenamento for lento ou a quantidade de memória RAM for insuficiente, ocorre o chamado gargalo de E/S (Input/Output bottleneck): a GPU permanece ociosa aguardando a CPU carregar, decodificar e transferir os lotes de imagens a partir do disco. Para garantir a integração plena do sistema, a Tabela 3 especifica o papel técnico dos componentes recomendados para uma estação de trabalho (workstation) ou servidor de desenvolvimento em IA. A Figura 28 exemplifica a disposição estrutural de uma estação de trabalho configurada para alta dissipação térmica e acomodação de placas de vídeo de grande porte.

Figura 28: Disposição física de uma estação de trabalho voltada à Visão Computacional, destacando o sistema de refrigeração e o espaço para alocação de GPUs dedicadas. Por fim, cabe diferenciar a aplicação de uma Estação de Trabalho (Workstation) — voltada ao desenvolvimento local, depuração de código e testes interativos por um único usuário — de um Servidor de Treinamento em Nuvem/Cluster, que atua em uma infraestrutura para treinamento de redes neurais sem gargalos.



Componente

Especificação Recomendada e Propósito Técnico Placa de Vídeo (GPU NVIDIA) NVIDIA GeForce RTX 4080 SUPER ou RTX 5080 (16 GB VRAM): A Tabela 3: Especificações técnicas recomendadas e o papel funcional de cada componente em uma infraestrutura para treinamento de redes neurais sem gargalos.

Componente	Especificação Recomendada e Propósito Técnico
Placa de Vídeo (GPU NVIDIA)	**NVIDIA GeForce RTX 4080 SUPER ou RTX 5080 (16 GB VRAM):** A memória de vídeo determina a resolução limite e o tamanho máximo do lote (*batch size*) sem erros de *Out of Memory* (OOM). Suporte CUDA nativo.
Processador (CPU)	**AMD Ryzen 9 7900X3D / 9900X3D (12 núcleos, 24 threads):** Executa o carregamento das imagens em disco, decodificação, Data Augmentation em tempo real e enfileiramento dos tensores (*workers=8*).
Memória RAM	**32 GB a 64 GB DDR5 (5600 MHz):** Memória intermediária para alocação dos DataLoaders e tratamento de conjuntos volumosos de dados.
Armazenamento	**SSD M.2 NVMe 2 TB (ex: Samsung 990 Evo Plus):** Taxas de leitura sequencial e aleatória superiores a 5.000 MB/s, evitando gargalos de E/S.
Placa-mãe	**Gigabyte B650M AORUS ELITE ou Asus X870:** VRMs robustos e dissipadores térmicos para suportar altas cargas contínuas.
Fonte de Alimentação	**850W a 1000W (Certificação 80 Plus Gold / Platinum):** Fornecimento estável de energia para picos de consumo em carga máxima.

Sistema de Refrigeração	**Water Cooler de 360 mm ou Air Cooler de Alta Performance:** Mantém CPU e GPU dentro de limites seguros, evitando *thermal throttling*.
Gabinete	**Gabinete E-ATX com Alto Fluxo de Ar (*High Airflow*):** Espaço físico e ventilação adequada para placas de vídeo de grande porte.
Sistema Operacional	**Ubuntu 22.04 LTS / 24.04 LTS (Linux):** Plataforma padrão na comunidade científica de IA com suporte maduro a drivers CUDA.

capacidade de memória de vídeo (VRAM) determina a resolução limite e o tamanho máximo do lote (batch size) processável simultaneamente. Com 16 GB, é possível treinar modelos YOLO intermediários e avançados com resoluções de 640 × 640 pixels ou superiores sem ocorrência de erros de falta de memória (Out of Memory - OOM). O ecossistema NVIDIA é recomendado devido ao suporte nativo da arquitetura CUDA e das bibliotecas cuDNN/ cuBLAS no framework PyTorch.

Processador (CPU)

AMD Ryzen 9 7900X3D / 9900X3D (12 núcleos, 24 threads): A CPU desempenha papel logístico crítico: executa o carregamento das imagens em disco, a decodificação dos arquivos gráficos, a aplicação de aumentos de dados em tempo real (mosaic, rotação, transformações HSV) e o enfileiramento dos tensores via subprocessos (workers=8) para a GPU.

Memória RAM

32 GB a 64 GB DDR5 (5600 MHz): Atua como a memória intermediária para alocação dos DataLoaders do PyTorch e tratamento de grandes conjuntos de imagens durante a fase de treino.

Armazenamento

SSD M.2 NVMe 2 TB (ex: Samsung 990 Evo Plus): Recomenda-se o uso de SSDs NVMe em substituição a discos rígidos mecânicos (HDDs) ou SSDs SATA, visando atingir taxas de leitura sequencial e aleatória elevadas (acima de 5.000 MB/s), indispensáveis para ler milhares de pequenas imagens e arquivos de marcação no formato YOLO (.txt) ou Pascal VOC (.xml) sem retenção de E/S.

Placa-mãe

Gigabyte B650M AORUS ELITE ou Asus X870: Deve apresentar Módulos Reguladores de Voltagem (Voltage Regulator Modules – VRMs) robustos e dissipadores térmicos adequados para suportar a demanda energética contínua sob cargas prolongadas de processamento.

Fonte de Alimentação

Fonte 850W a 1000W (Certificação 80 Plus Gold / Platinum): Garante o fornecimento contínuo e estável de corrente elétrica para picos de consumo da GPU e CPU em regime

de carga máxima.

Sistema de Refrigeração

Water Cooler de 360 mm ou Air Cooler de Alta Performance: Mantém as temperaturas operacionais da CPU e GPU dentro de limites seguros, evitando a redução automática de frequência por proteção térmica (thermal throttling).

Gabinete

Gabinete E-ATX com Alto Fluxo de Ar (High Airflow): Proporciona o espaço físico necessário para acomodar GPUs de grande porte e garante a circulação contínua de ar frio.

Sistema Operacional

Ubuntu 22.04 LTS / 24.04 LTS (Linux): Plataforma padrão na comunidade de Inteligência Artificial, oferecendo melhor estabilidade de drivers CUDA, otimização de bibliotecas Python e gerenciamento eficiente de memória de processo. de forma remota e compartilhada para múltiplos usuários. Ambas as estruturas compartilham os mesmos princípios de dimensionamento equilibrado de GPU, CPU, RAM e barramento de armazenamento para garantir máxima eficiência no ciclo de treinamento do YOLO.

REFERÊNCIAS

AMD. AMD Ryzen 9 9900X3D: especificações do processador. [S. l.]: AMD, [s. d.]. Disponível em: https://www.amd.com/pt/products/processors/desktops/ryzen/9000-series/amd-ryzen-9-9900x3d.html

CANONICAL. Ubuntu 24.04 LTS. [S. l.]: Canonical, 2024. Disponível em: https://documentation.ubuntu.com/release-notes/24.04/

NVIDIA. GeForce RTX 5080: especificações técnicas. Santa Clara: NVIDIA, [s. d.]. Disponível em: https://www.nvidia.com/pt-br/geforce/graphics-cards/50-series/rtx-5080/

ULTRALYTICS. Model training with Ultralytics YOLO. [S. l.]: Ultralytics, [s. d.]. Disponível em: https://docs.ultralytics.com/modes/train/

CAPÍTULO 6

BACKUP AUTOMATIZADO E OTIMIZADO COM RCLONE E PYTHON

1. O CENÁRIO DE RISCO E A NECESSIDADE DE BACKUP

O treinamento de redes neurais profundas, como os modelos da família YOLO, demanda tempo de processamento e gera múltiplos arquivos durante a execução: checkpoints de pesos (*best.pt* e *last.pt*), matrizes de confusão, curvas de precisão e revocação, imagens de validação e logs tabulares (*results.csv*). Interrupções de energia ou falhas na máquina host durante o processo podem corromper os arquivos locais e causar a perda de horas ou dias de processamento.

O envio manual ou a sincronização padrão para serviços de nuvem (como o Google Drive) enfrenta dois obstáculos práticos:

1. **Volume de arquivos pequenos:** as pastas de predições e logs contêm milhares de arquivos de poucos kilobytes. Serviços de nuvem impõem limites de transações por segundo (TPS) e rejeitam conexões que abrem muitas requisições simultâneas.
2. **Tamanho dos checkpoints:** os arquivos *best.pt* e *last.pt* ocupam dezenas ou centenas de megabytes. Conexões interrompidas exigem retomada automática sem corrupção do arquivo.

Para contornar esses limites, o Rclone é uma opção adequada para automatizar o envio.

2. O QUE É O RCLONE?

O Rclone é um utilitário de linha de comando para transferência e sincronização de arquivos entre sistemas locais e provedores de nuvem, com suporte a mais de 70 serviços (incluindo Google Drive, Amazon S3, Dropbox e OneDrive). Na sincronização de saídas do YOLO, seus principais recursos são:

- **Limite de requisições (*–tpslimit*):** restringe as transações por segundo para evitar o bloqueio de API pelo provedor.
- **Transferências simultâneas (*–transfers*):** gerencia o envio paralelo de arquivos, aproveitando a largura de banda de upload disponível.
- **Comparação por data de modificação e hash:** valida se o arquivo precisa ser atualizado pelo timestamp e pela assinatura. Como os arquivos *best.pt* e *last.pt* mantêm o mesmo tamanho em bytes entre épocas, a flag *–size-only* não deve ser usada, pois impediria a cópia dos novos pesos.
- **Tentativas automáticas (*–retries*):** repete a transferência após falhas momentâneas de conexão.

3. INTEGRAÇÃO ENTRE PYTHON E RCLONE

A execução do Rclone pode ser acoplada ao treinamento por meio de um script Python. O uso de *callbacks* da biblioteca Ultralytics permite disparar a sincronização ao final de cada época, preservando os pesos mais recentes na nuvem caso o computador seja desligado.

3.1 Script de Treinamento com Sincronização Periódica

O script a seguir configura as opções de GPU (cuDNN, AMP e TF32), instancia o modelo *yolov8s.pt*, registra a rotina de envio ao término de cada época e executa uma cópia final ao fechar o processo:

Script: *orquestrador_yolo_rclone.py*

```
import os
import site
import sys
import re
import time
import subprocess
from pathlib import Path

# Configuracao de bibliotecas nativas C++/CUDA para o ambiente
try:
    site_pkgs = site.getsitepackages()[0]
    cudnn_lib_path = os.path.join(site_pkgs, "nvidia", "cudnn", "lib")
    cublas_lib_path = os.path.join(site_pkgs, "nvidia", "cublas", "lib")
    ld_path = os.environ.get("LD_LIBRARY_PATH", "")
    os.environ["LD_LIBRARY_PATH"] = f"{cudnn_lib_path}:{cublas_lib_path}:{ld_path}"
    print("[OK] Variaveis do ambiente de suporte CUDA validadas")
```

```
except Exception as e:
    print(f"Aviso ao verificar caminhos CUDA: {e}")

try:
    import torch
    print("[OK] PyTorch importado com sucesso")
except Exception as e:
    print(f"Erro ao importar PyTorch: {e}")
    raise

try:
    from ultralytics import YOLO
    print("[OK] Ultralytics importado com sucesso")
except Exception as e:
    print(f"Erro ao importar Ultralytics: {e}")
    raise

# Habilitacao de cuDNN, AMP e precisao TF32 para alto desempenho
torch.backends.cudnn.enabled = True
torch.backends.cudnn.benchmark = True
if hasattr(torch.backends.cuda.matmul, "allow_tf32"):
    torch.backends.cuda.matmul.allow_tf32 = True
if hasattr(torch.backends.cudnn, "allow_tf32"):
    torch.backends.cudnn.allow_tf32 = True

base_dir = Path(".")
datasets_dir = base_dir / "datasets_extraidos"
modelos_dir = base_dir / "modelos"
output_dir = base_dir / "treinamentos"

# Nome correto do arquivo de pesos oficiais da Ultralytics
modelos_para_treinar = {
    "v8": "yolov8s.pt"
}

EPOCHS = 200
PATIENCE = 50
IMGSZ = 640
BATCH = 16
DEVICE = 0
WORKERS = 8
```

```
RCLONE_REMOTE = "gdrive"
RCLONE_PASTA = "BackupsYOLO/treinamentos"
RCLONE_TPS_LIMIT = "4"
RCLONE_TRANSFERS = "2"
RCLONE_CHECKERS = "2"
RCLONE_DELAY = 5

log_file = output_dir / "historico.txt"
log_file.parent.mkdir(parents=True, exist_ok=True)

def registrar_log(msg):
    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
    with open(log_file, "a", encoding="utf-8", buffering=1) as f:
        f.write(f"[{timestamp}] {msg}\n")
        f.flush()
        os.fsync(f.fileno())
    print(f"[{timestamp}] {msg}", flush=True)

def sanitizar_nome_pasta(nome: str):
    nome_limpo = re.sub(r"[\r\n]+", " ", nome)
    nome_limpo = re.sub(r"[^\w\s-]", "_", nome_limpo)
    return re.sub(r"\s+", "_", nome_limpo).strip("_")

def verificar_rclone():
    try:
        resultado = subprocess.run(
            ["rclone", "version"],
            capture_output=True,
            text=True
        )
        return resultado.returncode == 0
    except Exception as e:
        registrar_log(f"Rclone nao encontrado no sistema: {e}")
        return False

def sincronizar_treinamento_gdrive(pasta_treinamento: Path, dataset_nome: str, versao: str):
    if not rclone_disponivel or not pasta_treinamento.exists() or not pasta_treinamento.is_dir():
        return False
    destino = f"{RCLONE_REMOTE}:{RCLONE_PASTA}/{pasta_treinamento.name}"
    max_tentativas = 5
    tempos_espera = [30, 60, 120, 240, 300]
    for tentativa in range(1, max_tentativas + 1):
```

```
        try:
            # Nota: NAO utilizar --size-only em arquivos .pt!
            comando = [
            "rclone", "copy", str(pasta_treinamento), destino, "--transfers", RCLONE_TRANSFERS, "--checkers",
            RCLONE_CHECKERS, "--tpslimit", RCLONE_TPS_LIMIT, "--tpslimit-burst", "1",
            "-P", "--stats", "30s", "--log-level", "INFO"
            ]
            subprocess.run(comando,  check=True)
            registrar_log(f"Backup Rclone concluido para: {dataset_nome}/{versao}")
            return True




        except subprocess.CalledProcessError as e:
            registrar_log(f"Tentativa {tentativa}/{max_tentativas} de backup falhou: {e}")
            if tentativa < max_tentativas:
                time.sleep(tempos_espera[tentativa - 1])
            else:
                registrar_log(f"ERRO CRITICO: Falha definitiva de backup para {dataset_nome}")
                return False
        except Exception as e:
            registrar_log(f"Excecao inesperada durante backup: {e}")
            return False
    return False


def criar_data_yaml_padrao(pasta_dataset: Path) -> Path:
    yaml_path = pasta_dataset / "data.yaml"
    if yaml_path.exists():
        return yaml_path
    train_images = pasta_dataset / "images" / "train"
    val_images = pasta_dataset / "images" / "val"
    # Exige a presenca simultanea dos diretorios de treino e validacao
    if not (train_images.exists() and val_images.exists()):
        return None
    yaml_content = f"# Auto-generated data.yaml\npath: {pasta_dataset.resolve().as_posix()}\n"
    yaml_content += "train: images/train\nval: images/val\n"
    yaml_content += "\nnc: 2\nnames: ['class0', 'class1']\n"
    try:
        with open(yaml_path, "w", encoding="utf-8") as f:
            f.write(yaml_content)
        return yaml_path
```

```
        except Exception as e:
            registrar_log(f"Erro ao criar data.yaml em {pasta_dataset}: {e}")
            return None

configuracoes_data = []
if datasets_dir.exists():
    for pasta in sorted(datasets_dir.iterdir()):
        if not pasta.is_dir():
            continue
        yamls_encontrados = list(pasta.rglob("data.yaml"))
        if not yamls_encontrados:
            yaml_path = criar_data_yaml_padrao(pasta)
            if yaml_path is None:
                continue
            yamls_encontrados = [yaml_path]
        configuracoes_data.append({
            "nome": pasta.name,
            "nome_sanitizado": sanitizar_nome_pasta(pasta.name),
            "path": str(yamls_encontrados[0].resolve()),
            "task": "detect"
        })

rclone_disponivel = verificar_rclone()

for dataset in configuracoes_data:
    for versao, arquivo_modelo in modelos_para_treinar.items():
        caminho_modelo = modelos_dir / arquivo_modelo
        projeto_dir = output_dir / dataset["nome_sanitizado"]
        nome_run = versao
        pasta_treinamento = projeto_dir / nome_run

        # Callback para disparar backup periodico ao termino de cada epoca
        def on_fit_epoch_end(trainer):
            if rclone_disponivel and (trainer.epoch + 1) % 10 == 0:
                sincronizar_treinamento_gdrive(
                    pasta_treinamento, dataset["nome"], versao
                )

        try:
            # Se o arquivo local nao existir, o Ultralytics realiza o download
            fonte_modelo = str(caminho_modelo) if caminho_modelo.exists() else arquivo_modelo model =
            YOLO(fonte_modelo)
            model.add_callback("on_fit_epoch_end", on_fit_epoch_end)
```

```
                model.train(
                    data=dataset["path"],
                    task=dataset["task"],
                    epochs=EPOCHS, patience=PATIENCE,
                    imgsz=IMGSZ, batch=BATCH,
                    device=DEVICE, workers=WORKERS,
                    plots=True, amp=True,
                    project=str(projeto_dir), name=nome_run, exist_ok=True
                )
                registrar_log(f"Treinamento concluido com sucesso: {dataset['nome']}/{versao}")
            except Exception as e:
                registrar_log(f"Erro durante treinamento de {dataset['nome']}: {e}")
            finally:
                if rclone_disponivel:
                    time.sleep(RCLONE_DE
                    LAY)
                    sincronizar_treinamento_g
                    drive(
                        pasta_treinamento, dataset["nome"], versao
```

4. ENTENDENDO O CÓDIGO

A rotina combina o treinamento do detector com a transferência remota tolerante a falhas. Os principais blocos do script operam da seguinte forma:

4.1 Aceleração de Hardware (cuDNN, AMP e TF32)

O parâmetro *torch.backends.cudnn.enabled = True* ativa as rotinas de convolução otimizadas da cuDNN. A precisão mista (*amp=True*) reduz o consumo de memória de vídeo e acelera o processamento mantendo a estabilidade numérica. Em GPUs com arquitetura Ampere ou superior (RTX 30xx/40xx/50xx), a opção *allow_tf32 = True* acelera as multiplicações matriciais em ponto flutuante. A variável *LD_LIBRARY_PATH* deve ser configurada no ambiente do sistema operacional antes da chamada do script para permitir que o vinculador dinâmico (*ld.so*) localize as bibliotecas compartilhadas (*.so*) da NVIDIA.

4.2 Variáveis de Configuração e Parâmetros do Rclone

A configuração do Rclone define limites operacionais para a conexão:

- *RCLONE_REMOTE = "gdrive"*: nome do acesso remoto previamente configurado via terminal (*rclone config*).
- *RCLONE_TPS_LIMIT = "4"*: teto de requisições por segundo para prevenir bloqueios temporários de cota no Google Drive.
- *RCLONE_TRANSFERS = "2"*: envio paralelo de até 2 arquivos por ciclo.

4.3 Auto-Configuração do Arquivo data.yaml

A função *criar_data_yaml_padrao()* verifica a existência do arquivo de configuração do conjunto de dados. Se ele não for encontrado, a função valida a presença simultânea das pastas *images/train* e *images/val* (layout padronizado no Capítulo 3) e gera um arquivo *data.yaml* válido com o número de classes (*nc: 2*) coerente com a lista de nomes *['class0', 'class1']* delimitada por aspas simples retas.

4.4 Tolerância a Falhas e Descarte Obrigatório da Flag –size-only

O envio dos dados é orquestrado pela função *sincronizar_treinamento_gdrive()*, que combina o disparo do Rclone via *subprocess.run* com re-tentativas temporizadas em intervalos incrementais (30, 60, 120, 240 e 300 segundos) em caso de oscilação da conexão.

É necessário **não utilizar a flag *–size-only*** ao sincronizar repositórios do YOLO. Como os arquivos binários de checkpoints (*best.pt* e *last.pt*) contêm a mesma estrutura de camadas e contagem de parâmetros, seu tamanho em bytes permanece rigorosamente constante entre épocas sucessivas de treinamento. A flag *–size-only* levaria o Rclone a assumir incorretamente que o arquivo na nuvem já está atualizado, ignorando o envio dos novos pesos otimizados. Por essa razão, utiliza-se a comparação padrão baseada no horário de modificação (ModTime) e hash.

Por fim, o *callback on_fit_epoch_end* aciona a sincronização a cada época finalizada. Em caso de interrupção por falta de energia, os checkpoints na nuvem permitem retomar a execução a partir da última época sincronizada.

REFERÊNCIAS

JOCHER, Glenn; CHAURASIA, Ayush; QIU, Jing. Ultralytics YOLO. [S.l.: s.n.], 2023. Software de detecção, segmentação, classificação e estimativa de pose baseado na família YOLO. Disponível em: https://github.com/ultralytics/ultralytics.

RCLONE AUTHORS. Rclone: rsync for cloud storage. 2026. Disponível em: https://rclone.org.

CAPÍTULO 7

CONCLUSÃO, CICLO DE VIDA E IMPLANTAÇÃO

1. SÍNTESE DO CICLO DE DESENVOLVIMENTO

O ciclo de desenvolvimento de detectores baseados na família YOLO (do YOLOv8 ao YOLOv12) integra a modelagem arquitetural, o ajuste fino de hiperparâmetros, a avaliação quantitativa com matrizes de confusão e curvas P-R, o dimensionamento de hardware e as rotinas automatizadas de backup com Rclone para proteger os checkpoints contra quedas de energia.

2. COLETA, ANOTAÇÃO E PRÉ-PROCESSAMENTO DE DADOS

A base de dados define o limite de desempenho do detector. Duas etapas distintas atuam sobre as imagens:

- **Coleta e curadoria de dados reais:** captação de novas imagens sob variações operacionais (ângulos, iluminação e oclusões). A coleta acrescenta padrões visuais inéditos à base.
- **Aumento de dados (*data augmentation*):** transformações geométricas e de cor no treino (rotações, cortes e *mosaic*) para ampliar a diversidade amostral sem novas coletas.

Para a anotação das caixas delimitadoras (*bounding boxes*) com coordenadas normalizadas em arquivos *.txt*, utilizam-se ambientes como:

- **CVAT (*Computer Vision Annotation Tool*):** ferramenta de código aberto para rotulagem de imagens e vídeos em projetos colaborativos.
- **Label Studio:** aplicação configurável para anotação multimodal com integração a modelos pré-treinados de apoio.
- **Roboflow:** serviço para versionamento de bases, padronização de resolução e exportação direta para a estrutura de pastas do YOLO.

3. IMPLANTAÇÃO PRÁTICA: INFERÊNCIA, VALIDAÇÃO E EXPORTAÇÃO

Após a convergência do treinamento, o modelo passa pela validação em partição de teste independente, pela execução de inferências e pela conversão para formatos de execução otimizada. O script a seguir executa esse fluxo com o pacote *ultralytics*:

Script: Validação, Inferência e Exportação de Modelos YOLO

```python
from ultralytics import YOLO

def main():
    # 1. Carregar o modelo treinado (best.pt)
    model = YOLO("runs/detect/treinamento_yolo/weights/best.pt")

    # 2. Validacao formal no conjunto de teste isolado
    print("--- Executando Validacao Formal ---")
    metrics = model.val(data="data.yaml", split="val")
    print(f"mAP50-95 obtido: {metrics.box.map:.4f}")
    print(f"mAP50 obtido: {metrics.box.map50:.4f}")

    # 3. Inferencia em novas imagens ou videos
    print("\n--- Executando Inferencia ---")
    results = model.predict(
        source="dataset/images/val",
        conf=0.25,
        save=True,
        project="runs/detect",
        name="inferencia_teste"
    )
    print("Inferencia concluida. Resultados salvos em runs/detect/inferencia_teste")

    # 4. Exportacao para formatos otimizados de producao
    print("\n--- Exportando Modelo para Producao ---")
    # Exportacao para ONNX (Open Neural Network Exchange)
    onnx_file = model.export(format="onnx", dynamic=True)
    print(f"Modelo exportado para ONNX: {onnx_file}")

    # Exportacao para TensorRT (requer GPU NVIDIA e TensorRT instalado)
    try:
        engine_file = model.export(format="engine", device=0)
        print(f"Modelo exportado para TensorRT: {engine_file}")
    except Exception as e:
        print(f"Aviso: Exportacao TensorRT nao executada ({e})")


if __name__ == "__main__":
    main()
```

4. IMPLANTAÇÃO EM AMBIENTES DE PRODUÇÃO (EDGE E CLOUD)

O formato ONNX permite carregar o modelo via *onnxruntime* em linguagens como C++, C# e Python, sem a necessidade de instalar o PyTorch no servidor ou dispositivo de destino. Para dispositivos de borda (*Edge Computing*), como placas NVIDIA Jetson, Raspberry Pi e processadores dedicados, aplicam-se motores específicos:

- **TensorRT (*.engine*):** compila o grafo com fusão de camadas e quantização em meia precisão (FP16) ou inteiros de 8 bits (INT8) para GPUs NVIDIA.
- **OpenVINO, TFLite e NPUs:** adaptam a rede para arquiteturas Intel x86, núcleos ARM e aceleradores neurais dedicados.

5. CONSIDERAÇÕES FINAIS

A implantação de detectores em condições reais é um processo iterativo. Resultados preliminares frequentemente acusam perdas em iluminação desfavorável, sobreposições de classes com morfologia semelhante ou falsos negativos em objetos de pequena escala. A correção dessas falhas envolve a leitura sistemática da matriz de confusão, o reforço amostral nos cenários de erro, a recalibração de hiperparâmetros de perda e o teste comparativo entre versões de diferentes capacidades da arquitetura.

REFERÊNCIAS

JOCHER, Glenn; CHAURASIA, Ayush; QIU, Jing. Ultralytics YOLO. [S.l.: s.n.], 2023. Software de detecção, segmentação, classificação e estimativa de pose baseado na família YOLO. Disponível em: https://github.com/ultralytics/ultralytics.

ONNX RUNTIME DEVELOPERS. ONNX Runtime: cross-platform, high performance ML inferencing and training accelerator. 2026. Disponível em: https://onnxruntime.ai.

ÍNDICE REMISSIVO

A

agente computacional 13, 17, 20
AI Winter 28
algoritmos 12, 14, 17, 19, 22, 25, 26, 29, 30, 45, 73
análise de dados 12
aprendizado 12, 14, 17, 22, 23, 24, 28, 29, 30, 33, 34, 39, 40, 50, 51, 52, 53, 63, 72, 76
aprendizado computacional 12, 14
Aprendizado de Máquina 12, 76
aprendizado supervisionado 12, 22, 24
arquiteturas de redes 26

B

background 47, 57, 58
backpropagation 28, 31, 40, 75
barreiras matemáticas 28
best.pt 54, 62, 63, 64
biblioteca Pandas 43

C

checkpoints 74
Ciência da Computação 7, 12, 76
classificação de imagens 31, 57
computadores 21, 28
conhecimento 12
conjuntos de dados 17, 22, 26, 29, 47, 73, 76
controle de fluxo 75
CUDA 30, 43, 44, 51, 53, 75, 77, 78
cuDNN 43

D

dados padronizados 29
datasets 26, 54
detectores de objetos 51, 57

E

endereçamento 28
energia 63, 77
engenharia 10, 29, 30, 74
erros de generalização 13
estações de trabalho 75

F

fluxo de trabalho 75
framework Ultralytics 57, 61

G

gráficos 43, 54, 61, 70, 73

I

ILSVRC 29, 30, 31
Inteligência Artificial (IA) 12
inteligência humana 12
Inverno da IA 28

J

Jupyter Notebook 43, 45

L

labels 13, 14, 45, 47, 62, 64, 67
last.pt 54, 62, 63, 64
leitura automatizada de códigos 28

M

mAP 27, 29, 37, 40, 51, 52, 53, 60, 62, 63, 70, 71, 72
Máquina de Vetor de Suporte 29
máquinas 12, 54
matriz de confusão 54, 57, 58, 59, 60, 73, 90
modelos analíticos 12

N

numerações de cheques 28

O

overfitting 13, 40, 51

P

pacote Ultralytics 43
pandas 43
pilares tecnológicos 12
Precisão 27, 29, 40, 59, 60, 62, 67, 68, 69, 70, 71
processadores 30, 75, 90
processamento 10, 19, 24, 25, 26, 28, 30, 34, 36, 37, 43, 44, 51, 53, 54, 60, 75, 76
processamento de imagem 26
PyTorch 35, 43, 44, 55, 62, 63, 90

R

redes neurais artificiais 28
results.csv 62, 72
retropropagação do erro 28, 75
Revocação 27, 59, 60, 62, 67, 68, 69, 70, 71
rotulação 22, 23, 26
rótulos 13, 14, 17, 19, 22, 31, 36, 45, 46, 48, 52, 64, 67

S

scripts locais 43
sistemas computacionais 12
software 10, 25, 27, 40, 43, 44, 45
soluções automatizadas 26
supervisor 13, 14, 17, 24

T

tarefas 13, 14, 15, 17, 22, 28, 29, 35, 37, 39
treinamento 10, 12, 13, 16, 17, 23, 24, 26, 28, 30, 33, 34, 35, 36, 38, 39, 40, 43, 47, 48, 49, 50, 51, 52, 53, 54, 56, 61, 62, 63, 64, 65, 66, 70, 72, 73, 74, 75, 77
treinamento de redes 28, 75, 77

U

ultralytics 41, 42, 43, 50, 55, 56, 90
Unidade Central de Processamento (CPU) 75
unidade de processamento 28

Y

YOLO 10, 11, 17, 23, 24, 25, 26, 27, 28, 31, 32, 34, 35, 36, 38, 39, 40, 41, 43, 44, 45, 46, 49, 50, 51, 53, 55, 57, 59, 61, 62, 63, 64, 65, 67, 71, 74, 75, 90

Z

ZIP codes 28



contato@editoraomnisscientia.com.br
https://editoraomnisscientia.com.br/
@editora_omnis_scientia
https://www.facebook.com/omnis.scientia.9
+55 87 99920-5762

EDITORA
OMNIS SCIENTIA

contato@editoraomnisscientia.com.br
https://editoraomnisscientia.com.br/
@editora_omnis_scientia
https://www.facebook.com/omnis.scientia.9
+55 87 99920-5762