APRENDIZADO DE MÁQUINA E COMPUTAÇÃO SIMBÓLICA NA CORREÇÃO DE CÁLCULOS MATEMÁTICOS

MACHINE LEARNING AND SYMBOLIC COMPUTATION FOR THE CORRECTION OF MATHEMATICAL CALCULATIONS

Lucas Gabriel de Sousa e Silva*
Felipe Bastos Nunes**

RESUMO

Este trabalho propõe e avalia uma pipeline de visão computacional para a verificação de cálculos matemáticos simples em imagens capturadas por câmera. A pesquisa tem como finalidade desenvolver um sistema automatizado capaz de identificar se uma expressão matemática manuscrita está correta ou incorreta, contribuindo para o processo de correção no ambiente educacional. Para alcançar esse objetivo, a metodologia realiza uma análise comparativa entre três arquiteturas da familia YOLO (YOLOv8, YOLOv9 e YOLOv11), que são treinadas em um *dataset* customizado. Estas arquiteturas, sendo redes neurais convolucionais (CNNs), atuam como o motor de reconhecimento óptico de caracteres (OCR) do sistema, detectando e classificando cada dígito e operador. Em seguida, um algoritmo de reconstrução estrutural processa as detecções do YOLO, ordenando-as para formar uma string de equação. Por fim, esta string é validada pela biblioteca de computação simbólica SymPy, que avalia a expressão e determina se o cálculo está matematicamente correto. A avaliação compara o desempenho de cada modelo em métricas como precisão, revocação (*recall*) e *Intersection over Union* (IoU), enquanto a eficácia do sistema final é medida pela sua acurácia na classificação dos cálculos. A abordagem mostrou-se promissora, identificando a YOLOv11s como o modelo que oferece o melhor equilíbrio entre precisão e velocidade para correção automatizada de cálculos simples.

Palavras-chave: YOLO. SymPy. Visão computacional. aprendizado de máquina. Cálculo matemático. Correção automatizada.

ABSTRACT

This work proposes and evaluates a computer vision pipeline for the verification of simple mathematical calculations in images captured by camera. The research aims to develop an auto-

* Instituto Federal de Educação, Ciência e Tecnologia do Ceará, Aracati. CE, Brasil. E-mail: lucas.gabriel01@aluno.ifce.edu.br
** Especialista em Docência no Ensino Técnico, docente do Instituto Federal de Educação, Ciência e Tecnologia do Ceará, Aracati, CE, Brasil. E-mail: felipebastos@ifce.edu.br

mated system capable of identifying whether a handwritten mathematical expression is correct or incorrect, contributing to the correction process in the educational environment. To achieve this objective, the methodology performs a comparative analysis between three recent architectures from the YOLO family (YOLOv8, YOLOv9, and YOLOv11), which are trained on a customized dataset. These architectures, being Convolutional Neural Networks (CNNs), act as the system's Optical Character Recognition (OCR) engine, detecting and classifying each digit and operator. Next, a structural reconstruction algorithm processes the YOLO detections, ordering them to form an equation string. Finally, this string is validated by the SymPy symbolic computation library, which evaluates the expression and determines if the calculation is mathematically correct. The evaluation compares the performance of each model on metrics such as precision, recall, and *Intersection over Union* (IoU), while the final system's effectiveness is measured by its accuracy in classifying the calculations. The approach proved promising, identifying the **YOLOv11s** as the model that offers the best balance between precision and speed for the automated correction of simple calculations.

Keywords: YOLO. Computer Vision. Machine Learning. Mathematical Calculation. Automated Correction

1 INTRODUÇÃO

A matemática é uma disciplina fundamental no desenvolvimento cognitivo dos estudantes, proporcionando diversas áreas do conhecimento e do cotidiano. Entretanto, sua aprendizagem frequentemente é definida por desafios e erros que, se mal compreendidos, podem gerar frustrações e desinteresse nos alunos (ESCOBAR, 2016). A condição atual da educação na matemática revela uma preocupação crescente com a análise desses erros, não com meras falhas a serem corrigidas, mas como oportunidades valiosas para entender os processos cognitivos e as dificuldades enfrentadas pelos estudantes (ESCOBAR, 2016).

Dentro desse espectro geral, a pesquisa em educação na matemática tem buscado investigar os fatores que interferem na aprendizagem, desde questões metodológicas até aspectos psicológicos e sociais. O estudo dos erros ganha destaque ao evidenciar como os alunos interpretam os conceitos matemáticos, muitas vezes recorrendo a respostas intuitivas e automáticas, como foi descrito por Kahneman (2012), em sua teoria sobre os sistemas 1 e 2 de pensamento.

Nesse contexto, com o avanço crescente das tecnologias de visão computacional e inteligência artificial, agora é possível aplicar modelos de detecção de objetos em contextos cada vez mais complexos e específicos e pode ser uma aliada para identificar erros matemáticos de forma eficiente. Entre esses avanços, a arquitetura YOLO se destaca pela sua forma de conseguir identificar e localizar objetos em imagens de forma rápida e precisa, utilizando uma arquitetura end-to-end [1] (REDMON et al., 2016). Dessa forma, abrem-se portas para a aplicação

[1] É um termo em inglês que significa cadeia de ponta a ponta que auxilia no processo de produção. Ele é como

desses modelos em áreas educacionais, principalmente na identificação de erros em cálculos matemáticos, sejam eles manuscritos ou digitalizados.

A aplicação de *machine learning* tem um papel muito importante nesta direção, permitindo que sistemas aprendam a partir de dados e melhorem seu desempenho continuamente (ALPAYDIN, 2020). As redes neurais convolucionais (CNNs), por exemplo, são essenciais para processar imagens de cálculos matemáticos, identificando padrões e símbolos com alta precisão (GOODFELLOW; BENGIO; COURVILLE, 2016). Em seu trabalho, Wang e Liu (2021) demonstram a eficácia de modelos *encoder-decoder* na tradução de imagens de fórmulas matemáticas para sequências LaTeX, superando ferramentas tradicionais como o Mathpix.

Em um ambiente escolar e acadêmico, detectar erros em cálculos matemáticos é uma tarefa bem comum, mas que muitas vezes consome boa parte do tempo e às vezes está sujeita a erros humanos. Ter uma correção automatizada, por meio de métodos computacionais, pode ser uma solução que ajudaria professores, avaliadores e até mesmo softwares educacionais. Mas como a diversidade na forma como as expressões matemáticas são escritas e a dificuldade de interpretar contextos visuais que são complexos, torna esse problema bastante complexo de ser executado.

Este trabalho tem como objetivo geral desenvolver e avaliar uma pipeline de visão computacional para a verificação de cálculos matemáticos simples, baseada em imagens de câmera de celular. O sistema utiliza a arquitetura YOLO, que atua como o motor de Reconhecimento Óptico de Caracteres (OCR) do projeto, para realizar a detecção e classificação de cada caractere manuscrito. na sequencia, um algoritmo de reconstrução estrutural processa as detecções do YOLO, ordenando-as para formar uma *string* de equação. Por fim, essa *string* é analisada por um motor de validação lógica, implementado com a biblioteca de computação simbólica *SymPy*, que determina se o cálculo está matematicamente correto ou incorreto.

A hipótese central é que o YOLO, quando treinado em um *dataset* focado, é capaz de fornecer os dados de percepção (caracteres e suas posições) com precisão suficiente para que o *SymPy* possa avaliar e, por fim, o sistema possa apresentar o resultado de forma visual, demarcando a equação na imagem como 'Correta' ou 'Incorreta', automatizando, assim, o processo de correção.

O trabalho está organizado da seguinte maneira: a Seção 2 apresenta o referencial teórico, abordando os conceitos fundamentais de visão computacional, *machine learning* e as ferramentas utilizadas (YOLO e *SymPy*). A Seção 3 discute os trabalhos relacionados, destacando as abordagens existentes na literatura. A Seção 4 detalha a metodologia, descrevendo a criação do *dataset* customizado, a configuração dos experimentos e a implementação da *pipeline* de validação. Por fim, a Seção 5 apresenta a análise comparativa dos resultados obtidos no treinamento das arquiteturas e a validação visual do sistema proposto.

---

um sistema que pode ser monitorado desde a escolha da matéria-prima até a entrega ao cliente final, envolvendo as etapas de produção, de transporte e também de pós-venda.

2 REFERENCIAL TEÓRICO

Deve-se compreender que a busca por uma ferramenta com reconhecimento de escrita e interpretação desta perpassa algumas áreas de estudo muito discutidas, como identificação de padrões, aprendizado de máquina e inteligência artificial. Neste sentido, este capítulo apresenta uma ideia geral destas áreas e algumas de suas vertentes.

2.1 Ferramenta YOLO

Como foi descrito por (REDMON et al., 2016) a arquitetura YOLO é capaz de identificar e localizar diferentes objetos em uma única imagem com alta rapidez, utilizando exclusivamente os dados extraídos diretamente dos *pixels*. Seu funcionamento é direto, pois adota uma estrutura do tipo *end-to-end*, facilitando a integração e execução. O modelo gera múltiplas caixas delimitadoras e, para cada uma delas, é calculado um grau de confiança, indicando a chance (em um intervalo entre 0 e 1) de haver um objeto naquele local. Além disso, é feita uma estimativa da categoria correspondente ao objeto identificado. A combinação entre essa confiança e a classificação prevista resulta em uma pontuação final que representa a chance daquela área realmente conter o objeto esperado (PIEMONTEZ, 2023).

Em 2016 (um ano depois do lançamento do YOLO) surgiu o YOLOv2 que melhorou o modelo original ao incorporar a normalização de lotes, caixas de ancoragem e *clusters* de dimensão. Segundo Cimirro (2022), com o YOLOv2 se tem um detector mais preciso e rápido. Em vez de ele aumentar a rede, ela foi simplificada, tornando sua representação mais fácil de aprender. Após mais dois anos, em 2018, foi lançado o YOLOv3 que melhorou ainda mais o desempenho do modelo, utilizando uma rede de base mais eficiente, várias âncoras e agrupamento de pirâmides espaciais. A YOLOv3 é um detector bastante eficiente, especialmente na geração de caixas delimitadoras para objetos. No entanto, seu desempenho diminui conforme o limite de *Intersection over Union*(IoU) aumenta, o que indica uma certa dificuldade em alinhar com precisão as caixas aos objetos (REDMON; FARHADI, 2018).

Foram necessárias várias versões do YOLO até chegar à atual, que é a YOLO11, a qual contém diversas melhorias, com um desempenho aprimorado para detecção de objetos, segmentação, estimativa de pose, rastreamento e classificação, tornando-se uma escolha versátil para uma ampla gama de tarefas de visão computacional. Na Figura 1, é possível ver o comparativo com as outras versões do YOLO e como a versão 11 do YOLO é melhor em comparação com algumas versões anteriores (ULTRALYTICS, 2024).

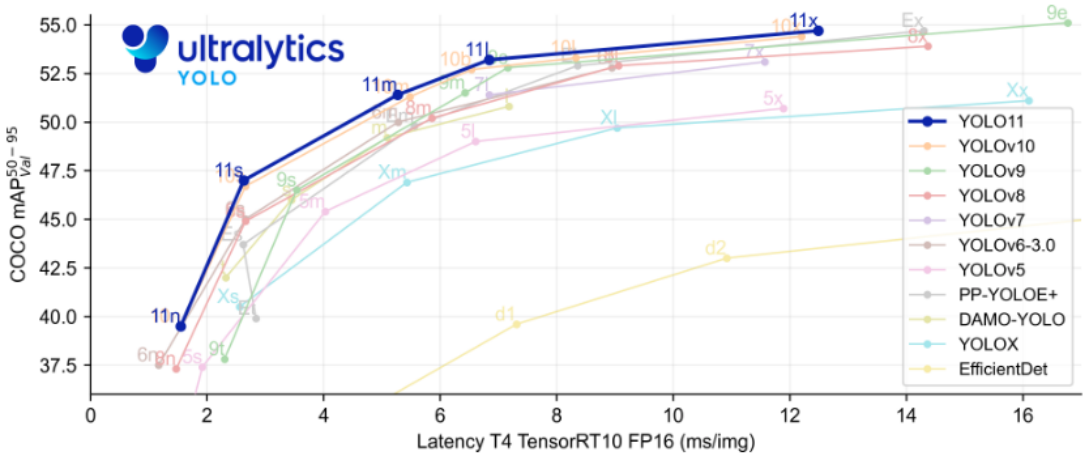


Figura 1 – Comparativo da YOLO11 com as versões anteriores. Fonte: (ULTRALYTICS, 2024)

A comparação é feita a partir de duas métricas principais, o eixo Y e o eixo X. No eixo y é onde mostra o mAP que significa *mean Average Precision* (média de precisão média) calculada sobre o conjunto de validação do *dataset* COCO. O 50-95 é a métrica que é medida considerando *IoU thresholds* (limiares de sobreposição entre caixas) que vão de 0.50 até 0.95, com passo de 0.05, e quanto maior o valor que o modelo tem melhor desempenho em identificar corretamente objetos com precisão. Já o eixo X mede o tempo médio que cada modelo leva para processar uma imagem. O *T4 TensorRT10* especifica o ambiente de medição usando uma GPU NVIDIA que é comum em inferência para alto desempenho, e quanto menor for o tempo isso significa que o modelo é mais rápido na inferência.

2.2 Utilização do SymPy

Na engenharia de software e na ciência da computação, a manipulação de expressões matemáticas é dividida em duas grandes abordagens: a computação numérica e a computação simbólica. A computação numérica, utilizada por bibliotecas como o *NumPy*, foca na avaliação de expressões em valores de ponto flutuante. É otimizada para velocidade e cálculos aproximados em grande escala. Já a computação simbólica, também conhecida como Sistema de Álgebra Computacional (CAS - *Computer Algebra System*), trata as expressões matemáticas de forma conceitual e exata. O objetivo não é apenas calcular um número, mas sim entender e manipular a própria estrutura da fórmula (MEURER et al., 2017).

Com isso, o *SymPy* é uma biblioteca de código aberto para a linguagem Python, projetada especificamente para computação simbólica. Algumas de suas principais características se destacam na manipulação de símbolos, permitindo que variáveis sejam tratadas como símbolos abstratos e não como contêineres para valores numéricos. Isso possibilita a execução de operações algébricas complexas, como simplificar a expressão $x \cdot x$ para x^2 ou representar $\sqrt{8}$ de forma exata como $2\sqrt{2}$. Além disso, o *SymPy* oferece uma análise robusta de expressões, com ferramentas para converter strings de texto (como 2 + 3 * 2) em objetos matemáticos compreensíveis. A biblioteca também garante uma avaliação precisa ao respeitar rigorosamente

a ordem padrão de operações, o que assegura que (2 + 3 * 2) seja corretamente avaliado como 8 e não sequencialmente como 10 (MEURER et al., 2017).

Devido a essa capacidade de interpretar e avaliar logicamente a estrutura de expressão matemática de forma segura e precisa, o *SymPy* se estabelece como uma ferramenta fundamental para a validação de cálculos em sistemas de software.

2.3 Machine Learning

Segundo Alpaydin (2020) aprendizado de máquina (*machine learning*) procura otimizar um critério de desempenho usando dados de exemplo ou experiências anteriores. Suas áreas de aplicação são abundantes: no varejo e no setor financeiro, dados históricos são analisados para construir modelos com aplicação em crédito, detecção de fraudes ou mercado de ações (ALPAYDIN, 2020). Mas o aprendizado de máquina não é apenas um problema de banco de dados, ele também faz parte da inteligência artificial. Para ser capaz de aplicar inteligência, um sistema que está em um ambiente em mudança deve ter a capacidade de aprender. A *machine learning* usa a teoria da estatística, técnicas de otimização e outras ferramentas matemáticas na construção de modelos, uma vez que a tarefa principal é fazer inferências a partir de uma amostra.

A aplicação de métodos da *machine learning* a grandes bases de dados é chamada de *Data Mining* (Mineração de dados) (ALPAYDIN, 2020). A analogia é que um grande volume de dados brutos é extraído, como em uma mina, e quando processado, leva a uma pequena quantidade de material muito valioso. De forma semelhante, no *data mining*, um grande volume de dados é processado para construir um modelo simples com valor útil, por exemplo, com uma alta precisão preditiva (ALPAYDIN, 2020). A *machine learning*, contudo, transcende o simples processamento de bases de dados, consolidando-se como o pilar da inteligência artificial que permite ao sistema aprender e se adaptar a novos cenários de forma autônoma. Para ser inteligente, um sistema deve ter um ambiente em que seja capaz de aprender. Se o sistema puder se adaptar a mudanças, o projetista não precisa prever todos os cenários e fornecer soluções para todos os possíveis problemas (ALPAYDIN, 2020).

A *machine learning* também nos ajuda a encontrar soluções para muitos problemas em visão, reconhecimento de fala e robótica. É possível, por exemplo, reconhecer rostos. Tarefa esta realizada com tanta facilidade todos os dias quando reconhecemos membros da família e amigos ao olhar para seus rostos ou fotografias, apesar de diferenças sutis de postura, iluminação e cabelo (ALPAYDIN, 2020). Mas fazemos isso de forma inconsciente e não conseguimos explicar de forma breve e instantânea como fazemos isso (ALPAYDIN, 2020). Do ponto de vista computacional, um rosto não é apenas uma coleção aleatória de pixels. Um rosto tem estrutura, é simétrico, existem os olhos, o nariz, a boca, localizados em certas posições e proporções relativas. Ao analisar imagens faciais de uma pessoa, um programa de aprendizado pode capturar o padrão específico que a representa e, então, reconhecer esse padrão ao verificar outras imagens.

2.4 Redes Neurais Convolucionais - CNN

As CNNs, a arquitetura proposta originalmente por Yann LeCun Lecun et al. (1998) constituem uma categoria específica de redes neurais projetadas para lidar com dados que possuem uma estrutura topológica bem definida, como é o caso das imagens, organizadas em grades bidimensionais. A operação de convolução, que dá nome a essas redes, é uma forma particular de operação linear que substitui a multiplicação matricial tradicional em ao menos uma de suas camadas. Quando aplicadas ao processamento de imagens, essas redes utilizam filtros (ou *kernels*), representados por pequenas matrizes, que percorrem a imagem de entrada. A operação consiste em multiplicar os valores dos *pixels* da imagem pelos valores correspondentes do filtro, e a soma dos resultados é registrada em um mapa de características, o que permite à rede extrair padrões relevantes da imagem (GOODFELLOW; BENGIO; COURVILLE, 2016).

2.5 Roboflow

O Roboflow[2] é uma plataforma avançada de inteligência artificial (IA) especializada no desenvolvimento de modelos para tarefas de visão computacional. Com o objetivo de simplificar o processo de criação, treinamento e implementação de imagens, detecção de objetos e segmentação de imagens, a plataforma atende tanto iniciantes quanto especialistas na área de *machine learning*. Uma das principais vantagens do *Roboflow* é sua capacidade de facilitar o processamento e a anotação de imagens. Os usuários podem facilmente carregar imagens para a plataforma e prepará-las para o treinamento de modelos.

2.6 MNIST

O *dataset* MNIST (Modified National Institute of Standards and Technology) é um dos conjuntos de dados mais icônicos e fundamentais na área de aprendizado de máquina e visão computacional. Ele consiste em uma vasta coleção de dígitos manuscritos, de 0 a 9, divididos em um conjunto de 60.000 imagens para treinamento e 10.000 para teste (LECUN; CORTES; BURGES, 2010). Cada imagem é normalizada em escala de cinza e centralizada em um formato de 28 x 28 pixels.

2.7 Transferência de Aprendizado (*Transfer Learning*)

A transferência de aprendizado, ou *transfer learning*, é uma tecnica de aprendizado de máquina onde um modelo desenvolvido para uma tarefa e reutilizado como ponto de partida para um modelo em uma segunda tarefa. Segundo Goodfellow, Bengio e Courville (2016), essa abordagem é crucial em cenários de *deep learning*, pois permite que a generalização aprendida em uma configuração (como um grande banco de dados de imagens genéricas) seja transferida para outra, melhorando o desempenho da aprendizagem da nova tarefa.

[2] https://roboflow.com/

Na prática, em redes neurais convolucionais, as camadas iniciais aprendem a detectar características visuais de baixo nível, como bordas e curvas, que são à maioria as imagens. Ao utilizar pesos pré-treinados de grandes *datasets* (como o COCO ou ImageNet), aproveita-se essa estratégia é particularmente benéfica quando o conjunto de dados da tarefa de destino é pequeno, pois evita o *overfitting* e acelera significativamente a convergência do treinamento, dispensando a necessidade de aprender todos os parâmetros do zero.

3 TRABALHOS RELACIONADOS

Nesta seção, são apresentados os trabalhos relacionados que fundamentam o desenvolvimento desta pesquisa. A revisão bibliográfica concentra-se em abordagens recentes de visão computacional e aprendizado profundo (*deep learning*) voltadas para o reconhecimento de expressões matemáticas manuscritas e impressas. Serão discutidos estudos que exploram desde a detecção de objetos, utilizando arquiteturas como YOLO e R-CNN, até a tradução estrutural de imagens para sequências lógicas, contextualizando as técnicas e as lacunas que motivaram a solução proposta neste trabalho

(ROSA et al., 2023) propõe uma solução para o reconhecimento de expressões matemáticas manuscritas, com o foco principal em operações de adição e subtração dispostas verticalmente, que é um formato mais comum no ensino fundamental, mas que pouco é explorado na literatura científica. As principais contribuições do artigo para resolver esse problema foi a criação de um novo conjunto de dados, contendo 300 imagens de expressões de adição e subtração vertical, escritas à mão por diferentes pessoas. Além disso, eles ampliaram o conjunto de dados MNIST[3] (Modified National Institute of Standards and Technology database) para gerar 2.600 imagens artificiais com a mesma estrutura vertical, visando melhorar o treinamento dos modelos.

A solução divide o problema para reconhecer as expressões em duas etapas, a primeira etapa é a de detecção de objetos e a segunda etapa de transcrição. Na etapa de detecção, o objetivo foi identificar a localização dos símbolos matemáticos e obter as caixas delimitadoras correspondentes. Em seguida, realizaram uma etapa de pós-processamento para melhorar os resultados da detecção. Depois de obterem a posição de cada símbolo matemático na imagem, usam um transcritor que recebe como entrada a caixa delimitadora e as classes e produz uma expressão resultante em LATEX. A Figura 2 mostra a solução HMER(Reconhecimento de expressões matemáticas manuscritas) proposta. Ainda na fase de detecção de objetos, aplicaram diferentes métodos baseados em redes neurais convolucionais. Foram avaliados os modelos *YOLOv7*, *YOLOv8*, *YOLO-NAS*, FCOS(Fully Convolutional One-Stage Object Detection) e *NanoDet* para identificar e localizar cada símbolo matemático na imagem.

Na etapa de transcrição, as caixas delimitadoras e as classes correspondentes são recebidas como entrada e produzem uma expressão LATEX relacionada à equação matemática na imagem.

[3] MNIST é um famoso conjunto de dados de dígitos manuscritos (0 a 9), amplamente utilizado para treinar e testar modelos de aprendizado de máquina e reconhecimento de imagem.

Algumas abordagens tradicionais para HMER resolvem esse problema criando uma gramática ou usando uma estratégia gráfica. Já neste trabalho, esta etapa foi simplificada, propondo uma transcrição baseada na estrutura posicional das caixas delimitadoras. As detecções são divididas em quatro categorias: número, operação (op), sinal de igual e transporte, onde o número se refere aos dígitos de 0 a 9, operação +,-, o sinal de igual refere-se a linha de igualdade na expressão e o transporte refere-se aos pequenos dígitos relacionados ao símbolo de transporte e espera-se que a expressão esteja no formato "A op B = R", onde A e B são os operandos e R é o número resultante, como é mostrado na Figura 3.

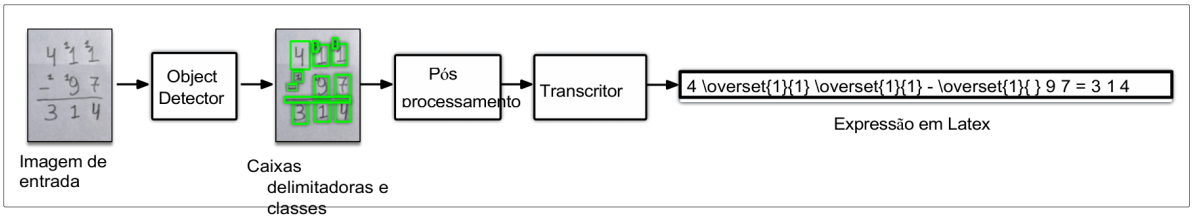


Figura 2 – Fluxograma da solução proposta para reconhecimento de adição e subtração de colunas. Fonte: (ROSA et al., 2023)

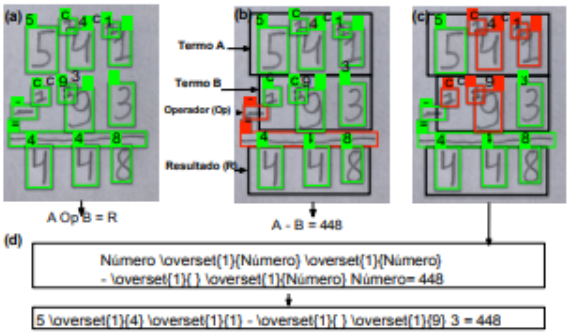


Figura 3 – Fase de transcrição. (a) caixas delimitadoras e previsões da detecção de objetos; (b) identificação do operador *Op*, simbolo de igual e números do resultado. Os resultados são obtidos a partir do número abaixo da caixa delimitadora do sinal de igual; (c) identificação dos símbolos de transporte e numero associado, e (d) construção da expressão com base nas caixas delimitadoras identificadas. Fonte: (ROSA et al., 2023)

Kacem (2024) propõe um sistema baseado em *deep learning* para detecção, reconhecimento e análise estrutural de fórmulas matemáticas presentes em documentos científicos (como PDFs digitalizados). Para construir o sistema, foi utilizado o modelo YOLOv8, ajustado com um *dataset* específico IBEM (*Image-Based Equation Metadata*), que se destaca por fornecer anotações detalhadas de fórmulas em contextos de documentos reais, permitindo ao modelo distinguir com precisão fórmulas embutidas (dentro do texto) e isoladas (fora do corpo do texto). Também foi realizado o reconhecimento de símbolos matemáticos com o uso do modelo *Faster R-CNN* para detectar e classificar os símbolos matemáticos individualmente dentro das fórmulas. Foi feita uma análise do layout da fórmula que emprega uma *GCN* (*Graph Convolutional Neural Networks*) para representar a fórmula como um grafo, onde símbolos são nós e relações espa-

ciais são arestas, e para analisar a estrutura sintática e espacial da fórmula (como sobrescritos, subscritos, frações etc).

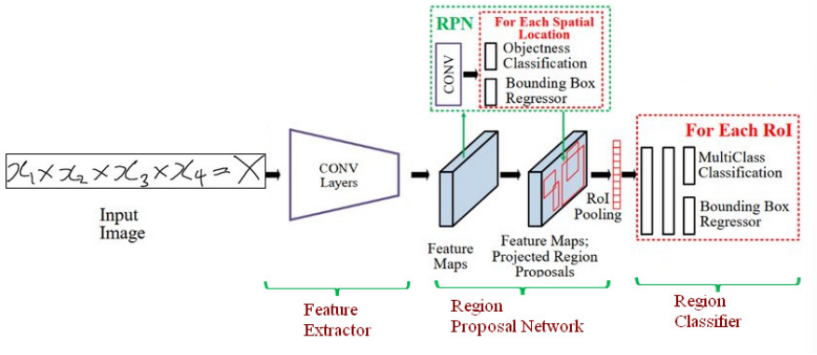


Figura 4 – Arquitetura do Faster R-CNN. Fonte: Kacem (2024)

Nesse sistema foi usado o *Region Proposal Network* que é uma rede neural totalmente convolucional de duas camadas. Uma camada vai ser para regressão de caixa e a outra é a camada de classificação. Ela começa executando a imagem pela CNN para gerar um mapa de características, então o algoritmo cria caixas de ancoragem representando cada classe de objeto com altura e largura específicas com base nos objetos no conjunto de dados de treinamento.

Wang e Liu (2021) propõe um modelo de rede neural profunda com uma arquitetura *encoder-decoder* que traduz imagens de fórmulas matemáticas para as suas sequências de marcação *LaTeX*. O codificador é uma rede neural convolucional que transforma imagens em um grupo de mapas de características. Para melhor captar as relações espaciais dos símbolos matemáticos, os mapas de características são aumentados com codificação posicional 2D antes de serem desdobrados em um vetor. Foi feita uma análise do software *Mathpix* executando manualmente 100 imagens do conjunto de teste do *IM2LATEX-100 K*. A Tabela 1 mostra os resultados detalhados dos experimentos dos dois sistemas. O *MI2LaTeX* obtém pontuações significativamente mais altas do que o *Mathpix* em todas as métricas de avaliação. De acordo com a observação que fizeram, o *Mathpix* é altamente preciso, mas se torna inconsistente.

As métricas apresentadas na Tabela 1 avaliam a eficácia da conversão de imagens para LaTeX sob diferentes perspectivas técnicas. O BLEU mede a similaridade estatística por meio da sobreposição de n-gramas, enquanto a *Image edit distance* quantifica o esforço necessário para a correção estrutural da sequência gerada. O *Exact Match* indica a correspondência total entre a predição e o gabarito, com a variação (-ws) ignorando espaços em branco irrelevantes para a sintaxe matemática. Por fim, a métrica IMEG valida a consistência visual e semântica da fórmula reconstruída em relação à imagem original.

Tabela 1 – Comparação entre Mathpix e MI2LaTeX

Métrica	**Mathpix**	**MI2LaTeX**
BLEU	80.64	92.08
Image edit distance	76.19	93.38
Exact match	8.00	82.00
Exact match (-ws)	34.00	84.00
IMEG	83.19	97.23

Fonte: Wang e Liu (2021)

Os resultados do desempenho mostrados na Tabela 2 demonstram o desempenho do modelo sem e com o treinamento de reforço que o autor criou. Todos os quatro modelos de aprendizagem profunda obtiveram um desempenho significativamente melhor do que o sistema *InftyReader*. Dentre os diferentes modelos de *deep learning* avaliados, destacaram-se aqueles baseados nas arquiteturas da família YOLO, que apresentaram desempenho superior em relação ao modelo de linha de base, representado por uma abordagem tradicional de detecção sem mecanismos de atenção. Esse ganho de desempenho é atribuído à utilização de redes neurais convolucionais mais profundas e à introdução de módulos de atenção, que permitem ao modelo focar em regiões relevantes da imagem durante o processo de detecção.

Tabela 2 – Avaliação de desempenho de diferentes modelos

Model	**BLEU**	**Image edit distance**	**Exact match**	**Exact match (-ws)**	**IMEG**
INFTY	66.65	53.82	15.60	26.66	–
WYGIWYS	87.73	87.60	77.46	79.88	90.26
Double attention	88.42	88.57	79.81	–	–
DenseNet	88.25	91.57	–	–	–
Ml2LaTeX w/o reinforce	89.08	91.09	79.39	82.13	95.41
Ml2LaTeX with reinforce	**90.28**	**92.28**	**82.33**	**84.79**	**96.15**

Fonte: Wang e Liu (2021)

Por fim, para sintetizar as abordagens analisadas e destacar as lacunas preenchidas por esta pesquisa, foi elaborado um quadro comparativo. A Tabela 3 evidencia que, embora o uso de Redes Neurais Convolucionais (CNN) e da arquitetura YOLO seja recorrente na literatura para a detecção de objetos, a proposta deste trabalho se diferencia pela integração desses métodos com a biblioteca *SymPy*.

Tabela 3 – Comparação entre os trabalhos relacionados e o trabalho proposto

Funcionalidade	**(ROSA et al., 2023)**	**(KACEM, 2024)**	**(WANG; LIU, 2021)**	**Presente Trabalho**
Uso da arquitetura YOLO	X	X		X
Utilização da CNN	X	X	X	X
Utilização do SymPy				X
Detecção em cálculos manuscritos	X			X
Identificação de erros matemáticos				X

Fonte: Autor

Essa combinação permite não apenas o reconhecimento dos caracteres manuscritos, mas também a validação lógica automatizada da equação matemática, uma funcionalidade não abordada centralmente nos trabalhos correlatos analisados.

4 METODOLOGIA

Nesta seção, são apresentados os procedimentos metodológicos adotados para o desenvolvimento do trabalho, cujo o objetivo é investigar a aplicação da arquitetura YOLO na identificação de erros em cálculos matemáticos a partir de imagens.

Para isso, foi desenvolvida e testada uma *pipeline* completa de verificação. Como motor principal de percepção visual, foram treinadas e avaliadas múltiplas arquiteturas da família YOLO (incluindo as versões v8, v9 e v11) a fim de identificar o modelo com melhor desempenho para esta tarefa especifica. Estas arquiteturas, que são em si poderosas Redes Neurais Convolucionais (CNNs), atuam como o sistema de OCR da pesquisa, identificando e localizando os elementos da expressão. Na sequência, um algoritmo de reconstrução lógica e um motor de validação simbólica são empregados para analisar e verificar a correção dos cálculos matemáticos.

4.1 Coleta e Preparação dos Dados

Inicialmente, considerou-se a utilização de bases de dados públicas para agilizar o processo. Após uma revisão bibliográfica em repositórios como o *Kaggle*, foram identificados dois conjuntos de dados: um elaborado por Clarence Zhao[4], contendo mais de 8500 imagens de expressões matemáticas, e outro por Govindaram Sriram[5], composto por 60 imagens. Entretanto, análises preliminares indicaram que o primeiro apresentava inconsistências na detecção de operadores e o segundo carecia de cálculos completos sem resultados, inviabilizando seu uso direto para a validação de erros. Posteriormente, a partir do apresentado no trabalho de Rosa et al. (2023), constatou-se que é utilizado o *dataset* MNIST para treinar os números de 0 a 9.

Diante das limitações das bases públicas, optou-se pela construção de um *dataset* customizado híbrido. Este conjunto foi projetado especificamente para suprir a carência de operadores aritméticos e fornecer exemplos de cálculos com erros propositais, fundamentais para a lógica de validação do sistema. Foram produzidos manualmente cerca de 135 cálculos matemáticos variados (incluindo operações de adição, subtração, multiplicação e divisão, com e sem erros propositais) distribuídos em 6 folhas A4 (Figura 5).

Posteriormente, utilizando ferramentas de edição de imagem, as folhas foram segmentadas digitalmente para que cada imagem contivesse apenas um cálculo isolado (Figura 6). Por fim, essas imagens geradas manualmente foram integradas aos dados de Clarence Zhao e Govindaram Sriram, compondo a base de dados unificada e definitiva utilizada para o treinamento

[4] Disponível em: <https://www.kaggle.com/datasets/clarencezhao/handwritten-math-symbol-dataset>

[5] Disponível em: <https://www.kaggle.com/datasets/govindaramsriram/handwritten-math-expressions-dataset?utm_source=chatgpt.com>

dos modelos. O *dataset* final ficou composto por **18.095** imagens, divididas em 70% para treino, 20% para validação e 10% para testes.

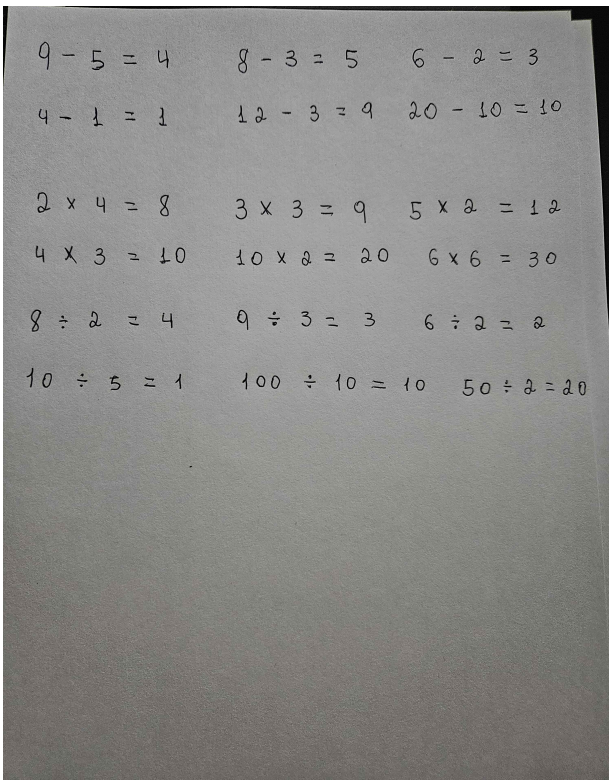


Figura 5 – Imagem de cálculos na folha A4. Fonte: Autor

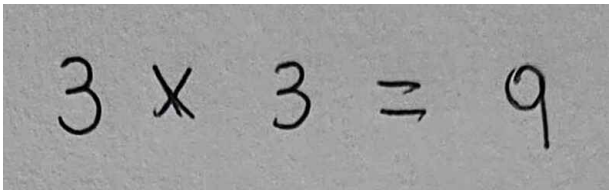


Figura 6 – Imagem de cálculos recortados na folha A4. Fonte: Autor

As imagens resultantes foram rotuladas manualmente com *bounding boxes*[6] utilizando a plataforma *RoboFlow*, gerando os arquivos de anotação no formato padrão YOLO (Figura 7). Cada região demarcada corresponde a um dígito ou operador matemático essencial para a interpretação da fórmula.

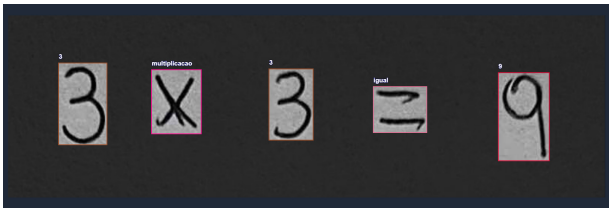


Figura 7 – Imagem do cálculo demarcado pelo RoboFlow. Fonte: Autor

[6] Bounding box é uma forma geométrica retangular que envolve um objeto em uma imagem digital, definindo sua localização e dimensão para facilitar a análise computacional

4.2 Treinamento do Modelo YOLO

Com o ambiente de desenvolvimento configurado, foram realizados testes iniciais para validar a viabilidade do treinamento. Primeiramente, utilizou-se os *datasets* públicos disponibilizados por Clarence Zhao e Govindaram Sriram, treinando um modelo base por apenas 50 épocas com a YOLOv11 para observar o comportamento inicial da curva de aprendizado. Testes de inferência revelaram um desempenho insatisfatório (*underfitting*), onde o modelo confundia classes visualmente semelhantes, por exemplo, o número '2' com '6' e o símbolo de igualdade ('=') com o de divisão ('÷'). Mesmo estendendo o treinamento para 100 épocas, a precisão permaneceu baixa.

Buscando contornar a dificuldade na detecção dos números, realizou-se uma tentativa utilizando o *dataset* MNIST, amplamente reconhecido por seu vasto volume de dígitos manuscritos. No entanto, os experimentos demonstraram que essa abordagem foi insuficiente para o contexto deste trabalho. O MNIST, por conter apenas dígitos isolados e pré-processados, não ofereceu a capacidade de generalização necessária para detectar equações completas em imagens naturais, além de não suprir a lacuna crítica da identificação dos operadores matemáticos (soma, subtração, etc).

Diante dessas limitações cumulativas, optou-se pela criação e refinamento de um *dataset* próprio e customizado. Visando contribuir com a comunidade científica e fomentar trabalhos futuros na área, este conjunto de dados foi disponibilizado publicamente em repositório aberto[7]. Com a base de dados consolidada, foi estabelecido um protocolo experimental para realizar uma análise comparativa de desempenho. Para isso, foram selecionadas três arquiteturas distintas da família YOLO: YOLOv8s, YOLOv9s e YOLOv11s.

A escolha pelas versões *small* (s) em todos os modelos visou padronizar a complexidade das redes neurais, permitindo uma comparação justa entre precisão e custo computacional. Os resultados iniciais demonstraram que o uso do novo *dataset* eliminou as confusões críticas de classes observadas anteriormente.

4.2.1 Configuração Experimental e Hiperparâmetros

Para garantir a igualdade na comparação entre as arquiteturas, foi estabelecido um protocolo rígido onde todos os modelos foram submetidos às mesmas condições. O ambiente de desenvolvimento utilizado foi o Kaggle Notebooks, utilizando aceleração de hardware via *GPU NVIDIA Tesla T4 x2*.

Os hiperparâmetros foram definidos para equilibrar o tempo de convergência e a capacidade de generalização. A configuração final adotada para todos os experimentos foi:

- **Épocas:** 100. Este valor foi estabelecido como um limite superior para garantir a convergência completa do modelo durante o processo de *transfer learning*, permitindo a estabilização das curvas de perda sem estender desnecessariamente o treinamento;

[7] Disponível em: <https://www.kaggle.com/datasets/lukeeg/handwritten-math-equations-for-yolo/data>

- **Batch Size:** 16. O tamanho do lote foi ajustado para maximizar a ocupação da memória VRAM da GPU NVIDIA Tesla T4 (disponível no ambiente Kaggle), garantindo a estabilidade do gradiente estocástico sem exceder os recursos de hardware;
- **Patience:** 50 épocas. Utilizado no mecanismo de *Early Stopping*, este valor permite que o treinamento seja interrompido prematuramente caso o modelo pare de apresentar melhorias na validação, prevenindo o *overfitting* (sobreajuste);
- **Otimizador:** AdamW. Selecionado por sua implementação desacoplada de decaimento de pesos (*weight decay*), que oferece uma generalização superior em redes profundas se comparado ao otimizador Adam tradicional ou SGD;
- **Resolução de Entrada:** 640×640 pixels. Mantida a resolução nativa padrão das arquiteturas YOLO pré-treinadas no *dataset* COCO, evitando distorções excessivas no redisionamento e preservando as características espaciais aprendidas originalmente.

Além disso, foi utilizada a técnica de transferência de aprendizado (*transfer learning*), iniciando os treinamentos com pesos pré-treinados no *dataset* COCO, acelerando a convergência inicial da rede.

4.3 Pipeline de Reconstrução e Validação

Após a detecção dos caracteres pelo modelo YOLO, os dados brutos (classes e coordenadas das caixas delimitadoras) não possuem, por si sós, significado matemático. Para converter essas detecções em um veredito de "Correto"ou "Incorreto", foi desenvolvido um algoritmo de pós-processamento dividido em três etapas, utilizando a linguagem Python e a biblioteca de computação simbólica *SymPy*.

Algorithm 1 Reconstrução e Validação de Cálculos

Require: Lista de detecções D do YOLO (classe, $x_{min}, y_{min}, x_{max}, y_{max}$)
Ensure: Imagem anotada com o veredito matemático

// Etapa 1: Reconstrução Estrutural
Ordenar detecções em D horizontalmente pelo eixo x_{min} (leitura esquerda $\rightarrow$ direita);
for cada detecção i em D **do**
    if i e o anterior $(i-1)$ são dígitos **then**
        Concatenar i à string sem espaços (ex: "1"e "0"vira "10");
    else
        Concatenar com espaço delimitador (ex: "10"e "+"vira "10 + ");
    end if
end for
// Etapa 2: Validação Simbólica (SymPy)
Dividir string no sinal "="em $Lado\_Esquerdo$ e $Lado\_Direito$;
$V_{Esq} \leftarrow SymPy.evalf(Lado\_Esquerdo)$;
$V_{Dir} \leftarrow SymPy.evalf(Lado\_Direito)$;
if $|V_{Esq} - V_{Dir}| < 0.0001$ **then**
    $Veredito \leftarrow$ "CORRETO"(Cor: Verde);
else
    $Veredito \leftarrow$ "INCORRETO"(Cor: Vermelho);
end if
// Etapa 3: Feedback Visual (OpenCV)
Identificar extremidades (min/max) de todas as boxes para criar a "Super Box";
Desenhar moldura colorida e rótulo de texto na imagem original;

A primeira etapa é a Reconstrução Estrutural. O algoritmo recebe a lista de detecções desordenadas fornecidas pelo YOLO e as ordena espacialmente com base na coordenada horizontal (x) de cada *bounding box*, da esquerda para a direita. Nesta fase, foi implementada uma lógica de agrupamento para concatenar dígitos adjacentes (por exemplo, as detecções '1' e '0' próximas tornam-se o número "10"), resultando em uma *string* de texto que representa a equação completa (exemplo: "10 + 5 = 15").

A segunda etapa é a Validação Simbólica. A *string* reconstruída é processada pela biblioteca *SymPy*. Diferente de métodos tradicionais de avaliação de texto, o *SymPy* realiza um *parsing* matemático seguro da expressão. O sistema divide a equação no sinal de igualdade, calcula o valor numérico exato do lado esquerdo e do lado direito e os compara. Se a diferença absoluta entre os valores for inferior a uma tolerância pré-definida (para evitar erros de ponto flutuante), o cálculo é classificado como correto.

Por fim, a terceira etapa é o Feedback Visual. Com base no veredito do *Sympy*, o sistema utiliza a biblioteca *OpenCV* para desenhar uma nova caixa delimitadora ("super box") ao redor de toda a equação na imagem original. Esta caixa é colorida semanticamente (verde para acertos, vermelho para erros), automatizando o processo de correção visual para o usuário final.

5 RESULTADOS E DISCUSSÃO

Nesta seção, são apresentados os dados obtidos após o treinamento das três arquiteturas selecionadas (YOLOv11s, YOLOv9s e YOLOv8s), bem como a avaliação da eficácia da pipeline completa de verificação de cálculos.

5.1 Análise do Treinamento: YOLOv11s

A Figura 8 apresenta as curvas de aprendizado obtidas durante o treinamento da arquitetura YOLOv11s ao longo de 100 épocas. A análise dos gráficos revela um comportamento de convergência extremamente rápido e estável.

Observando as curvas de perda (Box Loss, Class Loss e DFL Loss), nota-se um declínio acentuado logo nas primeiras 20 épocas, indicando que a rede foi capaz de extrair as características fundamentais dos caracteres manuscritos rapidamente. É importante destacar que as curvas de validação (linha inferior) acompanharam a descida das curvas de treino e estabilizaram sem apresentar tendências de subida, o que comprova a ausência de *overfitting*. Isso demonstra que as técnicas de regularização e as *augmentations* aplicadas no *dataset* foram eficazes.

Em relação às métricas de desempenho, o modelo atingiu níveis de excelência. A precisão e a revocação (*Recall*) aproximaram-se de 1.0 muito rapidamente, estabilizando-se nesse patamar por volta da época 40. Como consequência direta desses altos valores, o *F1-Score* (média harmônica que avalia o equilíbrio entre a precisão e o *Recall*) atingiu o valor de pico de **0.99**, mostrando uma robustez excepcional do modelo. O mAP50 manteve-se consistente acima de 0,99, enquanto o mAP50-95 apresentou um crescimento contínuo, terminando o treinamento próximo a 0,96. Um detalhe técnico relevante pode ser observado próximo à época 90 nas curvas de perda de treino, onde ocorre uma queda repentina final; este comportamento é característico do desligamento programado da técnica de *Mosaic Augmentation*, permitindo o refinamento final do modelo com imagens reais não distorcidas.

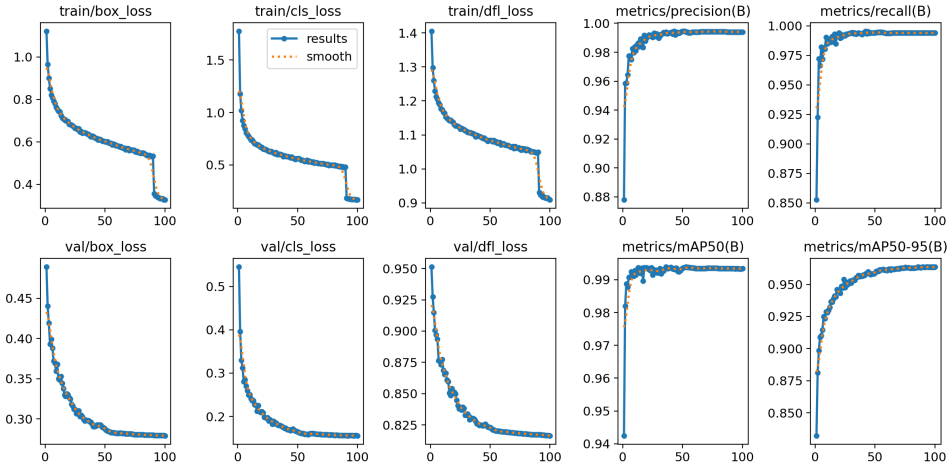


Figura 8 – Curvas de aprendizado e métricas da YOLOv11s após 100 épocas. Fonte: Autor

5.2 Análise do Treinamento: YOLOv9s

A arquitetura YOLOv9s, reconhecida por introduzir conceitos como Informação de Gradiente Programável (PGI), também apresentou um desempenho excepcional no *dataset* proposto, conforme ilustrado na Figura 9. As curvas de aprendizados revelam uma adaptação rápida à tarefa de detecção de caracteres matemáticos.

Observa-se que as curvas de perda (*box loss* e *class loss*) apresentaram um decaimento consistente e suave, indicando uma otimização eficiente dos parâmetros da rede sem oscilações instáveis. A ausência de divergência entre as curvas de treino e validação reafirma a generalização do modelo.

Nas métricas de precisão, a YOLOv9s demonstrou alta competitividade. O mAP50 estabilizou-se rapidamente acima de (0.99), enquanto a métrica mAP50-95 atingiu seu ponto mais alto próximo a (0.96) nas épocas finais. A curva de Revocação (*Recall*) merece destaque, aproximando-se de (1.0), o que sugere que a arquitetura v9 é particularmente sensível para evitar falsos negativos, garantindo que quase nenhum caractere da equação deixe de ser detectado.

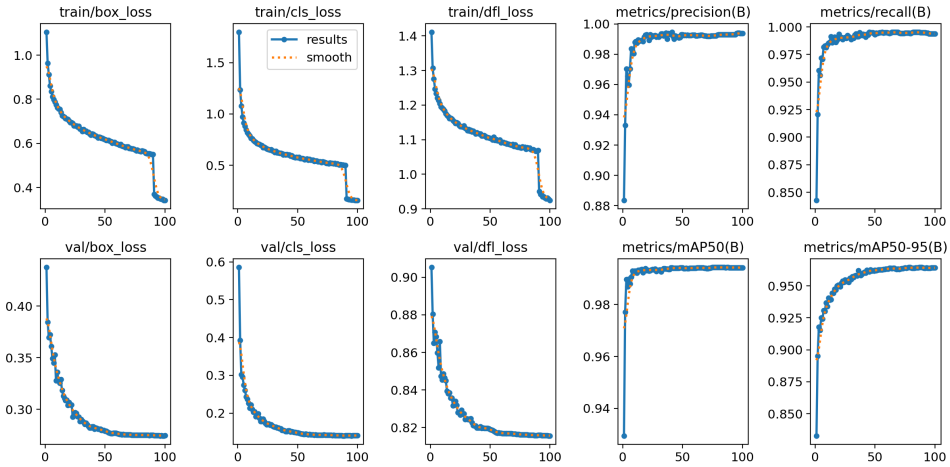


Figura 9 – Curvas de aprendizado e métricas da YOLOv9s após 100 épocas. Fonte: Autor

5.3 Análise do Treinamento: YOLOv8s

As curvas de aprendizado referentes ao treinamento da arquitetura YOLOv8s, apresentadas na Figura 10, demonstram uma estabilidade notável desde as primeiras épocas. Assim como observado na versão YOLOv11s, a YOLOv8s apresentou uma convergência acelerada, com as funções de perda (*loss functions*) de classificação e de caixa caindo drasticamente antes da época 20.

A análise das métricas de validação reforça a robustez desta arquitetura. Embora a Precisão e o *recall* tenham atingido valores próximos à totalidade (1.0) rapidamente, esse comportamento não indicou sobreajuste (*overfitting*), uma vez que as curvas de perda de validação acompanharam o decrescimento do treino sem apresentar divergência. O mAP50 (0.99) e o

mAP50-95 (aproximadamente 0.96) confirmam que o modelo não apenas detecta os caracteres corretamente, mas também ajusta as caixas delimitadoras com alta precisão espacial.

É relevante notar que, mesmo sendo uma versão anterior à v11, a YOLOv8 não apresentou sinais de *underfitting* ou dificuldade em aprender as características do *dataset*. O comportamento das curvas de perda de validação, que permanecem decrescentes e alinhadas às de treino, corrobora a ausência de *overfitting*, justificando a decisão de manter uma *patience* mais alta (50 épocas) em vez de interromper o treinamento prematuramente. Essa estratégia permitiu que o modelo alcançasse o evento de desligamento das *augmentations* de mosaico (próximo à época 90), o que gerou o refinamento final esperado, reduzindo ainda mais a perda de caixa e consolidando as métricas finais.

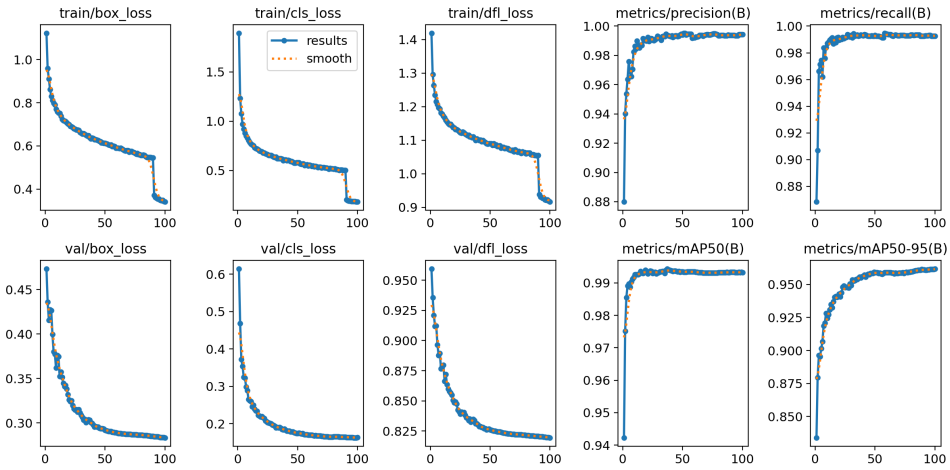


Figura 10 – Curvas de aprendizado e métricas da YOLOv8s após 100 épocas. Fonte: Autor

5.4 Discussão Comparativa e Seleção do Modelo

A Tabela 4 apresenta o comparativo final entre as três arquiteturas avaliadas. A análise dos dados revela um cenário de alta competitividade, onde todos os modelos atingiram um mAP50-95 superior a (0,96), validando a qualidade do *dataset* customizado. No entanto, critérios de desempate foram necessários para a seleção do modelo final.

A arquitetura YOLOv9s apresentou o melhor desempenho absoluto em termos de precisão, atingindo um mAP50-95 de **0,9645**. Contudo, este ganho marginal de precisão (apenas 0.0008 superior à v11) veio acompanhado de um custo computacional significativo: seu tempo de dedução foi de 15,6 ms, tornando-a aproximadamente 79% mais lenta que a concorrente mais veloz.

Por outro lado, a YOLOv8s destacou-se como a arquitetura mais eficiente, com o menor tempo de dedução (**8,7 ms**), embora tenha apresentado o menor mAP (**0.9618**) entre as três. O tempo de execução mais elevado da YOLOv9s justifica-se por sua arquitetura baseada em GELAN (*Generalized Efficient Layer Aggregation Network*) e PGI (*Programmable Gradient Information*) (WANG; YEH; LIAO, 2024). Essas tecnologias, embora refinem a captura de características e evitem a perda de informação em camadas profundas, aumentam a carga

computacional e a latência do modelo. A YOLOv11s posicionou-se como um meio-termo robusto, com precisão virtualmente idêntica à v9 (**0,9637**) e velocidade competitiva à v8 (**9.1 ms**).

Considerando o objetivo de aplicação em dispositivos móveis, onde a latência é um fator crítico, optou-se por selecionar a YOLOv11s como o motor definitivo da pipeline. Sua arquitetura oferece o melhor equilíbrio, garantindo a precisão necessária para a validação correta pelo *SymPy* sem comprometer a fluidez da experiência do usuário.

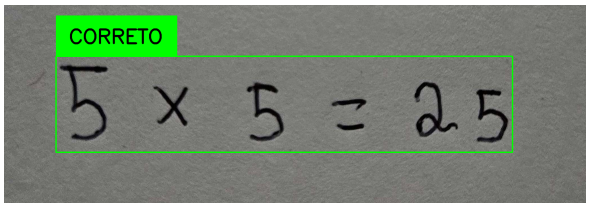
Tabela 4 – Comparativo detalhado das métricas de desempenho e eficiência

Modelo	Precisão (P)	Recall (R)	F1-Score	mAP (50-95)	Tempo (ms)
YOLOv8s	**0.9944**	0.9926	0.9935	0.9618	**8.7**
YOLOv9s	0.9923	**0.9945**	0.9934	**0.9645**	15.6
YOLOv11s	0.9941	0.9940	**0.9941**	0.9637	9.1

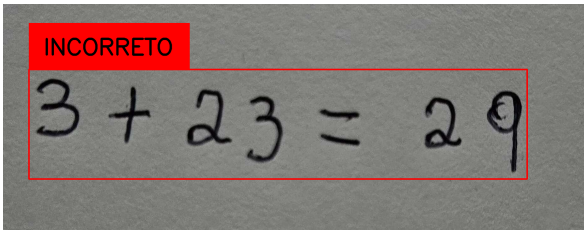
Fonte: O autor.

5.5 Validação Visual da Pipeline

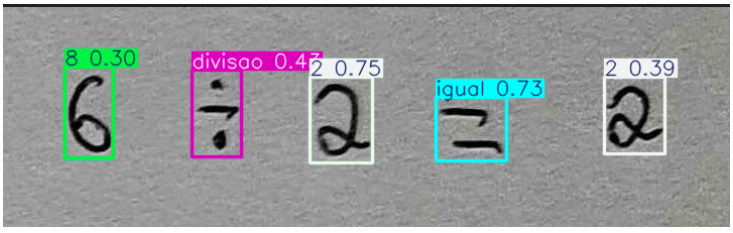
A eficácia da integração entre a detecção visual (YOLO) e a validação lógica (*SymPy*) pode ser visualizada na Figura 11. Esta etapa confirma a hipótese de que o sistema é capaz de automatizar o *feedback* ao usuário.



(a) Cálculo validado corretamente.



(b) Identificação de erro lógico.



(c) Falha de classificação (6 lido como 8).

Figura 11 – Resultados da *pipeline* completa de verificação. O sistema aplica uma "Super Box"com cores semânticas (verde para acertos e vermelho para erros) e demonstra a sensibilidade do modelo a caligrafias ambíguas.

Na Figura 11a, o sistema **(utilizando a arquitetura YOLOv11s selecionada na etapa anterior)** processou a imagem de um cálculo matematicamente válido. O algoritmo de reconstrução extraiu corretamente a sequência de caracteres e seus operadores. O motor *SymPy* processou

a expressão, verificou a igualdade entre os lados e retornou o status de aprovação. Como resposta, o sistema gerou um *bounding box* verde ao redor da expressão, acompanhada do rótulo "CORRETO", validando o fluxo completo de acerto. Em contrapartida, a Figura 11b demonstra a capacidade crítica do sistema de identificar inconsistências. Ao analisar uma expressão em que o resultado manuscrito não correspondia à operação matemática, o motor de validação detectou a divergência lógica nos valores numéricos. Imediatamente, a pipeline sinalizou o erro visualmente, desenhando uma caixa vermelha com o rótulo "INCORRETO".

Contudo, a investigação das limitações do modelo revela cenários de fala crítica na etapa inicial de visão computacional. Na Figura 11c, observa-se um erro de classificação na qual o número "6"foi interpretado como "8", apresentando um baixo índice de confiança de 0,30.

Esta inconsistência é atribuída à ambiguidade morfológica da caligrafia manuscrita, a curvatura na parte superior do número "6"assemelha-se estruturalmente ao fechamento do número "8", o que pode confundir a extração de características espaciais pela rede neural em condições de escrita menos padronizada. Tais erros de detecção propagam-se por toda a *pipeline*, resultando em uma *string* de entrada para a validação simbólica ($8 \div 2 = 2$) que diverge da expressão originalmente escrita ($6 \div 2 = 2$). Este fenômeno ressalta que a confiabilidade do veredicto final do sistema é estritamente dependente da acurácia alcançada na etapa de detecção e classificação visual.

Apesar dessas limitações pontuais, estes resultados qualitativos confirmam que a arquitetura **YOLOv11s** não apenas reconhece os caracteres, mas, em conjunto com o módulo lógico, cumpre o objetivo central de automatizar a correção de exercícios matemáticos.

6 CONCLUSÃO

Este trabalho teve como objetivo desenvolver e avaliar uma solução automatizada para a verificação de cálculos matemáticos manuscritos, integrando técnicas de visão computacional e validação lógica. A proposta buscou reduzir o esforço manual de correção em ambientes educacionais, utilizando redes neurais convolucionais modernas para a detecção de caracteres e algoritmos simbólicos para a conferência matemática.

Os experimentos realizados demonstraram que a utilização da arquitetura YOLO, treinada em um *dataset* híbrido (composto por dados públicos e imagens coletadas manualmente), foi eficaz na tarefa de reconhecer dígitos e operadores matemáticos em condições variadas de escrita. A análise comparativa entre as versões YOLOv8s, YOLOv9s e YOLOv11s revelou um cenário de alta competitividade, onde todos os modelos atingiram métricas de precisão (mAP50-95) superiores a 0,96.

A escolha da **YOLOv11s** como motor definitivo da solução mostrou-se acertada, pois o modelo ofereceu o melhor equilíbrio entre desempenho e custo computacional. Com uma latência de inferência de 9,1 ms e precisão praticamente idêntica à da versão mais robusta (v9s), a YOLOv11s viabilizou a aplicação do sistema em cenários que exigem resposta rápida, como dispositivos móveis. Além disso, a integração com a biblioteca *SymPy* garantiu que o sistema

não apenas "lesse"os caracteres, mas compreendesse a estrutura lógica da equação, fornecendo *feedbacks* visuais (Correto/Incorreto) com confiabilidade matemática.

Apesar do sucesso da *pipeline*, a análise qualitativa revelou limitações em casos de caligrafias ambíguas, onde a semelhança morfológica entre caracteres (como os números "6"e "8") pode induzir o modelo a erros de classificação com baixos índices de confiança. Tais inconsistências demonstram que a precisão do veredito final é estritamente dependente da acurácia da detecção visual inicial.

Portanto, conclui-se que a *pipeline* proposta atendeu aos objetivos da pesquisa, comprovando ser uma ferramenta viável e promissora para o auxílio de professores e estudantes no processo de correção de atividades matemáticas básicas.

6.1 Trabalhos Futuros

Apesar dos resultados satisfatórios, o desenvolvimento deste trabalho abriu novas perspectivas de aprimoramento e investigação. Como sugestões para trabalhos futuros, destacam-se:

- **Expansão do *Dataset*:** Ampliar a base de dados com uma maior variedade de caligrafias e condições de iluminação, visando aumentar a robustez do modelo em cenários de uso real não controlados;
- **Suporte a Operações Complexas:** Estender a capacidade do sistema para reconhecer e validar expressões matemáticas mais avançadas, incluindo frações, potências, raízes quadradas e equações algébricas com variáveis;
- **Aprimoramento do Algoritmo de Reconstrução:** Refinar a lógica de pós-processamento para lidar melhor com equações dispostas verticalmente ou em múltiplas linhas, superando as limitações da leitura puramente linear;

REFERÊNCIAS

ALPAYDIN, E. **Introduction to Machine Learning, fourth edition**. MIT Press, 2020. (Adaptive Computation and Machine Learning series). ISBN 9780262043793. Disponível em: <https://books.google.com.br/books?id=tZnSDwAAQBAJ>.

CIMIRRO, J. L. de S. **Reconhecimento de imagens: uso do método YOLO no reconhecimento de placas de trânsito**. Dissertação (Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação)) — Universidade Federal do Pampa, Alegrete, RS, 2022.

ESCOBAR, F. C. da C. **Investigando erros em matemática: fatores que interferem na aprendizagem dos educandos**. Dissertação (Dissertação (Mestrado Profissional em Educação Matemática)) — Universidade Federal de Juiz de Fora, Juiz de Fora, MG, 2016.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. MIT Press, 2016. (Adaptive Computation and Machine Learning series). ISBN 9780262035613. Disponível em: <https://books.google.com.br/books?id=Np9SDQAAQBAJ>.

KACEM, A. Deep Learning based-framework for Math Formulas Understanding. **ELCVIA Electronic Letters on Computer Vision and Image Analysis**, v. 23, n. 2, set. 2024. ISSN 1577-5097. Disponível em: <https://elcvia.cvc.uab.cat/article/view/1833>.

KAHNEMAN, D. **Rápido e devagar: Duas formas de pensar**. [S.l.]: Objetiva, 2012. Google-Books-ID: d3FloqhQHgQC. ISBN 978-85-390-0401-0.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998.

LECUN, Y.; CORTES, C.; BURGES, C. Mnist handwritten digit database. **ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist**, v. 2, 2010.

MEURER, A. et al. SymPy: symbolic computing in Python. **PeerJ Computer Science**, v. 3, p. e103, jan. 2017. ISSN 2376-5992. Disponível em: <https://peerj.com/articles/cs-103>.

PIEMONTEZ. **YOLO Versões 1 e 2 (Arquitetura)**. 2023. Disponível em: <https://visaocomputacional.com.br/yolo-versoes-1-e-2-arquitetura/>.

REDMON, J. et al. You Only Look Once: Unified, Real-Time Object Detection. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. Las Vegas, NV, USA: IEEE, 2016. p. 779–788. ISBN 978-1-4673-8851-1. Disponível em: <http://ieeexplore.ieee.org/document/7780460/>.

REDMON, J.; FARHADI, A. **YOLOv3: An Incremental Improvement**. arXiv, 2018. ArXiv:1804.02767 [cs]. Disponível em: <http://arxiv.org/abs/1804.02767>.

ROSA, D. et al. **Recognizing Handwritten Mathematical Expressions of Vertical Addition and Subtraction**. arXiv, 2023. ArXiv:2308.05820 [cs]. Disponível em: <http://arxiv.org/abs/2308.05820>.

ULTRALYTICS. **YOLO11 NOVO**. 2024. Disponível em: <https://docs.ultralytics.com/pt/models/yolo11>.

WANG, C.-Y.; YEH, I.-H.; LIAO, H.-Y. M. Yolov9: Learning what you want to learn using programmable gradient information. In: SPRINGER. **European conference on computer vision**. [S.l.], 2024. p. 1–21.

WANG, Z.; LIU, J.-C. Translating math formula images to LaTeX sequences using deep neural networks with sequence-level training. **International Journal on Document Analysis and Recognition (IJDAR)**, v. 24, n. 1-2, p. 63–75, jun. 2021. ISSN 1433-2833, 1433-2825. Disponível em: <https://link.springer.com/10.1007/s10032-020-00360-2>.