UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ROBSON CAZUO HORIQUINI

Comparação de métodos para identificação de defeitos em placas de circuito impresso usando redes neurais convolucionais

SÃO CARLOS -SP
2025

ROBSON CAZUO HORIQUINI

Comparação de métodos para identificação de defeitos em placas de circuito impresso usando redes neurais convolucionais

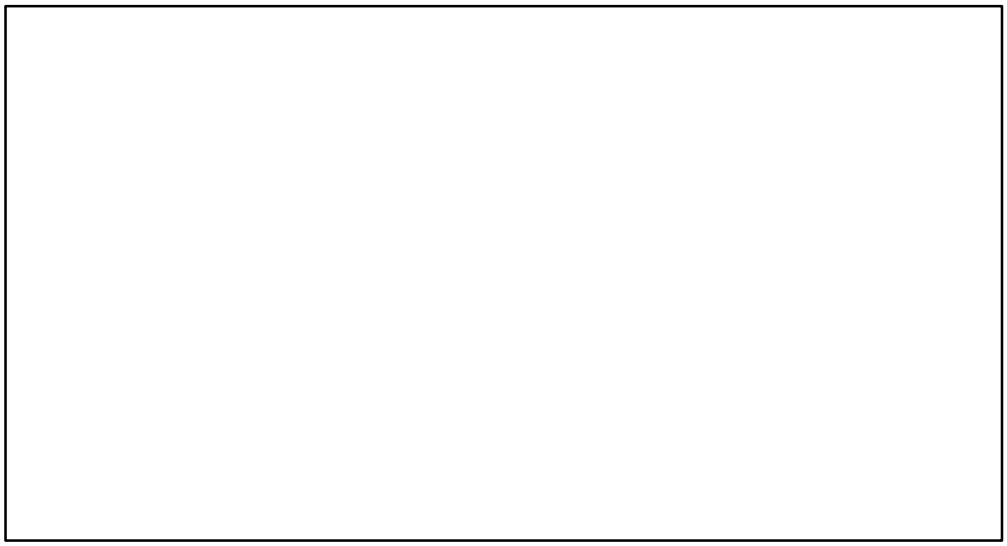
Trabalho de Conclusão de Curso apresentada ao Departamento de Engenharia Elétrica do Centro de Ciências Exatas e de Tecnologia da Universidade Federal de São Carlos, para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Robson Barcellos

São Carlos-SP
2025

FICHA CATALOGRÁFICA

Faça a sua preenchendo ESTE FORMULÁRIO AQUI.



UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Departamento de Engenharia Elétrica

Folha de aprovação

Assinatura dos membros da comissão examinadora que avaliou e aprovou o Trabalho de Conclusão de Curso do candidato Robson Cazuo Horiquini, realizada em 18/07/2025:

_____

Prof. Dr. Robson Barcellos
Departamento de Engenharia Elétrica
Universidade Federal de São Carlos

_____

Prof. Dr. Amilcar Flamarion Querubini Gonçalves
Departamento de Engenharia Elétrica
Universidade Federal de São Carlos

_____

Prof. Dr. Arlindo Neto Montagnoli
Departamento de Engenharia Elétrica
Universidade Federal de São Carlos

À minha família.

AGRADECIMENTO

Gostaria de agradecer à minha mãe, Aparecida, que sempre esteve presente na minha vida, cuidando de mim e minhas irmãs a todo momento e que se preocupou muito comigo longe de casa. Meu pai, Marcos, quem é uma pessoa incrível, um exemplo de pai, filho e amigo, por quem me espelho e sonho em ser 1% do ser humano que ele é. Essas duas pessoas são meu alicerce na vida e sem eles nada disso seria possível.

À minha avó, dona Jandira, que sempre que me vê pergunta do curso e ao meu avô, senhor Antônio, que é uma pessoa presente e amorosa.

Aos amigos que fiz durante a vida, em particular as amizades do ensino médio que estão presentes até os dias atuais, as pessoas que conheci nos projetos que participei, em especial no Taekwondo UFSCar, que me trouxe pessoas incríveis e que me ajudaram no processo de desenvolvimento deste trabalho.

Ao meu orientador, professor doutor Robson, que me cobrou na medida certa para atendermos o cronograma e sempre guiou nossas reuniões de forma muito agradável.

Por fim, agradeço a todas as pessoas com quem tive uma troca boa, de conhecimentos, risadas e bons momentos, mesmo que breve cada contato foi importante para minha formação como pessoa.

RESUMO

Este trabalho apresenta uma análise comparativa entre os modelos de aprendizado profundo YOLOv11 e Detectron2 para identificação e classificação de defeitos em placas de circuito impresso (PCIs).

A pesquisa abordou a crescente demanda por sistemas de Inspeção Óptica Automatizada (AOI) mais eficientes que os métodos tradicionais baseados em regras predefinidas, os quais apresentam limitações frente à variabilidade dos processos produtivos e surgimento de novos tipos de defeitos. O estudo utilizou técnicas de visão computacional baseadas em redes neurais convolucionais para detectar e classificar defeitos comuns em PCIs, incluindo curtos-circuitos, interrupções de trilha, falta de furos e excesso de cobre. A metodologia empregada envolveu o treinamento de ambos os modelos utilizando um *dataset* específico de imagens de PCIs com defeitos anotados, seguido da avaliação comparativa através de métricas de desempenho como mAP_{50} e $mAP_{50\text{-}95}$. Os resultados monstraram que o YOLOv11 alcançou desempenho superior com mAP_{50} de 0,98 e $mAP_{50\text{-}95}$ de 0,57, comparado aos valores de 0,92 e 0,41 do Detectron2, respectivamente. Em termos de convergência, o YOLOv11 apresentou processo de aprendizado gradual e consistente ao longo de 100 épocas com *early stopping*, enquanto o Detectron2 demonstrou convergência mais rápida e estável, estabilizando em aproximadamente 2000 iterações. O Detectron2 destacou-se pela maior previsibilidade e estabilidade no processo de treinamento, características vantajosas para aplicações industriais que exigem reprodutibilidade. As conclusões indicam que ambas as arquiteturas são tecnicamente viáveis para a aplicação proposta, sendo a escolha dependente de fatores específicos como recursos computacionais disponíveis, requisitos de tempo real e necessidades de integração com sistemas existentes. O trabalho contribui para a indústria de manufatura eletrônica fornecendo subsídios baseados em dados para a tomada de decisão sobre implementação de tecnologias de detecção de defeitos em linhas de produção, visando melhoria da qualidade, redução de custos operacionais e aumento da produtividade.

Palavras-chave: detecção de defeitos, placas de circuito impresso, YOLOv11, Detectron2, aprendizado profundo, visão computacional.

ABSTRACT

This paper presents a comparative analysis between the YOLOv11 and Detectron2 deep learning models for identifying and classifying defects in printed circuit boards (PCBs).

The research addresses the growing demand for Automated Optical Inspection (AOI) systems that are more efficient than traditional methods based on predefined rules, which have limitations in the face of the variability of production processes and the emergence of new types of defects. The study uses computer vision techniques based on convolutional neural networks to detect and classify common defects in PCBs, including short circuits, track interruptions, missing holes, and excess copper. The methodology employed involved training both models using a specific dataset of PCB images with annotated defects, followed by comparative evaluation through performance metrics such as mAP_{50} and mAP_{50-95}. The results demonstrate that YOLOv11 achieved superior performance with mAP_{50} of 0.98 and mAP_{50-95} of 0.57, compared to Detectron2's values of 0.92 and 0.41, respectively. In terms of convergence, YOLOv11 presented a gradual and consistent learning process over 100 epochs with early stopping, while Detectron2 demonstrated faster and more stable convergence, stabilizing in approximately 2000 iterations. Detectron2 stood out for its greater predictability and stability in the training process, advantageous characteristics for industrial applications that require reproducibility. The conclusions indicate that both architectures are technically feasible for the proposed application, with the choice depending on specific factors such as available computational resources, real-time requirements, and integration needs with existing systems. The work contributes to the electronic manufacturing industry by providing data-based support for decision-making on the implementation of defect detection technologies in production lines, aiming at improving quality, reducing operational costs and increasing productivity.

Keywords: defect detection, drinted circuit boards, YOLOv11, Detectron2, deep learning, domputer vision.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

AOI – *Automated Optical Inspection*

API – *Application Programming Interface*

CNN – *Convolutional Neural Network*

COCO – *Common Objects in Context*

CSV – *Comma Separated Values*

FAIR – *Facebook IA Research*

FN – *False Negative*

FP – *False Positive*

FPN – *Feature Pyramid Network*

GPU – *Graphics Processing Unit*

IA – Inteligência Artificial

IoU – *Intersection over Union*

JSON – *JavaScript Object Notatio*

mAP – *mean Average Precision*

PCB – *Printed Circuit Board*

PCI – Placa de Circuito Impresso

PLN – Processamento de linguagem natural

R-CNN – *Region-based Convolutional Neural Network*

ReLU – Unidade Linear Retificada

ResNet – *Residual Networks*

RPN – *Region Proposal Network*

RTMDet – *Real-Time Models for Detection*

TN – *True Negative*

TP – *True Positive*

TXT – *Text*

XML – *Extensible Markup Language*

YAML – *Yet Another Markup Language*

YOLO – *You Only Look Once*

SUMÁRIO

1 INTRODUÇÃO

A indústria eletrônica moderna depende intrinsecamente das Placas de Circuito Impresso (PCIs), componentes fundamentais que fornecem a base física e elétrica para a interconexão de componentes eletrônicos. A crescente demanda por dispositivos eletrônicos mais complexos, compactos e confiáveis impulsiona a necessidade de métodos de fabricação de PCIs cada vez mais precisos e eficientes. Nesse contexto, a qualidade e a confiabilidade das PCIs produzidas impactam diretamente o desempenho e a durabilidade dos produtos finais, tornando a garantia de qualidade uma etapa crítica e indispensável no ciclo de produção como dito por SUSA *et al.*, 2020.

Apesar dos avanços nas técnicas de fabricação, a ocorrência de defeitos em PCIs é inevitável, podendo ser resultado de variações no material, falhas no equipamento ou erros humanos. Defeitos comuns incluem, mas não se limitam a curtos-circuitos, interrupções de trilha, falta de furos, excesso de cobre e trilhas desconectadas dito em DING *et al.*, 2019. A identificação e classificação precisa desses defeitos são cruciais para evitar que produtos não conformes cheguem ao mercado. A falha na detecção de defeitos acarreta custos significativos com retrabalho, descarte de material, falhas em campo, insatisfação do cliente e danos à reputação da marca.

Historicamente, a inspeção de qualidade em PCIs era predominantemente realizada de forma manual por operadores humanos. Embora essa abordagem permita a identificação de uma variedade de defeitos, LETA; FELICIANO; MARTINS, 2008 comenta que ela é inerentemente subjetiva, suscetível à fadiga, lenta e, portanto, impraticável para inspeção em larga escala e em linhas de produção de alta velocidade.

A necessidade de uma inspeção mais rápida, consistente e confiável impulsionou o desenvolvimento de sistemas de Inspeção Óptica Automatizada (AOI - *Automated Optical Inspection*). Sistemas AOI convencionais frequentemente utilizam técnicas baseadas em regras predefinidas, comparação de imagens com modelos de referência ou análise estatística mostrado por TSAI; CHOU, 2020. Embora eficazes para certos tipos de defeitos, esses métodos podem apresentar dificuldades com variações aceitáveis no processo, o surgimento de novos tipos de defeitos ou ambientes de iluminação complexos, exigindo ajustes frequentes e um

extensivo conhecimento de domínio para programação e manutenção.

Recentemente, os avanços exponenciais no campo da Visão Computacional, impulsionados pelo Aprendizado de Máquina e, em particular, pelo Aprendizado Profundo, revolucionaram a capacidade das máquinas de "enxergar" e interpretar imagens complexas. Modelos baseados em aprendizado profundo, como as Redes Neurais Convolucionais (CNNs), têm alcançado resultados expressivos em diversas aplicações de processamento visual. Sua capacidade abrange desde a identificação do conteúdo geral de uma imagem, até a distinção e delimitação de diferentes regiões que compõem uma cena. Adicionalmente, as CNNs demonstram eficácia na localização e identificação de múltiplos objetos de interesse dentro de um campo visual.

Dentro do escopo do aprendizado profundo aplicado à visão computacional, a tarefa de detecção de objetos é particularmente relevante e poderosa para a inspeção de PCIs. Diferentemente da classificação de imagem, que apenas indica a presença de um defeito na imagem, a detecção de objetos é capaz de localizar os defeitos através de caixas delimitadoras (*bounding boxes*) e classificá-los em categorias específicas (como curto, aberto, arranhão, etc.) em uma única etapa. Isso fornece informações precisas sobre a localização e o tipo do defeito, que são essenciais para processos subsequentes de retrabalho ou análise da causa raiz.

No entanto, a escolha do algoritmo ou framework de detecção de objetos mais adequado para uma aplicação industrial específica, como a inspeção de defeitos em PCIs durante a fase de produção, não é trivial. Diferentes modelos possuem características e desempenhos variados que dependem diretamente das particularidades do *dataset* utilizado, dos tipos de defeitos a serem identificados e dos requisitos operacionais de velocidade e precisão. Modelos da família YOLO, por exemplo, são frequentemente reconhecidos por sua alta velocidade de inferência, enquanto frameworks como o Detectron2, que suportam diversas arquiteturas, podem oferecer níveis superiores de precisão, mas geralmente com maior custo computacional.

A motivação principal desta pesquisa reside em contribuir para a indústria de manufatura eletrônica, fornecendo uma análise comparativa baseada em dados representativos de cenários reais. O objetivo é avaliar a eficácia e a viabilidade de modelos de ponta de detecção de objetos para a tarefa específica de identificação e classificação de defeitos em PCIs durante a fase de produção. Compreender as

forças e fraquezas de cada abordagem em um contexto industrial prático é fundamental para auxiliar na tomada de decisão sobre a implementação dessas tecnologias em linhas de produção reais, visando a melhoria contínua da qualidade, a redução de custos operacionais e o aumento da produtividade.

O escopo deste trabalho abrange a utilização de técnicas de aprendizado profundo para a detecção e classificação de um conjunto específico de defeitos comuns em PCIs, a partir de imagens visíveis capturadas em ambiente de produção. O estudo se limitará à comparação do desempenho dos modelos YOLOv11 e Detectron2 utilizando um conjunto de dados predefinido de imagens de PCIs com defeitos. As métricas de comparação incluiram a qualidade geral da detecção dos defeitos efetuada pelos modelos, sua velocidade operacional e outros indicadores pertinentes ao desempenho no conjunto de dados de teste, avaliando sua adequação para a aplicação industrial proposta.

2 REVISÃO BIBLIOGRÁFICA

O presente capítulo oferece uma visão geral concisa dos estudos pregressos dedicados à detecção de defeitos em PCIs.

Em BANDERCHUK, 2021, propõe-se usar métodos de inteligência artificial que aprendem com exemplos para encontrar defeitos em PCI. Para treinar e testar o sistema, foi utilizado um conjunto de imagens disponível publicamente, chamado HRIPCB proposto por HUANG; WEI, 2019. O trabalho empregou um tipo específico de rede neural, a YOLOv4 explicado por BOCHKOVSKIY; WANG; MARK LIAO, 2020, conseguindo identificar seis tipos diferentes de defeitos com alta precisão, alcançando um resultado de 98,44%. Além disso, foi criada uma ferramenta online para facilitar a interação e visualização dos resultados.

Já MORAIS, 2023 investigou a detecção de defeitos em placas de circuito impresso utilizando diferentes estruturas de redes neurais, como Tiny-YOLOv4 melhor explicada em JIANG *et al.*, 2020, YOLOv5 abordada por KHANAM; HUSSAIN, 2024 e Faster R-CNN trabalha em REN *et al.*, 2016, avaliando como se adaptam a um dispositivo com recursos limitados, como o Raspberry Pi 3B+. A pesquisa também utilizou a base de dados HRIPCB. Para reduzir a exigência de processamento no treinamento e uso das redes neurais, foram aplicadas técnicas de otimização. O modelo YOLOv5, após otimizações, mostrou bom desempenho no Raspberry Pi, com uma precisão de 97,2%, demonstrando que essa abordagem é viável mesmo com *hardware* de processamento modesto.

Em SILVA, 2023, realizou-se uma comparação entre modelos emergentes de detecção de objetos, incluindo RTMDet explicada com mais detalhes po LYU *et al.*, 2022, YOLO-NAS e YOLOv8 usadas no trabalho de TERVEN; CORDOVA-ESPARZA, 2024, para a classificação em tempo real de pequenos componentes em placas de circuito impresso. O estudo avaliou a acurácia e a velocidade de inferência das redes neurais em um conjunto de dados com diferentes tamanhos de componentes. Essa base de dados foi coletada manualmente e contou com poucas imagens, sendo necessário utilizar técnicas para aumentar os dados. Os resultados demonstraram o potencial em oferecer um melhor equilíbrio entre precisão e desempenho para a inspeção automatizada de PCIs em tempo real.

Em GUIMARÃES, 2024, o autor descreve um método para encontrar defeitos em placas de circuito impresso utilizando a rede neural YOLOv7 também explicada

em TERVEN; CORDOVA-ESPARZA, 2024 e a criação de uma interface de programação (API) com FastAPI. O trabalho envolveu coletar e preparar dados, que também foram fornecidos pela base de dados publica HRIPCB, treinar a rede YOLOv7 e desenvolver uma API eficiente para identificar os defeitos. Os resultados mostraram alta precisão na detecção, destacando a implementação eficaz para inspeção de PCIs e a criação de uma ferramenta útil para uso em indústria e pesquisa.

Os trabalhos apresentados neste capítulo demonstram a crescente aplicação de técnicas de aprendizado profundo na detecção de defeitos em Placas de Circuito Impresso, explorando diferentes arquiteturas de redes neurais e métodos para otimizar o desempenho e a aplicabilidade em sistemas embarcados, sistemas web ou integrado em outros processos. A utilização de conjuntos de dados como o HRIPCB por múltiplos estudos ressalta sua relevância na área de treinamento de redes neurais para detecção de defeitos em PCI e foi utilizado neste trabalho sendo base comparativa para novas pesquisas. A presente revisão mostra o estado da arte e os desafios existentes, servindo como base para o desenvolvimento do trabalho proposto, que buscará contribuir para o avanço dessa importante área de pesquisa.

3 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por objetivo formalizar conceitos importantes para o entendimento e desenvolvimento deste trabalho e abordará a explicação sobre Inteligência Artificial (IA), métricas de avaliação e defeitos de confecção de placa de circuito impresso.

3.1 INTELIGÊNCIA ARTIFICIAL

A Inteligência Artificial (IA) é um campo multidisciplinar da ciência da computação que busca desenvolver sistemas capazes de realizar tarefas que normalmente exigiriam inteligência humana. Essas tarefas incluem, entre outras, aprendizado, raciocínio, resolução de problemas, percepção, compreensão da linguagem e tomada de decisões. O objetivo fundamental da IA é criar máquinas que possam "pensar" ou agir de forma inteligente, de maneira análoga aos seres humanos dito em RUSSELL; NORVIG, 2020.

Historicamente, o campo da IA passou por diversas fases, desde o entusiasmo inicial com a possibilidade de replicar a inteligência humana em sua totalidade, até abordagens mais focadas e especializadas. Atualmente, a IA abrange uma vasta gama de técnicas e abordagens, incluindo aprendizado de máquina (*Machine Learning*), processamento de linguagem natural (PLN), visão computacional, sistemas especialistas e robótica, falado por GOODFELLOW; BENGIO; COURVILLE, 2016.

No contexto industrial, como na produção de Placas de Circuito Impresso, a IA tem se mostrado uma ferramenta poderosa para otimização de processos, controle de qualidade e automação. A capacidade de sistemas de IA em analisar grandes volumes de dados e identificar padrões complexos permite a detecção precoce de anomalias e defeitos, contribuindo para a melhoria da eficiência e a redução de custos como mostrado em LEE *et al.*, 2018. Em WEIMER; SCHOLZ-REITER; SHPITALNI, 2016 a aplicação de IA na inspeção visual de PCI’s, por exemplo, visa superar as limitações da inspeção humana, como fadiga e subjetividade, garantindo maior precisão e consistência na identificação de falhas.

3.1.1 Redes Neurais Convolucionais

As Redes Neurais Convolucionais (*Convolutional Neural Networks* - CNNs) são uma classe especializada de redes neurais artificiais projetadas para processar dados que possuem uma topologia de grade, como imagens e vídeos. Sua arquitetura é inspirada no córtex visual biológico, onde neurônios individuais respondem a estímulos dentro de uma região restrita do campo visual, conhecida como campo receptivo. As CNNs se tornaram o padrão em tarefas de visão computacional, como classificação de imagens, detecção de objetos e segmentação semântica, devido à sua capacidade de aprender automaticamente hierarquias de características a partir dos dados brutos por GUIMARÃES, 2024.

A arquitetura padrão de uma CNN é composta por uma sequência de diferentes tipos de camadas, cada uma com uma função específica no processo de extração e transformação de características:

- **Camada Convolucional** (*Convolutional Layer*): Esta é a camada central de uma CNN. Nela, um conjunto de filtros (ou *kernels*), que são pequenas matrizes de pesos, desliza sobre a imagem de entrada, realizando operações de convolução. Cada filtro é treinado para detectar padrões visuais específicos, como bordas, cantos, texturas ou formas mais complexas em camadas mais profundas. O resultado da aplicação de um filtro sobre a imagem é um mapa de características (*feature map*), que indica a presença e a localização das características detectadas.
- **Camada de Agrupamento** (*Pooling Layer*): O objetivo principal desta camada é reduzir progressivamente a dimensionalidade espacial (largura e altura) dos mapas de características, o que contribui para diminuir a quantidade de parâmetros e cálculos na rede, controlando o *overfitting* e tornando a representação das características mais robusta a pequenas translações e distorções na imagem de entrada. As operações de *pooling* mais comuns são o *Max Pooling* (agrupamento máximo), que seleciona o valor máximo em uma região, e o *Average Pooling* (agrupamento médio), que calcula a média dos valores.
- **Camada de Ativação** (*Activation Layer*): Imediatamente após cada

operação de convolução, uma função de ativação não linear é aplicada elemento a elemento nos mapas de características. Funções como a Unidade Linear Retificada (ReLU) são amplamente empregadas devido à sua eficiência computacional e por ajudarem a mitigar o problema da dissipação do gradiente (*vanishing gradient*) durante o treinamento da rede e permite que a rede aprenda representações mais complexas.

- **Camada Totalmente Conectada** (*Fully Connected Layer*): Após uma série de camadas convolucionais, de ativação e de *pooling*, os mapas de características de alto nível resultantes são geralmente "achatados" (*flattened*) em um vetor unidimensional. Este vetor é então alimentado em uma ou mais camadas totalmente conectadas, onde cada neurônio possui conexões com todos os neurônios da camada anterior, de forma análoga às redes neurais *feedforward* tradicionais, melhor explicado em SILVA; SPATTI; FLAUZINO, 2010. A última camada totalmente conectada frequentemente utiliza uma função de ativação como a Softmax para problemas de classificação multiclasse, gerando uma distribuição de probabilidades sobre as classes de saída.

3.1.1.1 Detecção de Objetos com CNN’s

A detecção de objetos é uma das tarefas mais fundamentais e desafiadoras no campo da visão computacional, com o objetivo de não apenas classificar, mas também localizar instâncias de objetos em imagens ou vídeos. O advento e a consolidação do aprendizado profundo, em especial das Redes Neurais Convolucionais, representaram um marco transformador para esta área, superando drasticamente as abordagens clássicas e estabelecendo novos patamares de precisão e eficiência. As CNNs demonstraram uma capacidade ímpar de aprender hierarquias de características visuais diretamente dos dados, automatizando o processo de extração de características que antes era manual e complexo.

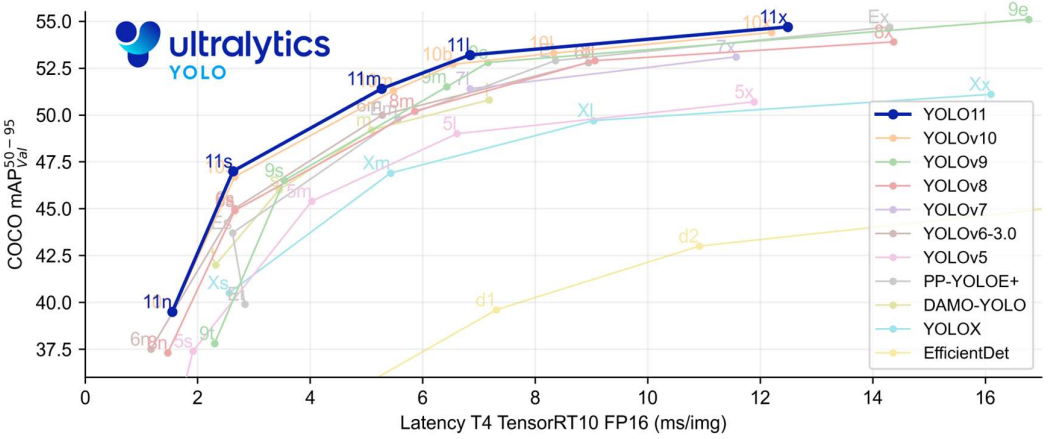
Neste capítulo, exploraremos os alicerces dessa tecnologia. Iniciaremos com uma análise da arquitetura e do funcionamento do YOLO (*You Only Look Once*) e o framework Detectron2, que implementa modelos de dois estágios (*two-stage*). O objetivo é fornecer a fundamentação teórica necessária para compreender o funcionamento, as diferenças e as aplicações dessas ferramentas.

3.1.1.2 YOLO

YOLO (*You Only Look Once*) é um algoritmo de detecção de objetos em tempo real. Criado inicialmente por Joseph Redmon e Ali Farhadi na Universidade de Washington, o YOLO revolucionou a forma como os objetos são detectados ao tratar a tarefa como um problema de regressão único mostrado em ULTRALYTICS, 2025. Isso significa que, em uma única passagem pela rede neural, o modelo é capaz de prever caixas delimitadoras ao redor dos objetos e as probabilidades de classe para cada objeto detectado diretamente a partir da imagem completa. Essa abordagem contrasta com outros métodos que geralmente envolviam múltiplas etapas ou componentes, como a geração de propostas de regiões seguida de classificação. A principal vantagem do YOLO reside em sua velocidade e precisão, tornando-o ideal para aplicações que exigem processamento em tempo real.

A YOLOv11, representa a iteração mais recente e avançada desta família de modelos. Ela não apenas continua o legado de detecção de objetos, mas também expande suas capacidades para incluir segmentação de imagem, estimativa de pose, rastreamento de objetos e classificação. Essa versatilidade consolida o YOLO como uma solução robusta para uma ampla gama de tarefas em Inteligência Artificial visual. A trajetória do YOLO é marcada por contínuas melhorias e otimizações, resultando em diversas versões. A Figura 1 mostra a comparação entre várias versões do YOLO em relação a latência - que representa o tempo de processamento necessário para realizar uma inferência, sendo um indicador crucial da velocidade do modelo - e a métrica mAP^{50-95} (*mean Average Precision*), que mede a precisão média do modelo em diferentes *thresholds* de IoU (*Intersection over Union*), variando de 0,5 a 0,95, sendo uma das métricas mais importantes para avaliar a qualidade da detecção de objetos no conjunto de dados COCO. O COCO (*Common Objects in Context*) é um dos *dataset* mais utilizados para avaliação de algoritmos de detecção de objetos, contendo mais de 330.000 imagens com 80 classes diferentes de objetos do cotidiano, servindo como *benchmark* padrão na comunidade científica.

Figura 1 - Comparação entre as versões do YOLO.



Fonte: https://github.com/ultralytics/ultralytics.

Como abordado em ROSA, 2025, o algoritmo YOLO aborda a detecção de objetos de uma maneira distinta dos métodos tradicionais de detecção em duas etapas (como R-CNN), que primeiro identificam regiões de interesse e depois classificam os objetos. O YOLO realiza tanto a localização quanto a classificação em uma única passagem pela rede neural.

Primeiramente, a imagem de entrada é dividida em regiões de S × S *pixels*, denominadas células, formando uma grade regular. Para cada célula da grade a rede neural assume a responsabilidade por detectar objetos cujo centro geométrico esteja localizado dentro de seus limites. Esta abordagem de divisão espacial permite que o algoritmo processe toda a imagem simultaneamente, ao contrário de métodos que analisam regiões separadamente.

Em cada célula a rede neural é encarregada de predizer N caixas delimitadoras (*bounding boxes*), onde N é um hiper parâmetro definido durante o treinamento do modelo. Uma caixa delimitadora é uma representação retangular que define a localização e as dimensões de um objeto detectado na imagem. Junto com cada caixa, a rede neural calcula uma pontuação de confiança, que é determinada pela Equação (1)

$$Confiança = Pr_{objeto} * IoU \quad (1)$$

Onde Pr_{objeto} representa a probabilidade de existir um objeto na caixa e IoU (*Intersection over Union*) mede a sobreposição entre a caixa predita e a caixa

verdadeira (*ground truth*). Esta pontuação indica simultaneamente a probabilidade de que a caixa contenha um objeto e quão precisa é a localização da caixa em relação ao objeto real.

As predições realizadas pela rede neural para cada caixa delimitadora incluem cinco parâmetros fundamentais:

- As coordenadas (x, y) do centro da caixa delimitadora, expressas como valores relativos às dimensões da célula. Isso significa que x e y variam entre 0 e 1, onde (0,0) corresponde ao canto superior esquerdo da célula e (1,1) ao canto inferior direito.
- A altura (h) e a largura (w) da caixa, normalizadas em relação às dimensões totais da imagem. Assim, h e w também variam entre 0 e 1, onde 1 representaria uma caixa que ocupa toda a altura ou largura da imagem, respectivamente.
- A pontuação de confiança individual da caixa, conforme descrito anteriormente.

É importante destacar que, para cada célula da grade, a rede neural prevê C probabilidades de classe, onde C corresponde ao número total de classes do *dataset* (por exemplo, C = 80 para o *dataset* COCO). Estas probabilidades são calculadas independentemente do número N de caixas delimitadoras que a célula prediz. A probabilidade de classe representa a distribuição de probabilidade sobre todas as classes possíveis, indicando qual tipo de objeto é mais provável de estar presente na célula, assumindo que existe um objeto.

O processamento e a classificação são realizados pela arquitetura da rede neural convolucional, que processa o conteúdo visual de cada célula através de suas camadas convolucionais e totalmente conectadas.

Finalmente, para determinar a probabilidade final de uma caixa específica conter um objeto de uma determinada classe, utiliza-se a Equação (2).

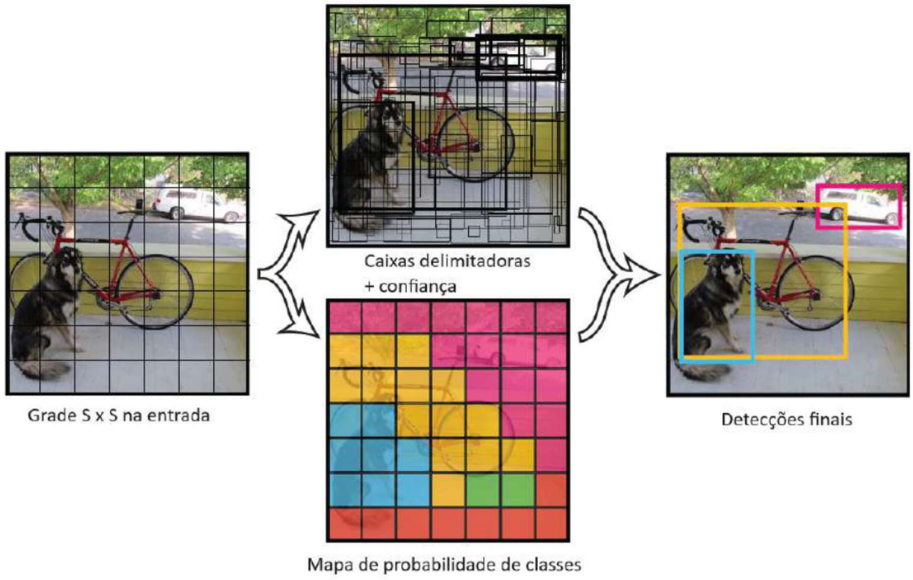
$$Probabilidade_{final} = Confiança_{caixa} * Probabilidade_{classe} \quad (2)$$

A $Probabilidade_{classe}$ refere-se à probabilidade de o objeto pertencer à classe, dado que existe um objeto na célula. Esta multiplicação produz uma pontuação que considera tanto a confiança na detecção do objeto quanto a certeza sobre sua

classificação.

Esse processo de definição das caixas é representado na Figura 2.

Figura 2 - Lógica básica do YOLO.



Fonte: Adaptado de REDMON *et al.*, 2016.

No que se refere ao treinamento, o YOLO oferece flexibilidade através de diferentes estratégias de implementação. Ele pode ser treinado do zero (*training from scratch*), embora seja comum o uso de modelos pré-treinados, como os disponibilizados pela Ultralytics – organização que fornece as versões oficiais do YOLO - que foram treinados com o *dataset* COCO.

Para otimizar o desempenho em tarefas específicas - que se referem a domínios de aplicação particulares que diferem significativamente do *dataset* original COCO, como detecção médica (raios-X, ressonâncias), inspeção industrial (defeitos em componentes eletrônicos, soldas), agricultura de precisão (identificação de pragas, doenças em plantas), ou segurança (detecção de armas, comportamentos suspeitos) - pode-se aplicar a técnica de ajuste fino (*fine-tuning*).

O *fine-tuning* consiste em utilizar um modelo já treinado em um *dataset* amplo e diversificado (como o COCO) como ponto de partida, e então retreiná-lo com um conjunto de dados específico do domínio alvo. Este processo envolve o ajuste dos pesos da rede neural através de uma taxa de aprendizado tipicamente menor que o treinamento inicial, permitindo que o modelo adapte suas características já

aprendidas para as particularidades da nova tarefa.

Esta abordagem é especialmente vantajosa porque aproveita o conhecimento prévio sobre características visuais gerais (bordas, texturas, formas básicas) já codificado nas camadas iniciais da rede, necessitando apenas adaptar as camadas mais profundas para as especificidades do novo domínio. No contexto de detecção de defeitos em placas de circuito impresso, por exemplo, o modelo pré-treinado já possui conhecimento sobre detecção de objetos retangulares e circulares, que são formas comuns de componentes eletrônicos, sendo necessário apenas especializá-lo para identificar as particularidades dos defeitos específicos encontrados em PCIs.

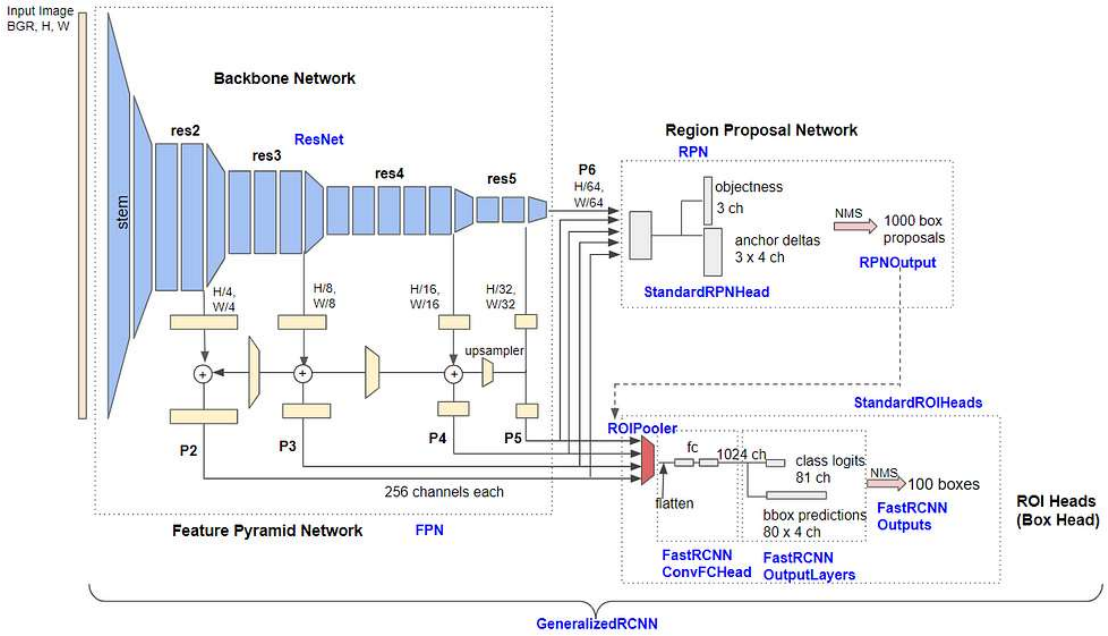
3.1.1.3 Detectron2

Desenvolvido pela equipe do *Facebook IA Research* (FAIR) a Detectron2 é uma plataforma de código aberto para detecção e segmentação de objetos dito por WU *et al.*, 2019.

Lançado um ano após seu antecessor Detectron, o Detectron2 manteve os modelos disponíveis na versão anterior e adicionou novos recursos, como *DensePose* - um sistema que mapeia todas as coordenadas de pixels humanos em uma imagem 2D para a superfície 3D do corpo humano - e *Cascade R-CNN* - uma arquitetura que utiliza múltiplos detectores em cascata com diferentes *thresholds* de IoU para melhorar progressivamente a qualidade das detecções. Além disso, a plataforma permite a exportação dos modelos treinados no formato *TorchScript*, que é um formato de serialização do *PyTorch* que permite executar modelos de forma independente do código Python original, facilitando a implementação em produção e melhorando a performance de inferência, como explicado em WU *et al.*, 2019.

Como apresentado em SILVA, 2023, o Detectron2 utiliza três blocos principais mostrados na Figura 3, o bloco *Backbone Network*, *Region Proposal Network* e *Box Head*.

Figura 3 - Arquitetura Detectron2.



Fonte: Adaptado de WU *et al.*, 2019.

O primeiro bloco, Backbone Network, extrai as características da imagem de entrada em diferentes escalas, já que objetos de interesse podem aparecer em diversos tamanhos na imagem. Para isso, utiliza-se uma arquitetura em pirâmide (como *ResNet* com *Feature Pyramid Network* - FPN) que processa a imagem em múltiplas resoluções espaciais: escalas altas capturam detalhes finos de objetos pequenos, enquanto escalas baixas capturam informações contextuais de objetos grandes. Tipicamente, são utilizadas 5 escalas diferentes (P2, P3, P4, P5, P6), onde cada nível possui uma resolução espacial diferente, variando de 1/4 até 1/64 da resolução original da imagem.

O bloco de *Region Proposal Network* (RPN) é amplamente utilizado em modelos de detecção em duas etapas e tem por finalidade gerar propostas de regiões onde os objetos podem estar localizados. Por padrão, são obtidas 1000 propostas durante o treinamento e 100 durante a inferência, cada uma com pontuações de confiança calculadas através de uma função sigmoide aplicada à saída de uma camada de classificação binária, que indica a probabilidade de cada região conter um objeto (*objectness score*). Essas pontuações variam entre 0 e 1, onde valores próximos a 1 indicam alta probabilidade de presença de objeto.

Por fim, o Box Head recebe as características extraídas pelo *backbone* e as

propostas de região do RPN para realizar a tarefa final, como a classificação do objeto em uma das classes predefinidas e o refinamento da localização da caixa delimitadora através de regressão das coordenadas, ou a geração de uma máscara de segmentação pixel a pixel para cada objeto detectado.

Desde seu lançamento, o Detectron2 tornou-se uma plataforma para a pesquisa em reconhecimento de objetos. Sua principal contribuição foi acelerar o ciclo de pesquisa, fornecendo uma base de código confiável e eficiente, juntamente com um "Model Zoo" (zoológico de modelos) - uma coleção organizada de modelos pré-treinados com diferentes arquiteturas, treinados em *dataset* padrão como COCO e *Open Images*, disponibilizados com seus pesos, configurações de treinamento e métricas de performance. Isso permite que pesquisadores comparem seus novos métodos com resultados de forma justa e reproduzível, eliminando a necessidade de reimplementar modelos de referência. O artigo original de (WU *et al.*, 2019) serve como a principal referência para o framework, detalhando sua filosofia de design e capacidades.

3.1.2 Métricas de Desempenho

Para entender se a CNN está efetuando a identificação e classificação de forma satisfatória para o objetivo da aplicação, é de suma importância o entendimento e obtenção de métricas de desempenho do sistema. Existem várias métricas para se avaliar um algoritmo de classificação, cada uma tendo enfoque em um comportamento e apresentando algumas limitações em relação a sua utilização.

3.1.2.1 Matriz de Confusão

Amplamente empregada em tarefas de classificação binária, a matriz de confusão permite a comparação entre os resultados preditos pelo modelo e os resultados observados na realidade. Sua importância reside na avaliação de algoritmos de classificação, fornecendo a base para a compreensão de métricas derivadas. A visualização do desempenho é feita pela quantificação dos verdadeiros positivos (TP), verdadeiros negativos (TN), falsos positivos (FP) e falsos negativos (FN), geralmente dispostos conforme apresentado na Figura 4.

Figura 4 - Matriz de confusão.

		Valor Predito	
		Positivo	Negativo
Valor Real	Positivo	Verdadeiro Positivo (TP)	Falso Negativo (FN)
	Negativo	Falso Positivo (FP)	Verdadeiro Negativo (TN)

Fonte: Autoria própria.

3.1.2.2 Recall

A sensibilidade ou do inglês *Recall* é a métrica que mede a capacidade do modelo de identificar corretamente todas as instâncias relevantes em uma imagem.

Utilizando as siglas apresentadas na matriz de confusão, a equação do *Recall* é mostrada na Equação (3).

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3}$$

Um alto valor de *Recall* indica que o modelo é eficaz em minimizar os Falsos Negativos. Em muitas aplicações de processamento de imagens, ter um alto Recall é de extrema importância. Por exemplo, em diagnóstico médico por imagem (como a detecção de tumores), é preferível ter um Recall elevado para não perder nenhum caso real, mesmo que isso signifique ter alguns Falsos Positivos que poderiam ser verificados posteriormente.

3.1.2.3 Precisão

Representando na Equação (4):

$$Precisão = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{4}$$

A precisão representa a proporção de verdadeiros positivos em relação a todos os positivos, falsos ou verdadeiros. Em outras palavras, informa a exatidão das detecções positivas.

Um valor alto de precisão significa que a maioria das classificações positivas estão corretas.

3.1.2.4 *F1-Score*

O F1-score é uma única métrica que combina precisão e recall, fornecendo uma medida equilibrada do desempenho de um modelo de classificação. Ele é calculado como a média harmônica da precisão e do recall. A fórmula para o F1-score é mostrada na Equação (5):

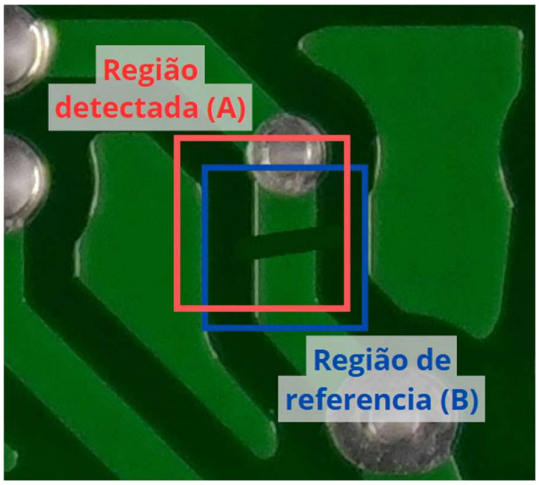
$$F1 - score = \frac{2 * \mathrm{TP}}{2 * \mathrm{TP} + \mathrm{FP} + \mathrm{FN}} \quad (5)$$

Ao avaliar modelos de classificação, especialmente em situações com classes desbalanceadas ou onde o equilíbrio entre identificar corretamente os positivos e evitar classificações positivas incorretas é crucial, o F1-score se destaca como uma métrica robusta e informativa.

3.1.2.5 *Intersection over Union* (IoU)

A métrica *Intersection over Union* (IoU), também conhecida como índice de Jaccard, desempenha um papel fundamental na avaliação de modelos em visão computacional, especialmente em tarefas como detecção de objetos e segmentação de imagens. Dado a região detectada pelo modelo, aqui chamada de A, e a região correspondente na anotação de referência do *dataset*, denominada B como mostrado na Figura 5, o cálculo da IoU é definido como a razão entre a intersecção e a união das áreas de A e B. Matematicamente, isso pode ser expresso pela Equação (6).

Figura 5 - Exemplo de região prevista e região de referência.



Fonte: Autoria própria.

$$IoU = \frac{|\mathrm{A} \cap \mathrm{B}|}{|\mathrm{A} \cup \mathrm{B}|} \tag{6}$$

A IoU fornece uma medida direta e intuitiva de quão precisamente um modelo está localizando objetos. Como dito em MORAIS, 2023 essa métrica pode ser utilizada para definir se uma detecção está correta (TP) ou não (FP ou FN) se for definido um valor de corte ou *treshold*, comumente de 0,5.

3.1.2.6 *Mean Average Precision* (mAP)

A mAP fornece uma avaliação da capacidade de uma rede neural de localizar e classificar corretamente objetos em todas as categorias presentes no conjunto de dados. Um modelo de detecção de objetos de alto desempenho precisa não apenas identificar a presença de diferentes objetos, mas também definir a região da imagem que se encontram esses objetos. A mAP associa esses dois aspectos em uma única métrica, tornando-o uma medida robusta do desempenho geral.

Dito isso, a mAP integra as métricas de Precisão e *Recall* para avaliar o desempenho do modelo de detecção no contexto de múltiplas classes e limiares de confiança. A determinação de uma detecção correta, ou Verdadeiro Positivo (TP), é comumente definida pela sobreposição espacial entre a região prevista e a região verdadeira, utilizando um limiar mínimo de IoU. Detecções que não correspondem a

nenhuma região verdadeira (ou não atingem o limiar de IoU) são consideradas Falsos Positivos (FP), enquanto regiões verdadeiras não detectadas pelo modelo constituem Falsos Negativos (FN).

Uma das grandes vantagens da mAP reside na sua capacidade de lidar eficientemente com cenários de multi-classe, nos quais o modelo deve identificar e classificar múltiplos tipos distintos de objetos simultaneamente. Em vez de uma única métrica global, a mAP calcula a métrica de Precisão Média (AP) para cada classe individualmente. Posteriormente, o mAP é obtido calculando a média dessas APs sobre todas as classes detectáveis. Essa abordagem por classe a torna intrinsecamente adequada para lidar com o desequilíbrio de classes, uma condição comum em conjuntos de dados de detecção onde a frequência de ocorrência de diferentes objetos varia significativamente. Ao avaliar o desempenho para cada classe separadamente antes de agregar, a mAP oferece uma medida mais justa do desempenho geral, não sendo excessivamente influenciada pelo desempenho apenas nas classes mais abundantes. Matematicamente, o mAP é expresso na equação (7).

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{7}$$

Sendo AP_i a precisão média calculada por cada classe e N o número total de classes.

3.1.2.7 *Loss*

A função de perda, do inglês *Loss*, é uma métrica fundamental que quantifica a diferença entre as predições do modelo e os valores reais esperados durante o processo de treinamento. Ela serve como indicador da qualidade das predições e orienta o algoritmo de otimização na busca pelos melhores parâmetros do modelo.

No contexto de detecção de objetos, a função de perda é tipicamente composta por múltiplos componentes que avaliam diferentes aspectos da predição:

- ***Loss* de Classificação**: Mede a precisão na identificação das classes dos objetos detectados, utilizando funções como a entropia cruzada para quantificar o erro entre as probabilidades preditas e as classes

verdadeiras.

- ***Loss* de Localização**: Avalia a precisão na localização espacial dos objetos, comparando as coordenadas das caixas delimitadoras (bounding boxes) preditas com as coordenadas reais dos objetos.
- ***Loss* de Confiança**: Em alguns modelos, mede a confiança do modelo em suas predições, penalizando predições com baixa confiança para objetos reais ou alta confiança para detecções falsas.

A função de perda total é geralmente expressa como uma combinação ponderada destes componentes, conforme apresentado na Equação (8) .

$$Loss_{total} = \alpha * Loss_{classificação} + \beta * Loss_{localização} + \gamma * Loss_{confiança} \quad (8)$$

Onde α, β e γ são hiper parâmetros que controlam a importância relativa de cada componente.

Um valor baixo de *Loss* indica que o modelo está fazendo predições próximas aos valores reais, enquanto valores altos sugerem que o modelo ainda precisa de ajustes. Durante o treinamento, o objetivo é minimizar esta função através de algoritmos de otimização como o gradiente descendente, permitindo que o modelo aprenda progressivamente a realizar detecções mais precisas.

3.2 PLACA DE CIRCUITO IMPRESSO

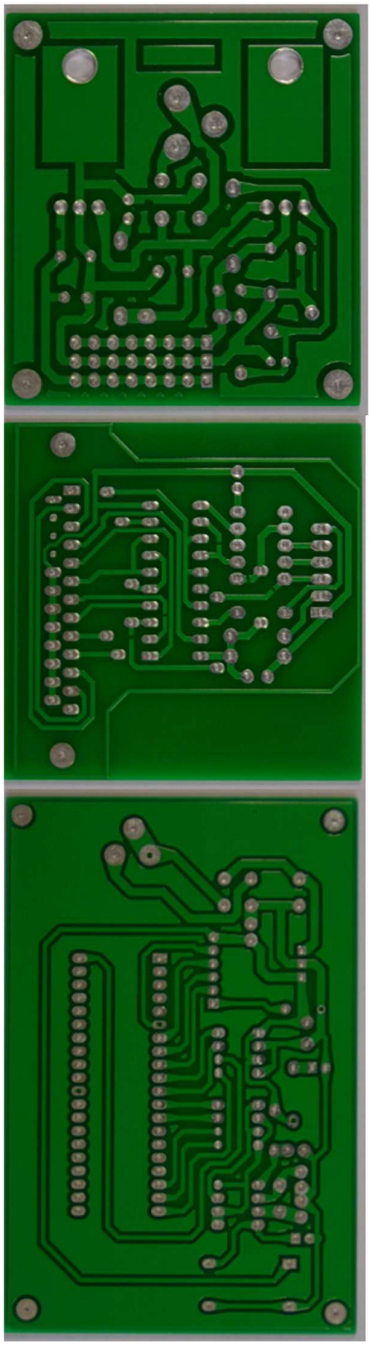
Uma PCI é constituída de dois principais componentes, uma base de material isolante e uma lâmina condutora, normalmente de cobre. A combinação de trilhas com presença ou ausência do material condutor é o que define os contatos e faz a conexão entre os elementos do sistema.

Apesar de amplamente utilizada e desenvolvida de diferentes formas, a PCI está sujeita a apresentar falhas, seja falhas de confecção ou proveniente de má instalação dos componentes, normalmente soldados, ou ainda por problemas de manuseio.

As falhas de confecção geralmente são ocasionadas por falta ou excesso de material condutor, bem como a ausência de furos para a fixação dos componentes na PCI.

A Figura 6 apresenta possíveis falhas geradas na confecção de uma PCI, esse tipo de falha será o objeto de interesse do presente trabalho e será explicado no próximo capítulo.

Figura 6 - Exemplo de PCI com falha.



Fonte: HUANG; WEI, 2019.

4 MATERIAIS E MÉTODOS

Este capítulo é destinado a mostrar como foram feitos os treinamentos das redes YOLOv11 e Detectron2 para detecção de falhas em PCIs.

4.1 RECURSOS COMPUTACIONAIS

O desenvolvimento dos algoritmos e treinamentos dos modelos foram feitos no ambiente de desenvolvimento Google *Colaboratory* ou *Colab*, que permite executar códigos em Python em estruturas de células de código.

O Google Colab permite executar códigos de forma online sem necessidade de configuração de um ambiente local e permite a utilização de *hardware* potente e integração com o Google Drive. Na versão Pro do Google Colab, é possível utilizar recursos avançados como GPU's de última geração para acelerar a compilação e simulação dos algoritmos necessários. Na Tabela 1 é possível ver os recursos utilizados nos treinamentos desse projeto. Essa tabela foi gerada pelo comando *nvidia-smi* que fornece informações detalhadas sobre placas de vídeo NVIDIA instaladas no sistema.

A primeira linha da tabela mostra a data de geração da mesma, o cabeçalho principal conta com as informações do driver NVIDIA instalado, nesse caso NVIDIA-SMI 550.54.15, a versão do drive, 550.54.15 e a versão do CUDA *Toolkit* suportada pelo driver, 12.4.

As próximas linhas são as identificações da GPU, contendo o índice da placa de vídeo, 0, o modelo da placa, NVIDIA A100-SXM4-40GB, além de informações adicionais como temperatura, 29ºC, estado de performance, máxima performance (indicada por P0), consumo de energia, consumo atual de 42 watts do limite máximo de 400 watts entre outros.

Tabela 1 - GPU utilizada no treinamento.

```
Thu May  8 00:03:20 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-SXM4-40GB          Off |   00000000:00:04.0 Off |                    0 |
| N/A   29C    P0             42W /  400W |       0MiB /  40960MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

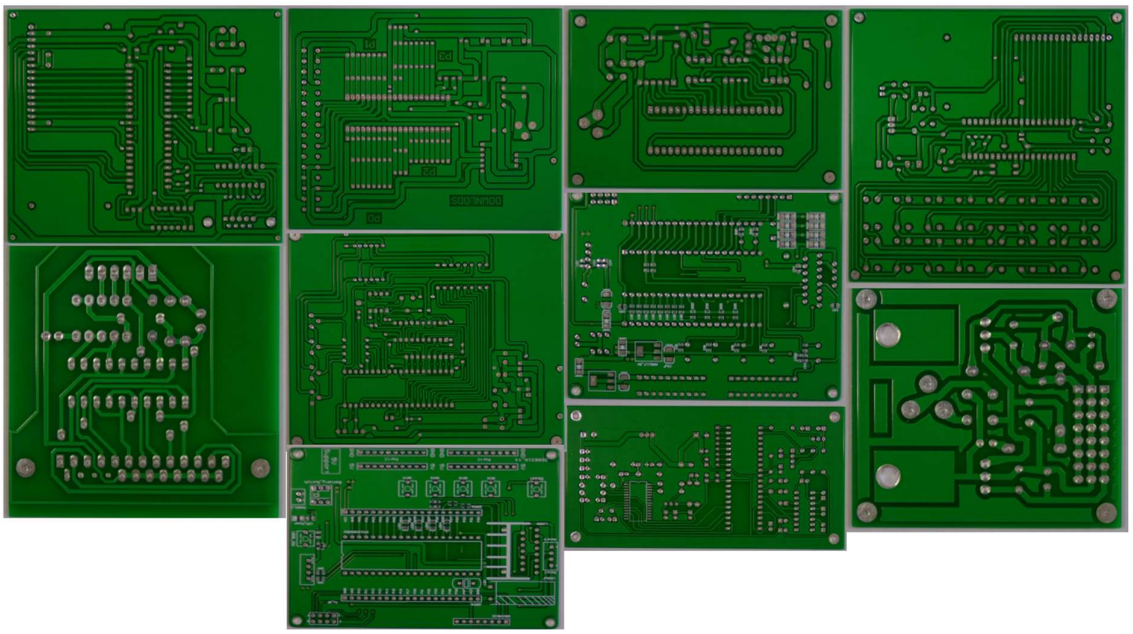
Fonte: Autoria própria.

4.2 BASE DE DADOS

A base de dados escolhida para este trabalho é a HRIPCB, proposta por HUANG; WEI, 2019 e utilizada em alguns dos trabalhos descritos no capítulo 2.

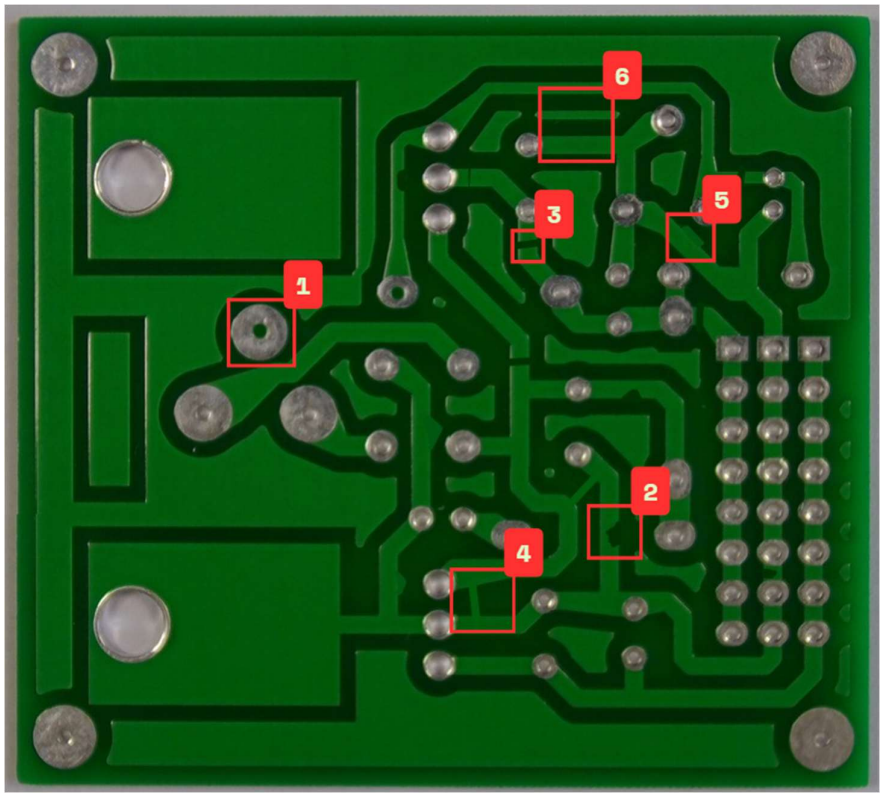
Essa base foi constituída através da captura de imagem de doze placas sem defeito utilizando uma câmera industrial com dezesseis megapixel de resolução Figura 7 e posteriormente adicionado 6 tipos de defeito de forma manual em um *software* de edição de imagens. Cada imagem pode apresentar de 2 a 6 tipos de defeitos simultaneamente como explicado em HUANG; WEI, 2019.

Figura 7 - PCI's base do *dataset*.



Fonte: HRIPCB

Figura 8 - Defeitos adicionados em *software* de edição.



Fonte: HRIPCB.

Um exemplo de cada um dos defeitos pode ser observado na Figura 8 e são eles:

1. *Missing Hol*e: Furo ausente.
2. *Mouse Bite*: Pequenas falhas nas trilhas de cobre.
3. *Open Circuit*: Interrupção indesejada na continuidade de uma trilha (circuito aberto).
4. *Short*: Junção indesejada de 2 trilhas distintas (curto-circuito).
5. *Spur*: Excesso de cobre em uma parte de uma trilha.
6. *Spurious Cooper:* Pedaços de trilhas isoladas (desconectadas do resto do circuito).

A distribuição desses defeitos na base de dados é mostrada na Tabela 2.

Tabela 2 - Distribuição dos defeitos na base de dados.

Tipo do Defeito	**Número de Imagens**	**Quantidade Total**
*Missing Hol*e	115	497
Mouse Bite	115	492
Open Circuit	116	482
Short	116	491
Spur	115	488
Spurious Cooper	116	503
Total	693	2953

Fonte: HUANG; WEI, 2019.

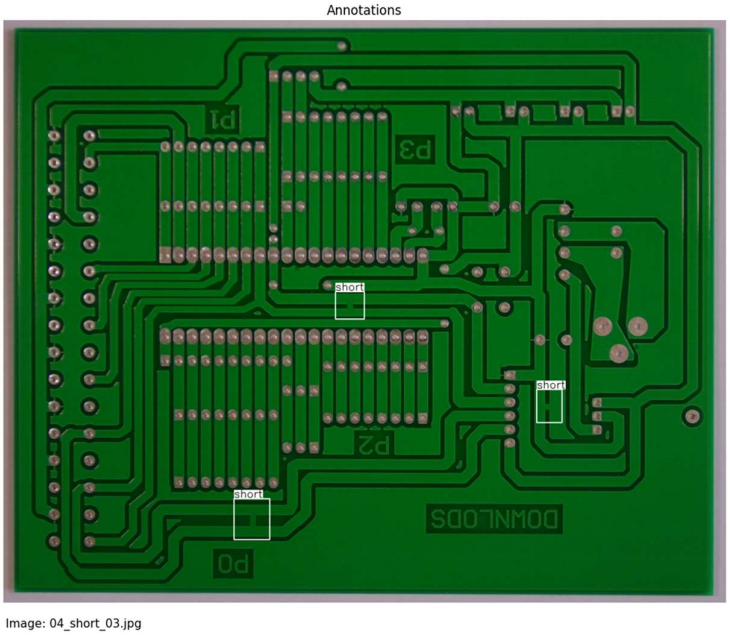
Além das imagens, a base de dados conta com arquivos de anotações dos defeitos. Os arquivos estão no formato .xml e contém informações de qual imagem ele referência, onde a imagem está salva, a localização cartesiana em pixels dos defeitos da imagem e o tipo do defeito. Um desses arquivos de anotação é mostrado na Figura 9 e corresponde as marcações das *bounding boxes* da Figura 10.

Figura 9 - Arquivo de anotação xml.

```xml
<annotation>
  <folder>Short</folder>
  <filename>04_short_03.jpg</filename>
  <path>/home/weapon/Desktop/PCB_DATASET/Short/04_short_03.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>3056</width>
    <height>2464</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>short</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>2255</xmin>
      <ymin>1562</ymin>
      <xmax>2362</xmax>
      <ymax>1702</ymax>
    </bndbox>
  </object>
  <object>
    <name>short</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>1404</xmin>
      <ymin>1148</ymin>
      <xmax>1526</xmax>
      <ymax>1264</ymax>
    </bndbox>
  </object>
  <object>
    <name>short</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>975</xmin>
      <ymin>2024</ymin>
      <xmax>1126</xmax>
      <ymax>2197</ymax>
    </bndbox>
  </object>
</annotation>
```

Fonte: HRIPCB.

Figura 10 - Placa com defeitos.



Fonte: Adaptado de HRIPCB.

Alguns arquivos presentes na base de dados não serão utilizados nesse trabalho. A estrutura das pastas contendo arquivos de interesse é mostrada na Figura 11.

Figura 11 - Estrutura de pastas da base de dados.

```
PCB_DATASET/
│
├── images/
│   ├── Mouse_bite/
│   │   ├── ├── 01_mouse_bite_01.jpg
│   │   ├── ├── 01_mouse_bite_02.jpg
│   │   ├── └── ... (+113 arquivos)
│   ├── Missing_hole/
│   │   ├── ├── 01_missing_hole_01.jpg
│   │   ├── ├── 01_missing_hole_02.jpg
│   │   ├── └── ... (+113 arquivos)
│   ├── Open_circuit/
│   │   ├── ├── 01_open_circuit_01.jpg
│   │   ├── ├── 01_open_circuit_02.jpg
│   │   ├── └── ... (+114 arquivos)
│   ...
│
├── Annotations/
│   ├── Mouse_bite/
│   │   ├── ├── 01_mouse_bite_01.xml
│   │   ├── ├── 01_mouse_bite_02.xml
│   │   ├── └── ... (+113 arquivos)
│   ├── Open_circuit/
│   │   ├── ├── 01_open_circuit_01.xml
│   │   ├── ├── 01_open_circuit_02.xml
│   │   ├── └── ... (+114 arquivos)
│   ├── Missing_hole/
│   │   ├── ├── 01_missing_hole_01.xml
│   │   ├── ├── 01_missing_hole_02.xml
│   │   ├── └── ... (+113 arquivos)
│   ...
```

Fonte: Autoria própria.

4.3 TREINAMENTO DA YOLOv11

Antes de iniciar o treinamento do YOLO uma etapa de pré-processamento se faz necessária. Essa etapa se inicia com a adequação das anotações da base de dados, inicialmente em XML para arquivos TXT organizados em um padrão específico de pastas.

Para isso foi desenvolvido uma função no código que percorre os arquivos XML e extrai as informações de dimensões da imagem (altura e largura), coordenadas das *bounding boxes* (x_{min}, y_{min}, x_{max}, y_{max}) e a classe do defeito.

O YOLOv11 requer que todos os arquivos TXT estejam em uma pasta e todas as imagens em outra, cada imagem tem um arquivo TXT com o mesmo nome contendo em cada linha as informações de uma *bounding box*. As informações são o tipo de defeito, que precisa ser um número inteiro, e os valores normalizados das coordenadas x y do centro da *bounding box* e os valores da sua largura e altura.

Os defeitos foram enumerados da forma apresentada na Tabela 3.

Tabela 3 - Definição numérica do tipo de defeito.

Valor Numérico	**Tipo de Defeito**
0	*missing_hole*
1	*mouse_bite*
2	*open_circuit*
3	*short*
4	*spur*
5	*spurious_copper*

Fonte: Autoria própria.

Para a normalização das coordenadas e dimensões se fez o uso das Equações (9), (10), (11) e (12).

$$x_c = \frac{x_{min} + x_{max}}{2 * largura} \tag{9}$$

$$y_c = \frac{y_{min} + y_{max}}{2 * altura} \tag{10}$$

$$bounding\_box\ _{largura} = \frac{x_{min} + x_{max}}{largura} \quad (11)$$

$$bounding\_box_{altura} = \frac{y_{min} + y_{max}}{altura} \quad (12)$$

Com os dados normalizados e o valor numérico dos defeitos foi montado para cada imagem um arquivo TXT como o mostrado na Figura 12.

Figura 12 - Arquivo de anotação no padrão usado pelo YOLO.

```
0 0.84765625 0.1140625 0.0234375 0.025
0 0.34375 0.51328125 0.021875 0.0234375
0 0.45859375 0.41015625 0.0203125 0.0265625
0 0.83046875 0.43671875 0.0203125 0.0171875
0 0.5796875 0.534375 0.025 0.025
```

Fonte: Autoria própria.

Para padronizar a base de dados e otimizar o treinamento, todas as imagens foram redimensionadas para ter 640x640 pixels, que é o tamanho padrão para o YOLOv11. Os arquivos com as anotações mostradas dentro das imagens também foram alterados para que as anotações sejam mostradas em tamanho adequado ao novo tamanho dos *bounding boxes*.

Após essa etapa, as imagens e anotações foram divididas nos grupos de treinamento, validação e teste e para cada grupo foi criada uma pasta contendo as imagens e outra contendo as anotações do grupo.

Além disso, para uma avaliação mais robusta, foi implementado no código a validação cruzada *k-fold* com k igual a 3. VELOSO, 2022 explica que esta técnica divide a base de dados em k subconjuntos, onde cada subconjuntos serve como conjunto de validação uma vez, enquanto os demais são usados para treinamento.

Para cada divisão da base, são criadas estruturas de diretórios específicas e arquivos YAML de configuração. Esse arquivo contém o caminho para os arquivos de anotação da divisão, o nome da pasta de treinamento, o nome da pasta de validação e o nome das classes, um exemplo desse arquivo é mostrado na Figura 13.

Figura 13 - YAML de configuração do treinamento do YOLOv11.

```
yaml:
'path': '/content/drive/MyDrive/PCB_DATASET/output/3fold_crossval/split_1'
'train': 'train'
'val': 'val'
'names': ['missing_hole', 'mouse_bite','open_circuit','short','spur','spurious_copper']
```

Fonte: Autoria própria.

O treinamento utilizou o modelo pré-treinado YOLOv11s (versão *small*) como ponto de partida. Esse modelo já vem com uma base de conhecimento em detecção de objetos adquirido pelo treinamento no conjunto de dados COCO e terá esse conhecimento adaptado para detecção de defeitos em PCI através do processo de *fine-tuning*.

Além dos arquivos YAML, alguns parâmetros foram definidos no código de treinamento e apresentados na Figura 14. O tamanho do *batch* define quantas imagens são processadas simultaneamente antes de atualizar os pesos da rede neural, *epochs* é o número de vezes que todo o *dataset* de treinamento passa pela rede neural, *imgsz* é o tamanho da imagem, lr0 (*Learning Rate Start*) é a taxa de aprendizado no início do treinamento, de forma gradual a taxa diminui até alcançar o lr1 (*Learning Rate End*), *mixup* representa a probabilidade da rede neural de aplicar técnica de *data augmentation*, que consiste em criar imagens híbridas a partir da junção pedaços de outras imagens, esse tipo de técnica ajuda a generalizar melhor o aprendizado, principalmente em base de dados com poucas imagens e o *save_period* define a frequência de salvamento do modelo.

Figura 14 - Parâmetros de treinamento.

```
batch = 16          # Tamanho do batch
epochs = 180        # Número de épocas
imgsz = 640         # Tamanho da imagem
lr0 = 0.001         # Taxa de aprendizado inicial
lrf = 0.0001        # Taxa de aprendizado final
mixup = 0.3         # Técnica de data augmentation
save_period = 1     # Período de salvamento do modelo
```

Fonte: Autoria própria.

O treinamento foi executado para cada uma das divisões da base de dados e ao final mostra as métricas de treinamento, além de salvar esses valores em um arquivo CSV.

Após o treinamento, o melhor modelo é carregado e testado com as imagens

separadas na pasta *test*.

4.4 TREINAMENTO DA DETECTRON2

Da mesma forma que o YOLO, para o Detectron2 também se fez necessário uma etapa de pré-processamento para adequar a base de dados aos formatos exigidos por essa topologia.

Primeiramente a base de dados foi dividida em 2 grupos, treino e validação, na proporção 80/20. Com isso foram separadas 554 imagens para treinamento e 139 para validação. Essa divisão foi feita de forma aleatória para garantir a representatividade estatística em ambos os conjuntos.

Após a divisão, todas as imagens foram redimensionadas para o tamanho de 640x640 pixels, essa etapa é essencial para padronizar o tamanho de entrada da rede neural, otimizar o uso de memória GPU e manter a consistência no processamento em lotes (*batches*).

Com as imagens redimensionadas, foi necessário adequar o formato das anotações, que no Detectron2 precisa estar no formato COCO e por isso exigiu uma reestruturação das informações presentes nos arquivos XML. O formato COCO organiza as anotações em quatro seções principais:

- **Info:** Metadados do *dataset*.
- **Categorias:** Definição das classes de objetos
- **Imagens:** Informações das imagens (ID, nome, dimensões)
- ***Annotations:*** Anotações individuais com referências às imagens e categorias.

Uma grande diferença entre os dados XML e o padrão COCO é a representação das *bouding boxes*. Enquanto os arquivos XML contêm as coordenadas x_{min}, y_{min}, x_{max}, y_{max} (pontos diagonalmente opostos) o formato COCO precisa das informações x, y, largura, altura (canto superior esquerdo e dimensões). A conversão entre os formatos foi simples e seguiu o mostrando nas Equações (13), (14), (15) e (16).

$$x = x_{min} \tag{13}$$

$$y = y_{min} \tag{14}$$

$$largura = x_{max} - x_{min} \tag{15}$$

$$altura = y_{max} - y_{min} \tag{16}$$

Além disso, como as imagens foram redimensionadas, as coordenadas das *bounding boxes* também foram ajustadas. Ao final desse processo obteve-se um arquivo JSON contendo as informações corretas das imagens do conjunto de treinamento e validação, um exemplo do corpo desse arquivo está na Figura 15.

Figura 15 - Anotações da base de dados no padrão COCO.


```
{
  "info": {
    "description": "PCB Defect Dataset",
    "url": "",
    "version": "1.0",
    "year": 2025,
    "contributor": "",
    "date_created": "2025-04-18 20:25:51"
  },
  "licenses": [],
  "images": [
    {
      "id": 1,
      "file_name": "09_spurious_copper_03.jpg",
      "width": 640,
      "height": 640
    },
    ⋮
  ],
  "annotations": [
    {
      "id": 1,
      "image_id": 1,
      "category_id": 5,
      "bbox": [
        318,
        125,
        12,
        34
      ],
      "area": 408,
      "iscrowd": 0
    },
      ⋮
```


Fonte: Autoria própria.

Para o treinamento do modelo houve a necessidade de configurar alguns hiper parâmetros mostrados na Figura 16.

Figura 16 - Hiper parâmetros para treinamento da Detectron2.

```
cfg.SOLVER.IMS_PER_BATCH = 2          # Tamanho do batch
cfg.SOLVER.BASE_LR = 0.00025          # Taxa de aprendizado
cfg.SOLVER.MAX_ITER = 10000           # Número máximo de iterações
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 6   # Número de classes (defeitos)
```

Fonte: Autoria própria.

Seguindo a mesma ideia do YOLO, para o Detectron2 também foi usado um modelo pré-treinado no *dataset* COCO, o Detectron/faster_rcnn_R_50_FPN_3x. Isso permite o treinamento de convergir mais rapidamente, melhora o desempenho em base de dados menores e também aproveita as características visuais gerais já aprendidas anteriormente.

Uma classe de treinamento personalizada que entende a classe padrão do Detectron2 foi utilizada para incorporar avaliações automáticas ao longo do treinamento, e o salvamento automático dos melhores modelos.

Ao final do treinamento o melhor modelo é salvo no *Drive* e também exporta um arquivo JSON com dados do treinamento e métricas de validação.

5 RESULTADOS

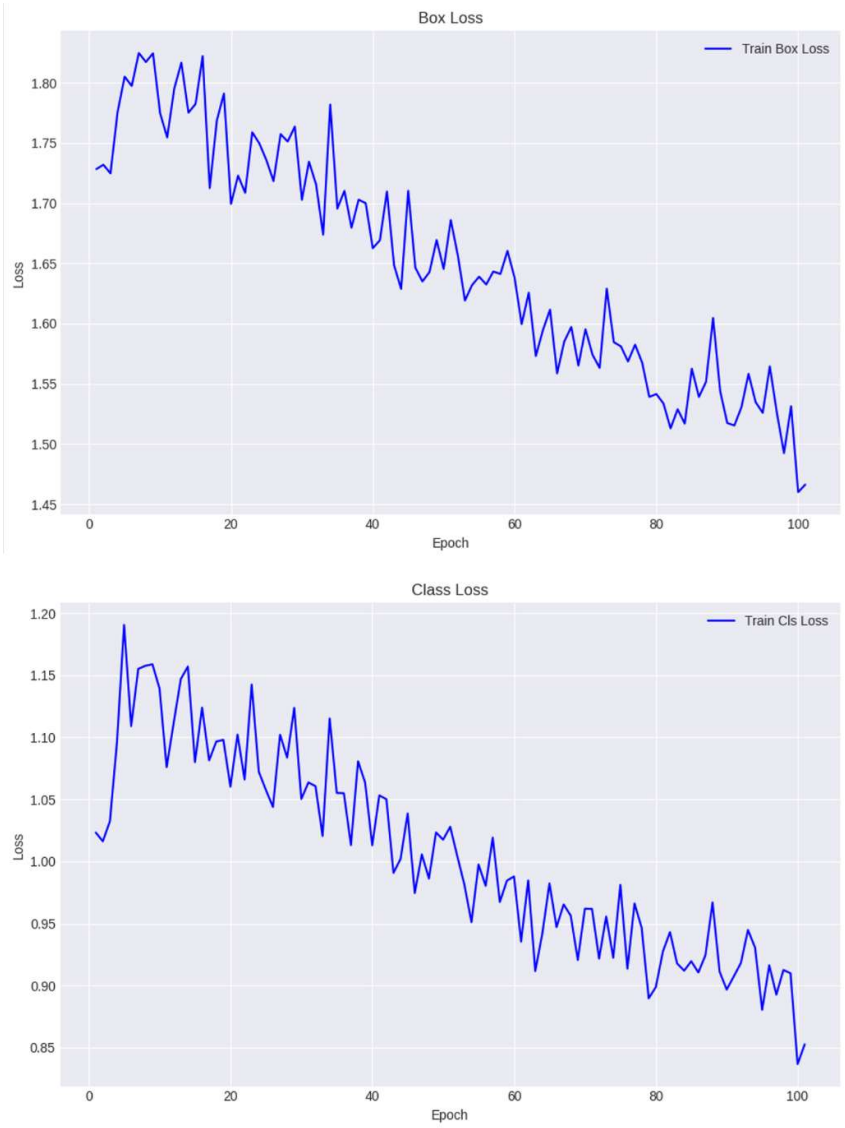
5.1 ANÁLISE DO COMPORTAMENTO DE TREINAMENTO

A análise comparativa do comportamento de treinamento entre YOLOv11 e Detectron2 revela características distintas em termos de convergência e estabilidade das funções de perda.

Para o YOLOv11, o aprendizado da rede neural foi divido em 3 treinamentos, cada treinamento herdava o modelo treinado pelo anterior, tendo o primeiro treinamento 180 épocas. Os demais treinamentos foram interrompidos na centésima época de forma automática pois após 100 épocas não apresentaram uma melhora significativa no mAP_{50}, esse comportamento é chamado de paciência ou early *stopping* e serve para evitar o *overfitting*, onde o modelo aprende os dados de treinamento de forma tão precisa que perde a capacidade de generalizar para novos dados. Os dados apresentados do YOLOv11 refletem o ultimo treinamento realizado.

O YOLOv11 apresentou um processo de aprendizado gradual e consistente ao longo de 100 épocas, com a função de perda para localização de caixas delimitadoras (*box loss*) iniciando em aproximadamente 1,8 e finalizando em valores próximos a 1,5 e a perda para classificação dos objetos (*class loss*) tendo um pico de 1,19 nas primeiras épocas e caindo até 0,85 como mostrando na Figura 17. Este comportamento indica um ajuste progressivo dos parâmetros do modelo, típico de arquiteturas YOLO que utilizam otimizadores adaptativos.

Figura 17 - Gráficos de *Loss* no treinamento do YOLOv11.



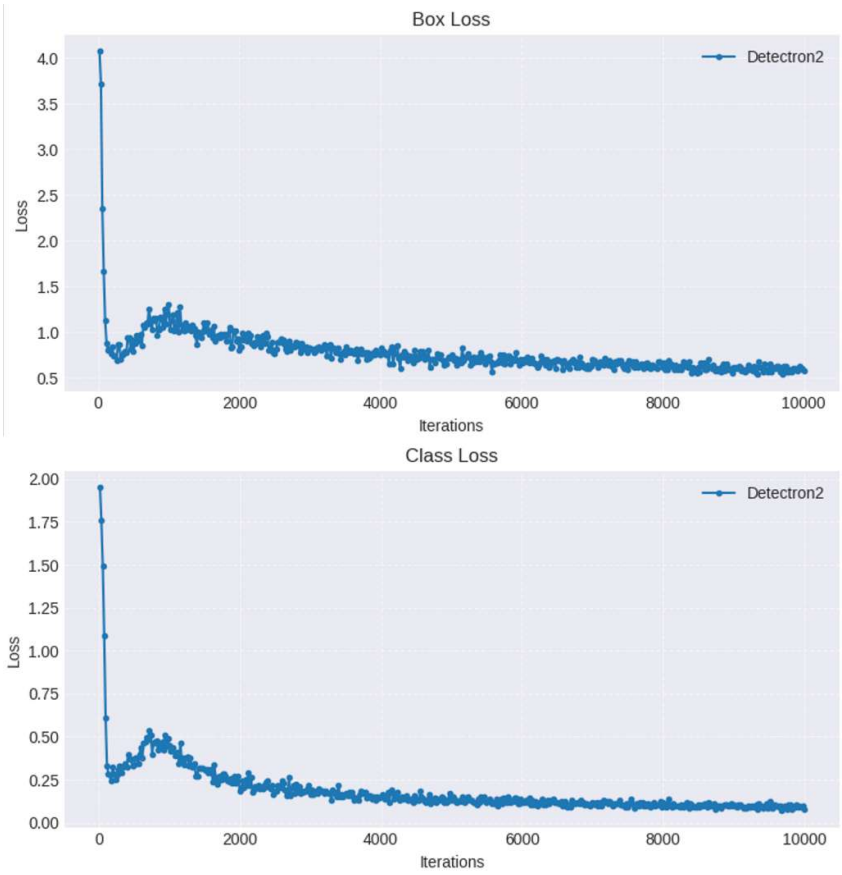
Fonte: Autoria própria.

É valido ressaltar que apesar da diminuição progressiva dos valores de *box loss* e *class loss*, não houve melhoras no mAP_{50} ao longo do último treinamento e por isso houve o early *stopping.*

Em contraste, o Detectron2 não apresentou early *stopping e* demonstrou uma convergência mais agressiva nas primeiras 2000 iterações, com a perda de localização de caixas delimitadoras estabilizando em valores entre 0,6 e 0,7. A perda de classificação apresentou uma redução dramática de aproximadamente 2,0 para valores próximos a 0,1, evidenciando a eficiência do processo de otimização da arquitetura Faster R-CNN e exibida na Figura 18. Esta convergência rápida pode ser

atribuída à estrutura de treinamento em duas etapas do Detectron2, onde a *Region Proposal Network* (RPN) e o classificador final são otimizados de forma coordenada.

Figura 18 - Gráficos de *Loss* no treinamento do Detectron2.



Fonte: Autoria própria.

A estabilidade do treinamento representa um aspecto crucial na comparação entre os modelos. O YOLOv11 exibiu oscilações moderadas durante todo o processo de treinamento, comportamento esperado devido à natureza do algoritmo de otimização e à complexidade da tarefa de detecção multi-classes.

Por outro lado, o Detectron2 apresentou maior suavidade nas curvas de perda após as primeiras iterações. Esta estabilidade superior pode ser vantajosa em cenários onde a previsibilidade do processo de treinamento é fundamental, especialmente em aplicações industriais onde a consistência dos resultados é prioritária.

5.2 AVALIAÇÃO DAS MÉTRICAS DE DESEMPENHO

As métricas de desempenho obtidas durante o treinamento fornecem insights valiosos sobre a eficácia de cada modelo na tarefa de detecção de defeitos em placas de circuito impresso. O YOLOv11 apresentou resultados particularmente notáveis, com mAP_{50} de 0,98. Este valor indica que o modelo mantém alta precisão quando avaliado com *threshold* de IoU de 0,5, padrão amplamente aceito na comunidade de visão computacional. Entretanto, o mAP_{50-95}, que considera thresholds mais rigorosos, apresentou o de 0,57, sugerindo que há margem para melhorias na precisão de localização e classificação em critérios mais exigentes.

O Detectron2 ficou um pouco abaixo em relação ao YOLO, tendo os valores finais de mAP_{50} e mAP_{50-95} sendo 0,92 e 0,41 respectivamente.

5.3 ANÁLISE COMPARATIVA DA EFICIÊNCIA DE TREINAMENTO

A eficiência de treinamento constitui um fator determinante na escolha entre as arquiteturas, especialmente considerando os recursos computacionais disponíveis e os prazos de desenvolvimento. O YOLOv11 necessitou de aproximadamente 50 a 70 épocas para atingir convergência satisfatória, processo caracterizado por melhoria gradual e consistente das métricas. Esta progressão mais lenta pode ser vista como vantajosa em termos de estabilidade, permitindo ajustes finos durante o processo de treinamento.

O Detectron2 demonstrou convergência significativamente mais rápida, com estabilização das principais métricas ocorrendo em torno de 2000 iterações. Esta eficiência superior pode resultar em economia substancial de tempo de treinamento e recursos computacionais, aspecto particularmente relevante em ambientes de desenvolvimento com limitações de hardware ou prazos apertados.

Tabela 4 - Comparação de eficiência de treinamento.

Aspecto	YOLOv11	Detectron2
Convergência	60-70 épocas	~2000 iterações
Estabilidade	Oscilações moderadas	Alta estabilidade
Tempo até convergência	Gradual (mais lento)	Rápido
Previsibilidade	Moderada	Alta

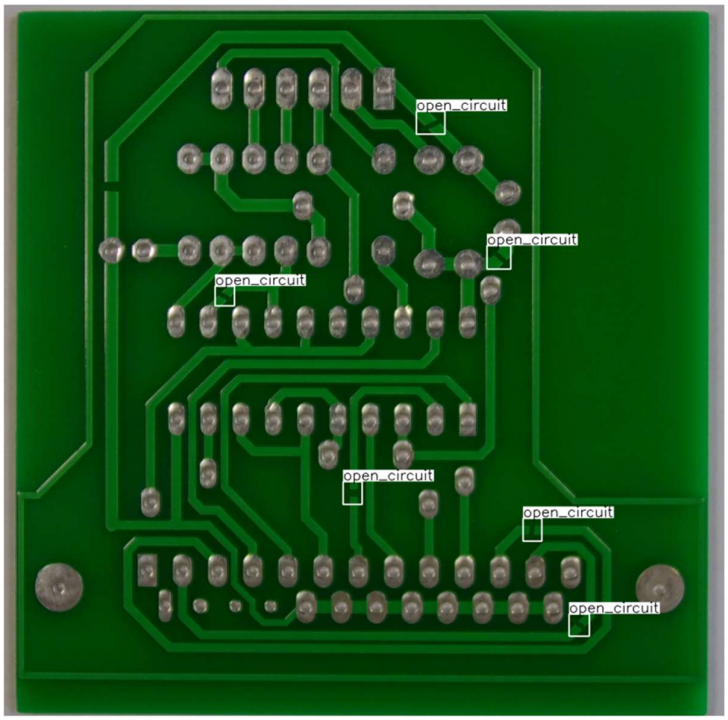
Fonte: Autoria própria.

A análise da estabilidade da Tabela 4 revela que o Detectron2 oferece maior previsibilidade no processo de treinamento, com menor sensibilidade a variações nos dados de entrada. Esta característica pode ser crucial em aplicações industriais onde a reprodutibilidade dos resultados é fundamental para a validação e certificação dos sistemas de detecção de defeitos.

5.4 ANÁLISE QUALITATIVA DOS RESULTADOS

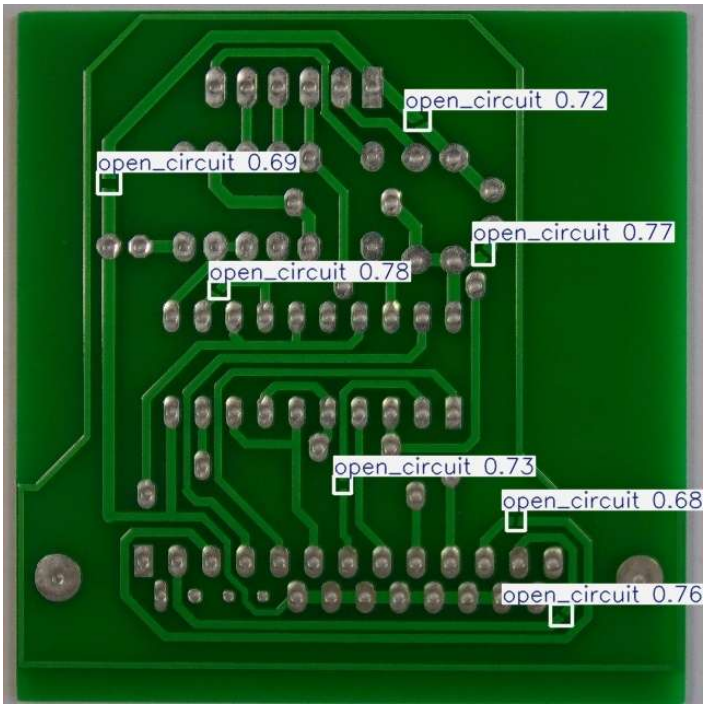
A análise qualitativa dos resultados obtidos revela aspectos fundamentais sobre a adequação de cada modelo para aplicações práticas de detecção de defeitos em placas de circuito impresso. O YOLOv11 demonstrou robustez considerável durante o processo de treinamento, mantendo tendência consistente de melhoria. Esta característica sugere que o modelo possui boa capacidade de adaptação a variações nos dados de treinamento, aspecto importante considerando a diversidade de defeitos que podem ocorrer em ambientes industriais reais. A Figura 19 mostra um PCI do *dataset* e suas anotações e na Figura 20 tem-se as detecções e classificações feitas pela YOLOv11.

Figura 19 - PCI do *dataset* com anotações.



Fonte: Adaptado de HRIPCB

Figura 20 - Detecções do YOLOv11.

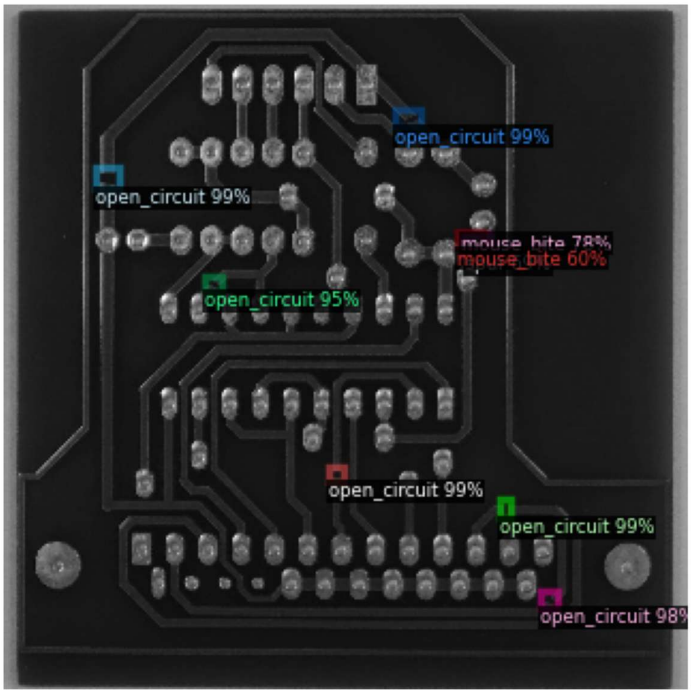


Fonte: Autoria própria.

O Detectron2, por sua vez, apresentou alguns erros de predições, comportamento esperado dado o fato de ter mAP menores que os valores da YOLO.

A Figura 21 apresenta as predições na mesma PCI da Figura 19. É possível observar que um dos defeitos da classe *open_circuit* foi classificado errado como *mouse_bite*, porém todos os defeitos classificados certos apresentam um valor de confiança igual ou acima de 95%, esse comportamento está de acordo com os valores reduzidos de *loss* apresentados no treinamento.

Figura 21 - Detecções do Detectron2.



Fonte: Autoria própria.

A adequação específica para defeitos em PCI pode ser avaliada através da análise do comportamento das diferentes componentes de perda. No caso do YOLOv11, a evolução equilibrada entre *box loss* e *class loss* indica que o modelo consegue simultaneamente aprender a localizar precisamente os defeitos e classificá-los corretamente. O Detectron2, através das suas perdas, demonstra capacidade similar, mas com maior especialização na confiança devido à arquitetura de duas etapas.

6 CONCLUSÕES/CONSIDERAÇÕES FINAIS

Os resultados obtidos nesta investigação fornecem evidências substanciais sobre as características distintivas de cada modelo na tarefa de detecção de defeitos em placas de circuito impresso. O Detectron2 emerge como a opção preferencial quando a prioridade é a estabilidade e previsibilidade do processo de treinamento, oferecendo convergência mais rápida e menor variabilidade nas métricas. Esta característica pode ser decisiva em ambientes industriais onde a consistência e a reprodutibilidade dos resultados são fundamentais para a validação e implementação dos sistemas de detecção.

Por outro lado, o YOLOv11 apresenta vantagens competitivas em termos de métricas finais de desempenho, com valores de mAP_{50} superiores a 0,98, indicando excelente capacidade de detecção. A arquitetura YOLO também oferece vantagens em termos de simplicidade de implementação e interpretabilidade dos resultados, aspectos que podem ser relevantes em cenários onde a compreensão detalhada do comportamento do modelo é necessária.

A Tabela 5 mostra os valores finais das principais métricas de treinamento. Entretanto, a escolha final entre YOLOv11 e Detectron2 deve considerar não apenas as métricas de desempenho, mas também fatores como recursos computacionais disponíveis, requisitos de tempo real, facilidade de integração com sistemas existentes e necessidades específicas de manutenção e atualização do modelo. Os resultados presentes na Tabela 6 indicam que ambas as arquiteturas são tecnicamente viáveis para a aplicação proposta, com compensações específicos que devem ser avaliados no contexto particular de cada implementação industrial.

Tabela 5 - Métricas finais.

Critério	YOLOv11 (final)	Detectron2 (final)
Box_loss	1,47	0,6
Class_loss	0,85	0,15
mAP_{50}	0,98	0,92
$mAP_{59\text{-}90}$	0,57	0,41

Fonte: Autoria própria.

Tabela 6 - Considerações finais.

Critério	YOLOv11	Detectron2	Recomendação
Estabilidade de Treinamento	Moderada	Alta	Detectron2
Métricas Finais	Excelentes	Boas	YOLOv11
Velocidade de Convergência	Moderada	Alta	Detectron2
Simplicidade de Implementação	Alta	Moderada	YOLOv11
Adequação para PCI	Excelente	Excelente	Empate

Fonte: Autoria própria.

Os resultados obtidos neste trabalho indicam a necessidade de investigações mais aprofundadas utilizando *dataset* mais amplos e diversificados. Trabalhos futuros devem focar na criação de dataset mais abrangentes que incluam diferentes tipos de PCI's, desde placas simples de camada única até placas multicamadas complexas utilizadas em aplicações automotivas, aeroespaciais e de telecomunicações. A expansão do *dataset* deve contemplar não apenas uma maior quantidade de amostras, mas também uma diversificação nas condições de imageamento, incluindo diferentes ângulos de captura, condições de iluminação, resoluções de imagem e backgrounds. Esta diversificação é fundamental para avaliar a robustez real dos modelos em ambientes industriais, onde as condições de operação podem variar significativamente ao longo do tempo e entre diferentes linhas de produção.

Uma linha promissora de pesquisa envolve a combinação das forças complementares do YOLOv11 e Detectron2 através de abordagens híbridas. Trabalhos futuros podem explorar a criação de sistemas que utilizem o YOLOv11 para detecção inicial rápida de regiões suspeitas, seguida por análise mais detalhada com Detectron2 para classificação precisa e localização refinada dos defeitos.

Referências

BANDERCHUK, A. C. **Detecção de Defeitos de Fabricação em Placas de Circuito Impresso Utilizando Técnicas de Aprendizado Profundo**. Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina, Campus Florianópolis. Florianópolis, SC, p. 86. 2021.

BOCHKOVSKIY, A.; WANG, C.-Y.; MARK LIAO, H.-Y. **YOLOv4: Optimal Speed and Accuracy of Object Detection**. [S.l.], p. 17. 2020.

DING, R. *et al.* TDD-Net: A Tiny Defect Detection Network for Printed Circuit Boards. **CAAI Transactions on Intelligence Technology**, 4, maio 2019.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [S.l.]: The MIT Press Cambridge, 2016.

GUIMARÃES, F. D. C. **Desenvolvimento de uma API utilizando uma rede neural convolucional YOLOv7 para detecção de defeitos em placas de circuito impresso**. Universidade Federal do Amazonas - UFAM. Manaus, p. 67. 2024.

HUANG, ; WEI,. **A PCB Dataset for Defects Detection and Classification**. Peking University. Shenzhen, CN. 2019.

JIANG, Z. *et al.* **Real-time object detection method for embedded devices**. [S.l.]. 2020.

KHANAM, ; HUSSAIN, M. **WHAT IS YOLOV5: A DEEP LOOK INTO THE INTERNAL FEATURES OF THE POPULAR OBJECT DETECTOR**. [S.l.]. 2024.

LEE, J. *et al.* Industrial Artificial Intelligence for industry 4.0-based manufacturing systems. **Manufacturing Letters**, 18, 2018., p. 20-23 Disponivel em: https://www.sciencedirect.com/science/article/pii/S2213846318301081. Acesso em: 21 maio 2025.

LETA, F. R.; FELICIANO, F. F.; MARTINS, F. P. "Computer vision system for printed circuit board inspection. **ABCM Symposium Series in Mechatronics**, 3, 2008.

LYU, C. *et al.* **RTMDet: An Empirical Study of Designing Real-Time Object Detectors**. [S.l.]. 2022.

MORAIS, L. E. C. **Detecção de Defeitos de Fabricação em Placas de Circuito Impresso Através de Visão Computacional e Aprendizado Profundo**. Universidade Federal do Rio Grande do Sul, Escola de Engenharia. Porto Alegre, p. 77. 2023.

REDMON, J. *et al.* You Only Look Once: Unified, Real-Time Object Detection. **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition**, 2016., p. 779-788

REN, S. *et al.* **Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks**. [S.l.]. 2016.

ROSA, A. A. D. **Reconhecimento por Imagem de Lances de Xadrez com Visão Computacional e Redes Neurais Convolucionais**. Instituto Federal de Educação, Ciência e Tecnologia de Santa. São José. 2025.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence:** A Modern Approach. 4. ed. [S.l.]: [s.n.], 2020.

SILVA, I. N. D.; SPATTI, D. H.; FLAUZINO, R. A. **Redes Neurais Artificiais para engenharia e ciências aplicadas**. São Paulo: Artliber, 2010.

SILVA, L. D. S. **Comparação de modelos emergentes para classificação de pequenos elementos em placas de circuito impresso (PCB) em tempo real.** UNIVERSIDADE ESTADUAL DE CAMPINAS. Limeira, p. 106. 2023.

SUSA, J. *et al.* **Cap-Eye-citor: A Machine Vision Inference Approach of Capacitor Detection for PCB Automatic Optical Inspection**. [S.l.]. 2020.

TERVEN, J.; CORDOVA-ESPARZA, D. **A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS**. [S.l.]. 2024.

TSAI, D.-M.; CHOU, Y.-H. Fast and Precise Positioning in PCBs Using Deep Neural Network Regression. **IEEE Transactions on Instrumentation and Measurement**, 69, 2020., p. 4692–4701

ULTRALYTICS. Ultralytics YOLO Documentos. **ultralytics**, 2025. Disponivel em: https://docs.ultralytics.com/pt/. Acesso em: 28 maio 2025.

VELOSO, L. T. **Um estudo comparativo de técnicas de validação cruzada aplicadas a modelos para dados desbalanceados**. Universidade de São Paulo. [S.l.]. 2022.

WEIMER, D.; SCHOLZ-REITER, B.; SHPITALNI, M. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. **CIRP Annals**, 65, n. 1, 2016., p. 417-420 Disponivel em: https://www.sciencedirect.com/science/article/pii/S0007850616300725. Acesso em: 21 maio 2025.

WU , *et al.* Detectron2. **Detectron2**, 2019. Disponivel em: https://github.com/facebookresearch/detectron2. Acesso em: 05 jun. 2025.