UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Lucas Eduardo Corazza Morais

Detecção de Defeitos de Fabricação em Placas de Circuito Impresso Através de Visão Computacional e Aprendizado Profundo

Porto Alegre

2023

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Lucas Eduardo Corazza Morais

Detecção de Defeitos de Fabricação em Placas de Circuito Impresso Através de Visão Computacional e Aprendizado Profundo

Projeto de Diplomação II, apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de Engenheiro Eletricista

UFRGS

Orientador: Prof. Dr. Tiago Oliveira Weber

Porto Alegre
2023

Resumo

O trabalho compreende o treinamento de um modelo de Aprendizado Profundo para a detecção de defeitos de fabricação em Placas de Circuito Impresso, utilizando as topologias de Rede Neurais Convolucionais Tiny-YOLOv4, YOLOv5 e Faster R-CNN, e sua posterior adaptação a uma plataforma de *hardware* limitado Raspberry Pi 3B+ Para o treinamento de validação dos modelos, foi utilizada a base de dados PCB Defect Dataset (DING *et al.*, 2019). No decorrer do trabalho, foi utilizada a técnica de *pruning* não estruturado L1, considerando diferentes níveis de esparsidade, e a quantização de pesos dos modelos, nos formatos numéricos 32 bits ponto flutuante, 16 bits ponto flutuante e 8 bits inteiro, por meio das bibliotecas ONNX e TFLite, visando diminuir a carga computacional e tamanho de memória ocupado pelo modelo. Após avaliação dos resultados obtidos com as diferentes topologias, considerando a aplicação das técnicas de *pruning* e quantização, um modelo final foi selecionado, segundo o critério da aproximação da Curva de Pareto, e testado na plataforma Raspberry Pi 3B+. Os resultados, obtidos com o modelo YOLOv5, submetido ao *pruning* e quantizado, na plataforma Raspberry Pi, são de mAP(IoU=0,5) de 0,972 e tempo de inferência de 1,872 segundos, demonstrando a viabilididade desta abordagem.

Palavras-chave: aprendizado profundo, detecção de objetos, defeitos de fabricação em placas de circuito impresso, YOLO, Faster R-CNN, pruning, quantização.

Abstract

The work comprises the training of a Deep Learning model for the detection of manufacturing defects in Printed Circuit Boards, using the Tiny-YOLOv4, YOLOv5 and Faster R-CNN Convolutional Neural Network topologies, and its subsequent adaptation to a Raspberry Pi 3B+ limited *hardware* platform For the training and validation of the models, the PCB Defect Dataset (DING *et al.*, 2019) database was used. Throughout the work, the L1 unstructured *pruning* technique was used, considering different levels of sparsity, and the quantization of model weights, in 32-bit floating point, 16-bit floating point and 8-bit integer formats, using the ONNX and TFLite libraries, in order to reduce the computational load and memory size occupied by the model. After evaluating the results obtained with the different topologies, considering the application of the *pruning* and quantization techniques, a final model was selected, according to the Pareto Curve approximation criterion, and tested on the Raspberry Pi 3B+ platform. The results obtained with the YOLOv5 model, subjected to *pruning* and quantized on the Raspberry Pi platform, are a mAP(IoU=0.5) of 0.972 and an inference time of 1.872 seconds, demonstrating the viability of this approach.

Keywords: deep learning, object detection, PCB manufacturing defects, YOLO, Faster R-CNN, pruning, quantization.

Lista de Figuras

Figura 1 – Seção lateral de diferentes tipos de PCBs 15
Figura 2 – Tipos de defeitos abordados no trabalho: a) perfuração faltante; b) Mordida de rato; c) Circuito aberto; d) Curto circuito; e) Esporão; f) Cobre com esporão. 17
Figura 3 – Topologia da rede YOLOv5m. 29
Figura 4 – Etapas de desenvolvimento do projeto 35
Figura 5 – Exemplo de imagem bruta 37
Figura 6 – Exemplo de imagem tratada, apresentando instâncias de defeito circuito aberto. 38
Figura 7 – Exemplo de imagens após aplicação de *data augmentation* 43
Figura 8 – Computador de placa única Raspberry Pi 3B+. 47
Figura 9 – Histograma dos rótulos. 50
Figura 10 – Correlograma dos rótulos. 51
Figura 11 – Perdas de classificação, caixa delimitadora e confiança de presença de objeto (respectivamente, *class*, *box* e *obj loss*) nos conjuntos de treinamento e validação durante o treinamento do modelo, em função da época. 54
Figura 12 – Evolução do mAP(IoU=0,5) para YOLOv5, em função da esparsidade alvo do *pruning*. 56
Figura 13 – Evolução do mAP(IoU=0,5) para Faster R-CNN em função da esparsidade alvo do *pruning*. 57
Figura 14 – Evolução do mAP(IoU=0,5) para o modelo YOLOv5 em função da esparsidade e método de quantização. 60
Figura 15 – Evolução do tempo de inferência para o modelo YOLOv5 em função da esparsidade e método de quantização. 60
Figura 16 – Evolução do tamanho de modelo para YOLOv5 em função da esparsidade e método de quantização. 61
Figura 17 – Evolução do mAP(IoU=0,5) para o modelo Faster R-CNN em função da esparsidade e quantização ONNX ponto flutuante 32. 62
Figura 18 – Evolução do tempo de inferência para o modelo Faster R-CNN em função da esparsidade e quantização ONNX ponto flutuante 32. 63
Figura 19 – Evolução do tamanho de modelo para Faster R-CNN em função da esparsidade e quantização ONNX ponto flutuante 32. 63
Figura 20 – Comparativo dos modelos treinados frente ao Estado da Arte. 64
Figura 21 – Comparativo dos modelos quantizados frente ao Estado da Arte. 65
Figura 22 – Rótulos verdadeiros em placas inteiras da base PCB Dataset. 66

Figura 23 – Resultados da detecção em placas inteiras da base PCB Dataset. 67
Figura 24 – Matriz de confusão para as previsões do modelo final, no conjunto de teste (valores normalizados por coluna). 68
Figura 25 – Rótulos verdadeiros em imagem do conjunto de teste. 69
Figura 26 – Resultados da detecção em imagem do conjunto de teste. 70

Lista de Tabelas

Tabela 1 – Trabalhos recentes abordando a detecção de defeitos em PCBs empregando técnicas clássicas. 32
Tabela 2 – Trabalhos recentes abordando a detecção de defeitos em PCBs empregando Aprendizado Profundos. 33
Tabela 3 – Distribuição das classes de defeito nas imagens brutas. 36
Tabela 4 – Parâmetros de *data augmentation* do modelo YOLOv5 42
Tabela 5 – Parâmetros de *data augmentation* do modelo Faster R-CNN 44
Tabela 6 – Distribuição das classes de defeito nas imagens pré-tratadas. 49
Tabela 7 – Resultados de validação e teste para o treinamento inicial Tiny-YOLOv4. 52
Tabela 8 – Resultados por classe para o treinamento inicial Tiny-YOLOv4. 52
Tabela 9 – Resultados de validação e teste para o treinamento inicial YOLOv5. . . 53
Tabela 10 – Resultados por classe para o treinamento inicial YOLOv5 53
Tabela 11 – Resultados de validação e teste para o treinamento inicial Faster R-CNN. 54
Tabela 12 – Resultados por classe para o treinamento inicial Faster R-CNN. 55
Tabela 13 – Resultados para o *pruning* YOLOv5. 55
Tabela 14 – Resultados para o *pruning* Faster R-CNN. 56
Tabela 15 – Resultados para a quantização Tiny-YOLOv4, considerando o modelo não submetido ao *pruning* 58
Tabela 16 – Resultados para a quantização YOLOv5 59
Tabela 17 – Resultados para a quantização Faster R-CNN. 62
Tabela 18 – Resultados de testes com as placas inteiras da base PCB Defect Dataset. 66
Tabela 19 – Comparação dos resultados obtidos no conjunto de teste para cada etapa do experimento com o modelo YOLOv5, com esparsidade de 0,1 e quantização ONNX 32 bits. 68
Tabela 20 – Decomposição do tempo total de inferência (em segundos). 71
Tabela 21 – Decomposição do tempo total de inferência (em segundos). 71

Lista de abreviaturas

ANN	Artificial Neural Network
AP	Average Precision
CNN	Convolutional Neural Network
CPU	Central Processing Unit
PCB	Printed Circuit Board
GPU	Graphics Processing Unit
IoU	Intersection Over Union
mAP	Mean Average Precision
PF	Ponto Flutuante
R-CNN	Region Convolutional Neural Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RoI	Region Of Interest
RPN	Region Proposal Network
TCC	Trabalho de Conclusão de Curso
YOLO	You Only Look Once

Sumário

1 INTRODUÇÃO 11
1.1 Justificativa 12
1.2 Objetivos 13
1.2.1 Objetivo Geral 13
1.2.2 Objetivos Específicos 13

2 FUNDAMENTAÇÃO TEÓRICA E REVISÃO BIBLIOGRÁFICA . . 14
2.1 Processo de fabricação de PCBs 14
2.1.1 Defeitos comuns de fabricação em PCBs 15
2.1.2 Inspeção óptica automatizada de PCBs 18
2.2 Aprendizado de Máquina 18
2.2.1 Redes Neurais Artificiais 19
2.2.1.1 Treinamento da Rede Neural Artificial 20
2.2.1.2 Redes Neurais Convolucionais 21
2.2.1.3 *Data Augmentation* 23
2.2.1.4 Aprendizado por Transferência 24
2.3 Detecção de Objetos 25
2.3.1 Métricas de desempenho 26
2.3.2 Topologias comuns de detecção de objetos 27
2.3.2.1 YOLOv4 27
2.3.2.2 YOLOv5 28
2.3.2.3 Faster R-CNN 29
2.3.3 Pruning 30
2.3.4 Quantização 31
2.4 Estado da Arte 32
2.4.1 Detecção de Defeitos em PCBs 32
2.4.1.1 Técnicas Clássicas 32
2.4.1.2 Aprendizado Profundo 33

3 METODOLOGIA 35
3.1 Materiais utilizados 35
3.1.1 Ferramentas computacionais 36
3.2 Bases de dados utilizada 36
3.2.0.1 Análise da base de dados 38
3.2.0.2 Divisão em conjuntos de treinamento, validação e teste 39
3.3 Seleção de modelos 39

3.3.1 Seleção de topologias e treinamento dos modelos . . . 39
3.3.2 Pruning . . . 40
3.3.3 Quantização . . . 40
3.4 Pré-processamento dos dados . . . 40
3.4.1 Adequação das anotações . . . 40
3.5 Treinamento do modelo no conjunto de dados PCB Defect Dataset 41
3.5.0.1 YOLOv5 . . . 41
3.5.0.2 Faster R-CNN . . . 43
3.5.0.3 Tiny-YOLOv4 . . . 44
3.6 Pruning . . . 44
3.7 Quantização . . . 45
3.8 Análise dos resultados finais e comparação com o Estado da Arte . 46
3.9 Estudo de caso: Implementação do sistema de detecção embarcado na plataforma Raspberry Pi . . . 46
3.9.1 Materiais utilizados . . . 46
3.9.2 Seleção do modelo embarcado . . . 47
3.9.3 Testes realizados . . . 48

4 RESULTADOS . . . 49
4.1 Análise da base de dados . . . 49
4.2 Treinamento inicial dos modelos . . . 51
4.2.1 Tiny-YOLOv4 . . . 52
4.2.2 YOLOv5 . . . 53
4.2.3 Faster R-CNN . . . 54
4.3 Pruning dos modelos . . . 55
4.3.1 YOLOv5 . . . 55
4.3.2 Faster R-CNN . . . 56
4.4 Quantização dos modelos . . . 57
4.4.1 Tiny-YOLOv4 . . . 57
4.4.2 YOLOv5 . . . 58
4.4.3 Faster R-CNN . . . 61
4.5 Análise final dos resultados . . . 64
4.5.1 Comparação com o Estado da Arte . . . 64
4.5.1.1 Modelos originais . . . 64
4.5.1.2 Modelos após *pruning* e quantização . . . 64
4.5.2 Análise da base de dados de placas inteiras . . . 65
4.6 Estudo de caso - Implementação no Raspberry Pi 3B+ . . . 67
4.6.1 Desempenho de classificação . . . 67
4.6.2 Análise do tempo de inferência . . . 70
4.6.3 Análise do uso de recursos computacionais . . . 71

5 CONCLUSÕES 72
5.1 Trabalhos futuros 72

REFERÊNCIAS BIBLIOGRÁFICAS 74

1 Introdução

O controle de qualidade representa uma grande preocupação no processo de fabricação de Placas de Circuito Impresso (PCIs, ou PCBs, do inglês *Printed Circuit Board*). É nesta etapa que são detectados possíveis defeitos na fabricação, como curto-circuitos, conexões em aberto, solda mal aplicada e furos desalinhados. Estes defeitos podem impedir o correto funcionamento do sistema no qual a placa é instalada, causando prejuízos imediatos ou a longo prazo. É essencial que as placas defeituosas sejam retiradas de circulação antes da montagem de componentes eletrônicos e da instalação em equipamentos, a fim de evitar desperdícios de material e tempo e impedir a expedição de equipamentos defeituosos.

Os métodos de detecção de defeitos em PCB utilizados atualmente dividem-se em duas grandes classes, com e sem contato (EBAYYEH; MOUSAVI, 2020). A inspeção com contato, realizada por meio de sondas que aplicam e medem sinais elétricos, necessita de equipamentos custosos e complexos, sendo pouco adaptável e podendo provocar danos na placa avaliada. A inspeção sem contato abrange técnicas como a imagem por raios-X e por ultrassom, sendo a Inspeção Óptica a mais utilizada. A inspeção visual humana é o método mais simples, porém apresenta baixa sensibilidade e repetibilidade, além de ser custosa em tempo. Por estes motivos, a indústria tende a utilizar a Inspeção Óptica Automatizada, por meio da visão computacional clássica e/ou Aprendizagem de Máquina. Os equipamentos de Inspeção Óptica Automatizada, contudo, também são pouco acessíveis em custo, dificultando sua adoção massiva em indústrias de menor escala e a tornando inacessível para a fabricação artesanal de PCBs (seja em contexto acadêmico, de projetos pessoais ou da fabricação customizada de PCBs para aplicações comerciais).

Os métodos de visão computacional por Aprendizado Profundo se inserem como uma nova alternativa para a Inspeção Óptica Automatizada, permitindo obter maior precisão que os métodos tradicionais e facilitando a inspeção de circuitos cada vez mais complexos (RICHTER; STREITFERDT; ROZOVA, 2017). A capacidade de aprendizagem das Redes Neurais Convolucionais para tratar de diferentes problemas complexos tornam esta técnica especialmente promissora na detecção de defeitos de fabricação. A sua principal vantagem, face aos métodos tradicionais, é a adaptabilidade, pois um modelo de Aprendizado Profundo treinado em uma base de dados suficientemente grande permite a detecção de defeitos em placas de diversos *layouts*, utilizando apenas a imagem não processada da placa, dispensando a necessidade de uma etapa complexa extração de características, necessária para os métodos tradicionais, que acarreta maior tempo de processamento e menor adaptabilidade, em função da exigência de ajustes finos (BHATTACHARYA; CLOUTIER, 2022).

As técnicas de Aprendizado Profundo para detecção de objetos encontram cada vez mais aplicações comerciais, como decorrência do desenvolvimento de modelos pré-treinados (como as CNNs Res-Net e Xception) e de topologias especialmente adaptadas para a detecção (como as famílias YOLO e Faster R-CNN) (JIAO *et al.*, 2019), além de uma maior disponibilidade de base de dados e poder de processamento para efetuar o treinamento destes modelos.

1.1 Justificativa

Em um contexto de produção industrial de PCBs, a busca por controle de qualidade cada vez mais rigoroso é fundamental, permitindo maior valorização do produto e menores desperdícios durante a produção. Os métodos existentes de Inspeção Óptica Automatizada, baseados na visão computacional clássica, apresentam bom desempenho comprovado, porém possuem desvantagens como o seu custo elevado e falta de adaptabilidade, devido à necessidade de ajustes finos e da posse de *templates* de PCBs sem defeitos. Os métodos de Aprendizado Profundo podem oferecer desempenho igualmente elevado na detecção de defeitos em PCBs, conforme abordado por diversos trabalhos acadêmicos, além de também oferecerem maior adaptabilidade, característica importante para um setor que necessita de agilidade.

A aplicação destes métodos em um contexto industrial, contudo, permanece relativamente inexplorada. Entre os entraves presentes, estão o tempo de treinamento e latência de inferência elevados (exigindo bastante em termos de poder computacional) e a falta de bases de dados extensas o suficiente para o treinamento adequado dos modelos. Desta forma, a adaptação das técnicas de Estado da Arte em detecção de defeitos de PCBs para sua implementação prática, incluindo um estudo do impacto de técnicas de aumento de diversidade de bases de dado e de simplificação de modelos, apresenta grande potencial de impacto.

Adicionalmente, a utilização de modelos com número reduzido de parâmetros pode servir como base para a implementação em dispositivos portáteis, como computadores de placa única (como o Raspberry Pi) e *smartphones*, os quais muitas vezes apresentam *hardware* com poder de processamento limitado (pouca memória RAM, ausência de GPU, CPUs com menor poder de cálculo). A utilização de plataformas computacionais móveis e de baixo custo abriria a possibilidade de aplicação de um sistema de detecção de defeitos baseado em Aprendizado Profundo no contexto da confecção artesanal de PCBs. A fabricação de PCBs com desenho personalizado é frequentemente necessária para atividades educacionais, *hobbies*, reparo de equipamentos ou atividades econômicas em pequena escala. Estas atividades não justificam a aquisição de um sistema industrial de Inspeção Óptica Automatizada, porém poderiam se beneficiar de uma plataforma de detecção de falhas

de fabricação integrada a um aplicativo para celular ou a um microcomputador de baixo custo. Assim, um estudo de caso sobre a implementação de um sistema de detecção de defeitos em uma plataforma de baixo poder de processamento é justificado.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver sistema de detecção e classificação visual de defeitos de fabricação em placas de circuito impresso utilizando Aprendizado de Máquina Profundo e o avaliar, como estudo de caso, em um sistema com restrição de hardware.

1.2.2 Objetivos Específicos

- Estudar a literatura para identificar estado da arte no problema de classificação de defeitos em PCB;
- Implementação de técnicas encontradas na literatura e comparar em base de dados externa;
- Otimização da topologia adotada por meio de *pruning* e quantização;
- Comparação dos resultados obtidos com o Estado da Arte ;
- Estudo de caso: avaliação do modelo em plataforma de baixo poder computacional para aferição da latência.

2 Fundamentação Teórica e Revisão Bibliográfica

2.1 Processo de fabricação de PCBs

As principais etapas da fabricação de uma PCB são a aplicação da pasta de solda, instalação dos componentes na placa e a soldagem dos componentes nos *pads* (EBAYYEH; MOUSAVI, 2020).

O processo de fabricação começa pela elaboração do diagrama esquemático de circuito, uma representação funcional do circuito elétrico e dos componentes a serem instalados. Em seguida, é elaborado o *layout* da placa, definindo a localização física dos componentes na placa e a forma dos circuitos e conexões. Estas etapas são comumente realizadas com o apoio de softwares CAD (*computer aided design*).

O material bruto de uma PCB de lado único é uma placa elaborada de material isolante (o mais comum é o epóxi FR-4), revestida de cobre laminado em apenas um dos lados. A placa é devidamente limpa para evitar a presença de resíduos, como óleos e óxidos.

A etapa seguinte corresponde à transferência do *layout* para a placa. A transferência é realizada por diversos métodos, entre eles, a aplicação de tinta diretamente sobre a placa, a transferência térmica e a fotolitografia, por meio da aplicação de uma resina sensível à luz ultravioleta e, posteriormente, exposição a uma máscara contendo o *layout*. Este processo pode utilizar tanto uma imagem positiva quanto negativa do *layout*.

Na sequência, ocorre a etapa da gravura, um método subtrativo de fabricação no qual o cobre é removido das áreas indesejadas. Para este fim, são utilizados diversos processos químicos, entre eles a imersão ou pulverização com hipoclorito de ferro, persulfato de amônia, ácido crômico e a amônia alcalina.

Posteriormente, são realizadas as perfurações na placa, na qual os componentes serão inseridos. Uma vez completada, o processo de fabricação da placa está completado e a mesma pode seguir para a etapa de montagem, compreendendo a instalação e soldagem dos componentes.

Atualmente, a complexidade crescente das PCBs, com grande número de componentes e conexões entre eles, faz com que PCBs de duas camadas ou mais camadas se popularizem (KHANDPUR, 2006). Em PCBs de duas camadas, ambos os lados da placa são ocupados por circuitos condutores, separados pelo substrato isolante. Em PCBs multicamadas, com mais de duas camadas condutoras, a fabricação envolve a junção de

duas ou mais PCBs em formato de "sanduíche", separadas por material isolante. Em ambos os tipos de placa, a conexão entre as diversas camadas é realizada através da metalização da parede das perfurações. A Figura 1 apresenta os diferentes tipos de PCBs abordados (camada única, duas camadas e lado único, múltiplas camadas e duas camadas e dois lados).

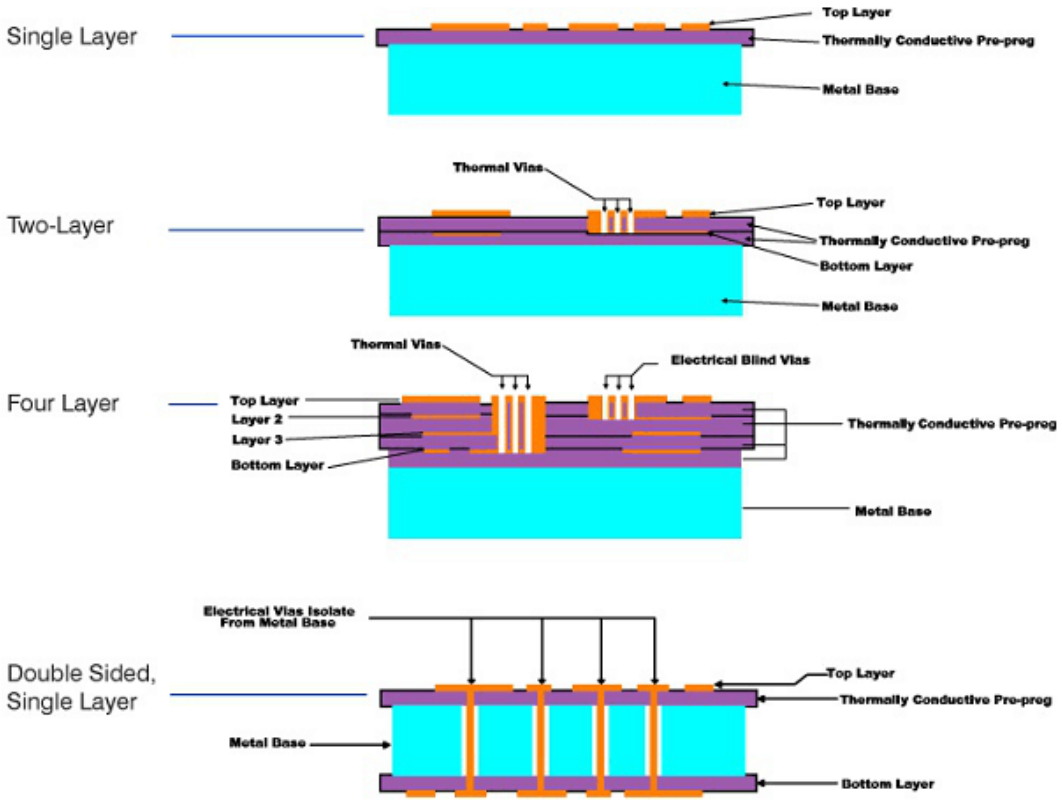

Figura 1 – Seção lateral de diferentes tipos de PCBs
Fonte - (WANG, 2021).

2.1.1 Defeitos comuns de fabricação em PCBs

Este trabalho, contudo, abordará os defeitos ocorridos até o processo de perfuração da placa, denominados defeitos pré-montagem. Caso estes defeitos não sejam identificados, o possível mau funcionamento decorrente pode ser custoso, especialmente em PCBs que contem componentes complexos, como microprocessadores. Em circuitos com alta densidade de componentes, as taxas de falha podem atingir 20% (EBAYYEH; MOUSAVI, 2020).

Os defeitos de fabricação pré-montagem podem ser divididos em dois grandes grupos: defeitos funcionais e visuais. Enquanto a primeira classe de defeito impede o bom funcionamento da PCB, o segundo tipo de defeito não provoca danos imediatos, mas pode comprometer o funcionamento do sistema no longo prazo (IBRAHIM; AL-ATTAS, 2005). Embora alguns defeitos possam ser identificados por meio de testes elétricos aplicados na

PCB, um grande número de defeitos visuais pode ser detectado apenas graças a inspeção visual automatizada (EBAYYEH; MOUSAVI, 2020).

Entre os defeitos que podem ser inspecionados visualmente, estão o mau posicionamento e orientação de componentes, soldagem defeituosa, falhas nas trilhas de cobre e nas perfurações da placa.

Os defeitos abordados neste trabalho são:

- Circuito aberto (*open circuit*): trilha de cobre incompleta.
- Curto-circuito (*short circuit*): conexão indesejada entre trilhas.
- Perfuração faltante (*missing hole*): falha da perfuração no local de instalação de um componente.
- Mordida de rato (*mousebite*): trilha de cobre apresenta redução da espessura de forma irregular em um segmento.
- Esporão (*spur*): trilha de cobre apresenta aumento da espessura de forma irregular em um segmento.
- Cobre com esporão (*spurious copper*): segmentos isolados e irregulares de cobre presentes no substrato da placa.

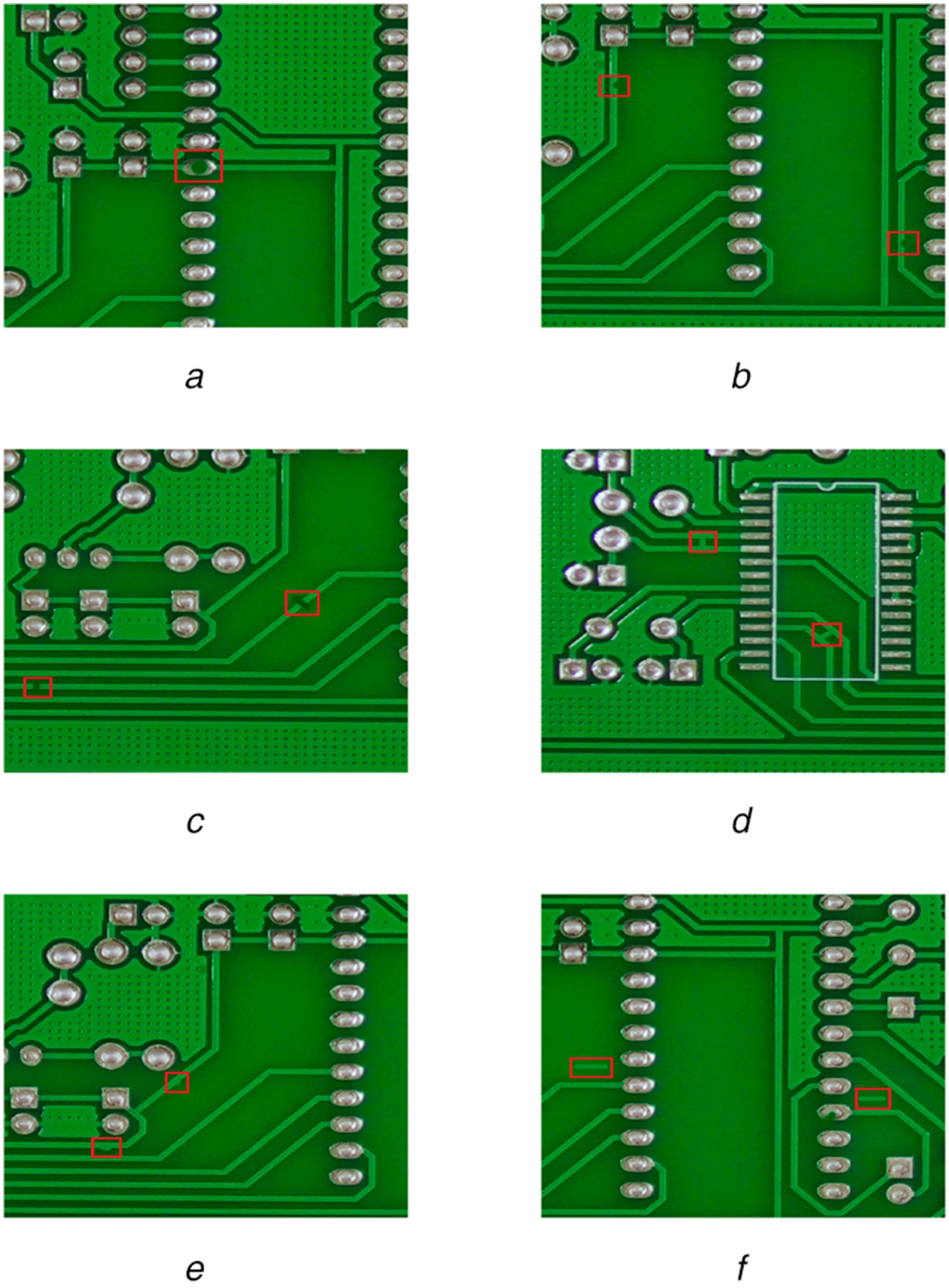

Figura 2 – Tipos de defeitos abordados no trabalho: a) perfuração faltante; b) Mordida de rato; c) Circuito aberto; d) Curto circuito; e) Esporão; f) Cobre com esporão.
Fonte - (DING *et al.*, 2019).

2.1.2 Inspeção óptica automatizada de PCBs

Os métodos de testes de PCBs se dividem em técnicas destrutivas (com efeitos negativos sobre a integridade da placa) e não destrutivas (sem efeitos colaterais) (EBAYYEH; MOUSAVI, 2020). Os testes destrutivos não podem ser efetuados em todas as PCBs produzidas, apenas por amostragem, por exigirem o descarte posterior da placa. Entre as técnicas destrutivas, encontra-se o teste de circuito com sonda elétrica, a análise por raio-x, o teste de penetração com tingimento e a termografia (utilizados para detectar fraturas na placa). Entre as técnicas não destrutivas, encontram-se o teste por ultrassom e a inspeção óptica.

A inspeção óptica é uma técnica bastante utilizada, tendo em vista sua rapidez, baixo custo e capacidade de detectar uma grande gama de defeitos superficiais, seja pré ou pós-montagem. Quando realizada manualmente, a inspeção óptica é sujeita a erros humanos e aumenta custos com mão-de-obra. Consequentemente, a inspeção óptica automatizada (AOI) tende a ser adotada pela indústria.

O processo de AOI consiste nas seguintes etapas:

1. Aquisição de imagens: as imagens são adquiridas por meio de uma ou mais câmeras de alta resolução.

2. Extração de características: é feito o processamento das imagens, utilizando-se tanto técnicas de visão computacional clássica, como a detecção de bordas e a transformada de Hough, quanto a aplicação de modelos de Aprendizado de Máquina.

3. Classificação: os defeitos existentes são devidamente localizados e classificados, por meio de técnicas como a correspondência de *template* (a imagem pré-tratada é comparada ao *template* de uma PCB sem defeitos, por meio de uma operação XOR, a fim de localizar as diferenças), e a aplicação de modelos de Machine Learning clássicos.

A aplicação de técnicas de Aprendizado Profundo é bastante investigada em artigos recentes (EBAYYEH; MOUSAVI, 2020), devido a sua grande capacidade em identificar características complexas nas imagens. Estes modelos, contudo, requerem grande poder de processamento e bases de dados extensas para o seu treinamento, dificuldades que precisam ser mitigadas para o seu emprego na detecção de defeitos em PCBs.

2.2 Aprendizado de Máquina

O Aprendizado de Máquina consiste na aplicação de modelos computacionais treinados na execução de determinadas tarefas. O processo de treinamento, realizado a

partir de um conjunto de dados fornecido, buscando otimizar a resposta do modelo de acordo com uma determinada métrica. Pode-se dizer, portanto, que o modelo "aprende"com os dados (GERON, 2019).

O Aprendizado de Máquina pode ser dividido em diversas categorias, dentre elas:

- Aprendizado Supervisionado: os dados de treinamento consistem em características (variáveis) explicativas (numéricas ou qualitativas) e uma (ou mais) variável(is) alvo (que também podem ser numéricas ou qualitativas), sendo que o objetivo do modelo é inferir a variável alvo a partir das variáveis explicativas. Esta classe de problemas engloba diversas sub-categorias, incluindo problemas de classificação de imagens e detecção de objetos.

- Aprendizado Não-Supervisionado: não é fornecida uma variável alvo nos dados de treinamento, o treinamento busca otimizar a extração de informações e padrões diretamente a partir das características explicativas. Esta classe de problemas engloba, entre outros, o agrupamento *clustering* de informações e a redução de dimensionalidade de um conjunto de dados.

- Aprendizado por Reforço: o modelo é treinado para uma determinada tarefa por meio de um processo de retroalimentação, onde um sinal de sucesso (ou fracasso) na tarefa pretendida é retornado em função da resposta fornecida pelo modelo.

2.2.1 Redes Neurais Artificiais

Uma Rede Neural Artificial (RNA) é um modelo computacional de Aprendizado de Máquina inspirado no funcionamento do cérebro humano, projetado para realizar tarefas de aprendizado de máquina, como classificação, regressão, reconhecimento de padrões e processamento de dados. As redes neurais artificiais são compostas por unidades de processamento interconectadas, chamadas neurônios artificiais.

Primeiramente concebido na década de 1950 (ROSENBLATT, 1958), o neurônio artificial, também conhecido como *perceptron*, tem comportamento inspirado nos neurônios que compõem o cérebro biológico. A entrada do neurônio é um vetor de características, e cada uma delas é associada a um peso que determina a força da sua influência. Além disso, cada neurônio também possui um viés (bias) que controla o ponto de ativação do neurônio. A ativação de um neurônio é calculada como uma combinação ponderada dos valores de entrada, juntamente com o viés, passada por uma função de ativação. Esta função possibilita a introdução de não-linearidades essenciais, permitindo ao neurônio capturar relações mais complexas nos dados (PABLO, 2018).

Ainda assim, o neurônio artificial possui sérias limitações, sendo limitado a conjuntos de dado com separação linear. A solução para estas limitações é a adição de múltiplas

camadas de neurônios artificias, formando uma Rede Neural Artificial. Nela, a saída de uma camada é utilizada como entrada da camada posterior, sucessivamente. A estrutura básica de uma Rede Neural Artificial inclui:

- Camada de Entrada: Essa camada recebe os dados de entrada, que podem ser imagens, textos, sinais, etc. Cada característica dos dados é representada por um neurônio na camada de entrada.
- Camadas Ocultas: São uma ou mais camadas intermediárias entre a camada de entrada e a camada de saída. Cada neurônio nessas camadas processa informações e transmite para os neurônios subsequentes. As camadas ocultas são cruciais para a capacidade da rede neural de aprender representações hierárquicas e complexas dos dados.
- Camada de Saída: Essa camada produz os resultados da rede neural. A quantidade de neurônios na camada de saída depende da natureza da tarefa. Por exemplo, em uma tarefa de classificação, o número de neurônios na camada de saída corresponderia ao número de classes possíveis.

A escolha da função de uma função de ativação adequada é fundamental, dado que ela é responsável por introduzir comportamentos não-lineares no modelo. Entre as funções mais comumente utilizadas, estão as funções identidade, sinal, unidade linear retificada (ReLU), sigmoide e tangente hiperbólica. Todas as funções apresentadas são monotonicamente crescentes, e, com exceção das funções identidade e ReLU, apresentam saturação. Exceto pela função sinal, estas funções apresentam também a vantagem de ser diferenciáveis (PABLO, 2018).

2.2.1.1 Treinamento da Rede Neural Artificial

Os parâmetros (pesos e vieses) da Rede Neural Artificial podem ser otimizados por meio da técnica da retropropagação (*backpropagation*) e de algorítimos de otimização, como a descida de gradiente estocástico (SGD, do inglês *stochastic gradient descent*) (PABLO, 2018).

Primeiramente, é necessário definir uma função de perda diferenciável. Entre as mais utilizadas, estão a perda quadrática (em problemas de regressão), a função de perda logística (problemas de classificação binária) e a entropia cruzada (classificação multi-classes).

A retropropagação consiste na aplicação da regra da cadeia visando obter a derivada da função de perda em função de cada um dos parâmetros (pesos e vieses) do modelo, permitindo a aplicação de um método de gradiente na otimização destes parâmetros,

visando minimizar a função de perda. O algoritmo da retropropagação é dividido em duas etapas:

- Fase de avanço: os valores de uma instância da amostra de treinamento são alimentados no modelo, e, a partir da saída fornecida e do resultado esperado, é calculada a perda.
- Fase de retropropagação: o gradiente da função de perda em relação a cada um dos parâmetros do modelo é calculado, partindo da camada final do modelo, e por meio da aplicação da regra da cadeia, computado sucessivamente para cada uma das camadas anteriores.

Uma vez que o gradiente de todos os pesos em função da perda é determinado, é possível otimizar os pesos visando a minimização da função de perda. Visando facilitar o processo de treinamento e diminuir a carga computacional, a otimização é comumente realizada em etapas, a partir de subconjuntos do conjunto de treinamento, e não com o conjunto inteiro. Um dos algoritmos mais utilizados para este fim é o SGD e suas variações. O algoritmo é composto das seguintes etapas (CHOLLET, 2018):

1. Amostragem aleatória de um subconjunto a partir do conjunto de treinamento, incluindo as características explicativas e a variável(eis) alvo.
2. Inferência e cálculo da função de perda para o subconjunto de treinamento selecionado.
3. Aplicação da retropropagação e cálculo do gradiente da função de perda em função dos parâmetros do modelo.
4. Atualização dos parâmetros na direção oposta ao gradiente, seguindo a fórmula $W_{i+1} = W_i - \alpha \frac{\partial L}{\partial W}$, onde i é a iteração atual, L o valor da função de perda, W o valor dos parâmetros e α o passo da otimização, que pode ser fixado ou ajustado no decorrer do treinamento.

2.2.1.2 Redes Neurais Convolucionais

As redes neurais convolucionais, ou CNNs (do ingês *convolutional neural network*) são baseadas no conceito de características ocultas, informações e padrões presentes nos dados de entrada (geralmente uma imagem ou *frame* de vídeo), que podem ser extraídas através das camadas convolucionais da rede. Em um problema de visão computacional, as características ocultas refletem componentes primitivos das imagens, como linhas e traços, e as camadas posteriores refinam o conhecimento extraído, identificando características mais complexas, como texturas, padrões e partes individuais de objetos, possibilitando,

por exemplo, que as camadas finais do modelo realizem a classificação de objetos (PABLO, 2018). O papel das camadas convolucionais, portanto, é de reduzir as dimensões iniciais dos dados, porém mantendo a informação necessária para desempenhar a tarefa na qual a CNN é empregada.

Uma rede neural convolucional apresenta geralmente entrada tri-dimensional, com as duas primeiras dimensões representando coordenadas espaciais e, a terceira, o número de canais (em uma imagem RGB, esta estrutura corresponde a um mapa de píxeis, contendo 3 canais de cor distintos). As aplicações de redes neurais convolucionais não são estritamente limitadas a imagens, porém é fundamental que os dados de entrada apresentem uma dimensão espacial.

Uma rede neural convolucional apresenta comumente as seguintes camadas:

- Camada convolucional: esta camada é composta por filtros convolucionais que realizam uma operação de convolução sobre a entrada do modelo, tendo como entrada uma matriz de dimensão $M_0 \times N_0 \times D_0$ e retornando um mapa de características de dimensão $M_1 \times N_1 \times D_1$. O filtro consiste em uma matriz, de dimensão $m \times n \times d$. As dimensões M_1 e N_1 do mapa de características resultante são determinadas pelas dimensões m e n do filtro e do passo da convolução, enquanto a dimensão $D - 1$ (o número de canais da saída) é igual ao número de filtros em cada camada. Muitas vezes, a fim de se preservar as dimensões de entrada e evitar a perda de informação, é realizado o *padding*, o alargamento das bordas da matriz de entrada e preenchimento com um valor, normalmente 0. Uma característica importante das camadas convolucionais é o fato delas serem esparsas, isto é, existem poucas conexões entre os diferentes neurônios.

- Camada de *pooling*: esta camada realiza o janelamento de um único canal da entrada, retornando uma saída de dimensões reduzidas. No *Max Pool*, a operação de janelamento considera apenas o elemento de maior valor incluído na janela. Novamente, as dimensões da saída dependem do tamanho da janela e de seu passo. O número de canais de entrada, contudo, é mantido.

- Camada ReLU: tipicamente, a função de ativação ReLU é aplicada após cada bloco convolucional.

- Camada densa: as camadas densas, geralmente correspondendo as camadas finais do modelo, recebem como entrada um vetor unidimensional correspondente ao mapa de características achatado. Sua função é de atuar como classificador final, a partir das características ocultas extraídas nas camadas anteriores. Seu nome representa o fato destas camadas serem densamente conectadas.

2.2.1.3 *Data Augmentation*

A técnica de aumento de dados (*Data Augmentation*, em inglês) é uma abordagem comum usada no treinamento de modelos de aprendizado de máquina, especialmente em tarefas relacionadas a visão computacional, processamento de imagem e processamento de áudio. O objetivo principal do aumento de dados é aumentar a diversidade e a quantidade de dados de treinamento disponíveis, melhorando assim a capacidade de generalização e o desempenho do modelo (GERON, 2019).

O processo de aumento de dados envolve a aplicação de transformações controladas aos dados de treinamento existentes, gerando novos exemplos que são variações realistas dos dados originais. Em imagens, essas variações podem incluir:

- Rotação: Girar a imagem em um ângulo específico para simular diferentes perspectivas.
- Espelhamento: Refletir a imagem horizontal ou verticalmente.
- Corte e Redimensionamento: Recortar uma parte da imagem original e redimensioná-la para gerar novas imagens.
- Translação: Transladar a imagem ao longo dos eixos X e/ou Y.
- Zoom: Ampliação ou redução das imagens.
- Alteração de Brilho e Contraste: Ajustar o brilho, contraste ou outros atributos de iluminação da imagem.
- Adição de Ruído: Introduzir ruído aleatório na imagem para simular variações do ambiente.
- Transformações Geométricas: Aplicar transformações como estiramento, distorção e cisalhamento à imagem.
- Mistura de Imagens: Combinação de várias imagens para criar exemplos compostos.
- Variação de Cor: Alterar as cores da imagem, como matizes e saturação.

As variações introduzidas nas imagens mantêm as características essenciais da tarefa, permitindo que o modelo aprenda a lidar com diferentes variações e condições. Este procedimento ajuda a evitar o *overfitting* do modelo aos dados de treinamento originais.

Além de melhorar o desempenho do modelo, o aumento de dados também pode reduzir a necessidade de coletar uma grande quantidade de dados de treinamento, útil em cenários em que a coleta de dados é custosa, demorada ou impraticável.

2.2.1.4 Aprendizado por Transferência

A técnica de aprendizado por transferência (*Transfer Learning*) é um conceito fundamental em aprendizado de máquina, onde conhecimentos adquiridos ao resolver um problema são transferidos para melhorar o desempenho em um problema relacionado. Modelos complexos de Aprendizado Profundo, como as CNNs, exigem conjuntos de treinamento extensos, e este processo exige muitos recursos computacionais, tornando o treinamento convencional pouco prático ou mesmo impossível para muitas tarefas.

O aprendizado por transferência é baseado na ideia de que as habilidades e representações aprendidas em uma tarefa podem ser úteis para melhorar o desempenho em uma tarefa diferente, porém similar. Como as representações aprendidas durante o treinamento na tarefa de origem já capturam informações úteis, o tempo e a quantidade de dados necessários para treinar efetivamente o modelo na tarefa de destino podem ser reduzidos.

Inicialmente, um modelo é treinado em uma tarefa de origem que possui uma abundância de dados e recursos. Esse treinamento inicial pode ser realizado usando técnicas convencionais de aprendizado de máquina, como redes neurais profundas. O modelo aprende a mapear entradas para saídas relevantes para essa tarefa específica.

Após o treinamento na tarefa de origem, o modelo é ajustado para a tarefa de destino, que geralmente possui menos dados disponíveis ou recursos limitados. O ajuste-fino consiste no re-treinamento do modelo, utilizando os dados disponíveis da tarefa de destino. No entanto, grande parte das camadas (e parâmetros) do modelo original são congeladas, isto é, permanecem inalteradas durante o treinamento. Geralmente, apenas as camadas finais do modelo serão re-treinadas. Assim, reduzindo-se o número de parâmetros otimizáveis, o tempo de treinamento é reduzido, e, dado que o modelo já foi treinado para uma tarefa similar, o ajuste-fino exige menor quantidade de dados de treinamento para que o modelo obtenha bom desempenho na tarefa de destino.

Essa abordagem tem sido amplamente utilizada em uma variedade de campos, incluindo visão computacional, processamento de linguagem natural e reconhecimento de padrões, permitindo economia de tempo e recursos durante o desenvolvimento de modelos de aprendizado de máquina.

Em problemas de classificação e detecção de objetos, a abordagem de aprendizado por transferência é muito frequentemente utilizada. As camadas extratoras de características são treinadas em bases de dados abertas, de grande extensão (as bases COCO e PASCAL-VOC são comumente utilizadas) e são congeladas durante o ajuste-fino. Assim, o ajuste fino consiste apenas no treinamento das camadas de classificação final.

2.3 Detecção de Objetos

A detecção de objetos é um dos principais tópicos de pesquisa na área de Visão Computacional, devido às suas várias aplicações (controle de veículos autônomos, reconhecimento facial, sistemas de vigilância eletrônica, entre outros) (JIAO *et al.*, 2019). Avanços recentes na arquitetura de redes de Aprendizado Profundo e o poder computacional cada vez mais acessível impulsionam o desenvolvimento de modelos e sistemas de detecção, abrangendo desde a pesquisa acadêmica até aplicações comerciais.

A detecção de objetos combina duas tarefas distintas, mas relacionadas: a localização de uma instância de objeto (sua delimitação espacial em um sistema de coordenadas) e a classificação deste mesmo (em um conjunto finito de classes). Trata-se, portanto, de um problema mais complexo do que a classificação simples de uma imagem. Devido a este fator, a abordagem de Aprendizado por Transferência é comumente utilizada.

A fim de permitir o treinamento e a avaliação do desempenho dos modelos, as instâncias das classes a serem detectadas presentes nas imagens de treinamento são comumente rotuladas por meio de caixas delimitadoras, ou *bounding boxes*. Estas caixas representam as coordenadas dentro das quais a instância de objeto está completamente contido, bem como a classe deste objeto.

Os modelos de detecção costumam ter como saída um conjunto de caixas delimitadoras propostas, contendo as coordenadas dentro das quais se estima a presença de um objeto, e a probabilidade de que este objeto pertença a uma determinada classe (*score* de confiança). A fim de obter-se uma classificação binária de classe, um piso mínimo de *score* de confiança pode ser definido, e a detecção é considerada apenas se a probabilidade for superior a este piso.

Existem duas grandes classes de modelos de detecção de objetos: modelos de estágio único ou de dois estágios. Ambas as classes possuem a mesma estrutura básica.

Os modelos de dois estágios, cujo maior representante é a família Faster R-CNN, podem ser divididos em três seções distintas, a espinha (*backbone*), pescoço (*neck*) e a cabeça (*head*)(JIAO *et al.*, 2019). A espinha, geralmente uma rede CNN, é responsável pela extração de características das imagens. Na sequência, o primeiro estágio da rede, a RPN (*Region Proposal Network*), propõe caixas delimitadoras para os objetos candidatos (todas as possíveis instâncias de objetos a serem detectadas). Por fim, o segundo estágio (*RoI Pooling*) extrai características de cada uma das caixas delimitadoras, para, por fim, classificar o objeto e determinar os limites precisos da caixa delimitadora por meio de uma regressão.

Os detectores de estágio único, como as redes YOLO, possuem princípio de funcionamento similar, porém sem a inclusão da RPN. A delimitação das caixas e classificação do

objeto é feita diretamente a partir das características extraídas pela espinha, permitindo uma predição mais rápida.

2.3.1 Métricas de desempenho

As principais métricas de avaliação de desempenho para a detecção de objetos são a *Precision* Média, ou *Average Precision* (AP), e sua derivação para problemas de classificação multiclasses, a mAP (*mean Average Precision* (PADILLA; NETTO; SILVA, 2020). Estas métricas são baseadas em alguns conceitos básicos:

- Verdadeiro Positivo, ou *True Positive* (TP): número de definições corretas de caixa delimitadora (em relação aos limites reais das instâncias de objetos a serem detectados),
- Falso Positivo, ou *False Positive* (FP): número de definições incorretas de caixa delimitadora, quando não existe nenhuma instância de objeto ou quando suas coordenadas não correspondem aos limites reais,
- Falso Negativo, ou *False Negative* (FN): quando uma instância existente de objeto deixa de ser detectada.

A medida da *Intersection over Union* (IoU) permite avaliar quantitativamente a qualidade da caixa delimitadora proposta. Definindo-se B_p como a caixa delimitadora proposta para uma instância de objeto, e B_{gt} como a caixa delimitadora real de uma instância de objeto de mesma classe, a IoU é dada por:

$$IoU = \frac{B_p \cap B_{gt}}{B_p \cup B_{gt}}$$

É possível utilizar a IoU para diferenciar uma detecção correta (Verdadeiro Positivo) ou errônea (Falso Positivo ou Falso Negativo) a partir da definição de um *treshold* mínimo, a partir do qual a detecção passa a ser contabilizada. Um valor comumente adotado para este *treshold* é de $0,5$, porém outras quantidades são possíveis, incluído a possibilidade de se realizar uma varredura do mAP calculado para diferentes valores de IoU e tomar a média entre eles.

A partir da medida de IoU e da definição de um *treshold* mínimo, é possível construir duas métricas de desempenho, (*Precision*) e (*Recall*)[1]:

[1] Em aplicações de inteligência artificial, frequentemente os termos *Recall* e *Precision* são convertidos para Sensibilidade e Precisão. No entanto, por não estarem de acordo com a definição destes termos tal qual o VIM (Vocabulário Internacional de Metrologia) e por entendermos que esta aplicação abrange a instrumentação, optamos por não utilizar os termos em português

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

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

A partir destas métricas, é possível traçar a curva de *Precision X Recall*, que descreve o comportamento da métrica *Precision* de um modelo em função do *Recall* obtido. Conforme descrito na seção anterior, a saída do modelo de detecção é uma probabilidade de que a instância de objeto detectada pertença a uma determinada classe. A curva é traçada separadamente para cada classe de objetos, variando-se o piso do *score* de confiança para uma detecção válida de 0 a 1, calculando as métricas de Precision e Recall para cada um destes valores, e traçando os pontos que compõem a curva. A fim de eliminar possíveis oscilações na curva, é feita uma interpolação dos pontos, considerando, para cada valor de *recall* sempre o valor máximo de *precision* cujo *recall* é igual ou superior ao *recall* vigente. A *Precision* Média, ou *Average Precision* (AP), pode ser obtida a partir do valor médio da curva traçada pela interpolação. Ao se calcular a média da AP(IoU = 0,5) para todas as classes, se obtêm a mAP(IoU = 0,5) (YOHANANDAN, 2020).

2.3.2 Topologias comuns de detecção de objetos

2.3.2.1 YOLOv4

A família de modelos YOLOv4 (BOCHKOVSKIY; WANG; LIAO, 2020) é uma classe de modelos de detecção de estágio único, baseado na série de modelos YOLO e implementada no *framework* Darknet, escrito na linguagem de programação C. Comparada a outros modelos de detecção, como a família YOLOv3 e EfficientDet(D0-D4), os modelos de tipo YOLOv4 oferecem um bom compromisso entre desempenho de detecção (mensurado pela mAP) e velocidade de inferência.

Os modelos YOLOv4 adotam a seguinte estrutura:

- Backbone: CSPDarknet53
- Neck: SPP, PAN
- Head: YOLOv3

Além dos componentes básicos de detecção, o modelo implementa uma série de funcionalidades adicionais a fim de melhorar seu desempenho durante o treinamento. Entre

elas, está a adoção de técnicas *data augmentation*, de tipo CutMix, MixUp, Mosaic e Bluring. A arquitetura completa do modelo é resumida na Figura ... .

O treinamento do modelo YOLOv4 foi efetuado com a base de dados MS COCO 2017 (LIN *et al.*, 2015), composta por 123.287 imagens e 886.284 instâncias de 80 diferentes classes.

A idade do modelo, lançado ao público em 2020, impõe algumas limitações. Novos modelos, como o YOLOv7, já apresentam desempenho de detecção melhorado. Em contrapartida, a maturidade dos modelos YOLOv4 permite o surgimento de diversas bibliotecas adicionais, implementando novas funcionalidades. Uma delas é a biblioteca *tensorflow-yolov4-tflite* , que permite a conversão do modelo Darknet para o *framework* TFLite, adequado a dispositivos com poder de processamento e memória reduzidos.

2.3.2.2 YOLOv5

A classe de modelos YOLOv5 (ULTRALYTICS, 2022) é uma família de modelos de detecção de estágio único, desenvolvida a partir da série de modelos YOLO. A família possui modelos de diferentes tamanhos, desde o YOLOv5n até o YOLOv5l, apresentando desempenho crescente (em termos de mAP, avaliados na base de dados COCO, porém as custas de maior número de parâmetros e latência na inferência.

A Figura 3 resume a arquitetura comum compartilhada por todos os modelos. O bloco básico formador da rede é uma série de camadas convolucionais, de função de ativação SiLU (*sigmoid linear unit*).

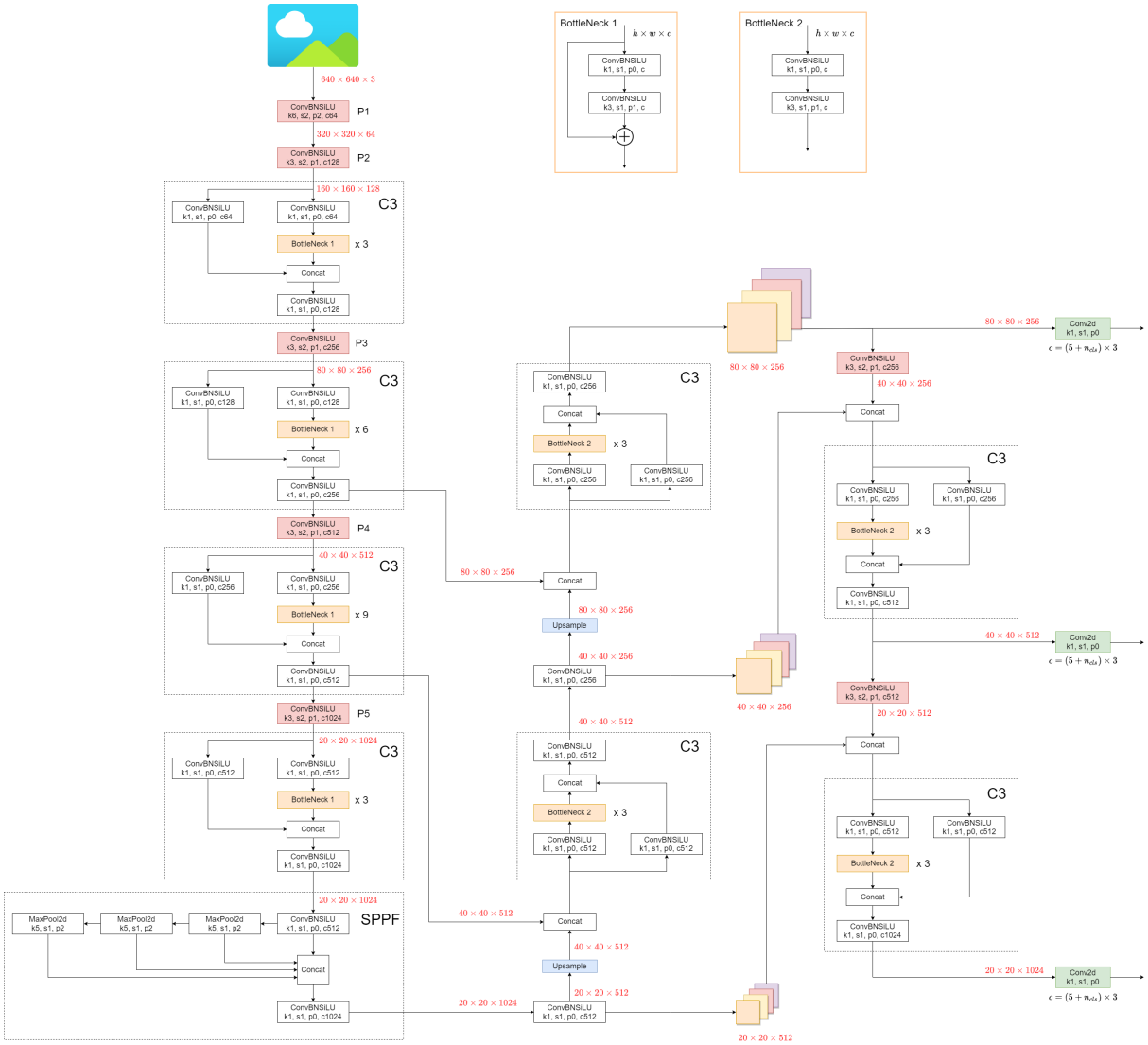

Figura 3 – Topologia da rede YOLOv5m.
Fonte - (ULTRALYTICS, 2022)

A rede é pré-treinada na base de dados COCO, com 330.000 imagens e 1,5 milhões de instâncias de objetos. O trinamento da biblioteca se dá por meio de linhas de comando, e diversos scripts incluídos no repositório GitHub *ultralytics/yolov5* permitem realizar predições e avaliar os resultados do modelo.

Os modelos YOLOv5 já integram uma etapa de *Data Augmentation*, permitindo diminuir os efeitos de bases de dado pequenas. Os hiperparâmetros de treinamento e *Data Augmentation* podem ser ajustados manualmente pelo usuário.

2.3.2.3 Faster R-CNN

A família de modelos de detecção de dois estágios Faster R-CNN (REN *et al.*, 2016) representou, em seu lançamento, um dos melhores desempenhos de detecção, em termos

de *Precision* e eficiência computacional, e ainda apresenta bom desempenho em relação a outras topologias (JIAO *et al.*, 2019).

A topologia é composta por dois módulos, a RPN (*region proposal network*, ou rede de proposição de regiões), uma CNN responsável por delimitar as coordenadas dos *bounding boxes* nos quais uma possível instância de objeto se encontra, e uma rede Fast R-CNN, responsável pela classificação dos objetos nas regiões propostas.

Uma inovação importante introduzida na topologia é a presença de caixas-âncora, permitindo a translação das regiões propostas para diferentes escalas, e assim aumentando o poder de detecção. A presença das caixas-âncora evita a necessidade de se utilizar múltiplos filtros convolucionais para cada escala, ou redimensionar as imagens, e assim tornando a topologia mais eficiente

Para permitir a economia de recursos computacionais, as camadas convolucionais são compartilhadas entre os dois módulos. Estas camadas podem ser um modelo extrator de características pré-treinado, como, por exemplo, a topologia VGG-16.

2.3.3 Pruning

A técnica de *pruning* (poda) em CNNs é um método de compressão e otimização que visa reduzir o tamanho e a complexidade dos modelos, removendo (ou atribuindo o valor 0) seletivamente conexões, neurônios ou camadas menos importantes. O objetivo é obter uma CNN mais leve e eficiente, podendo tornar a inferência mais rápida e diminuindo o tamanho de memória ocupado pelo modelo (na prática, permitindo sua compressão) (LI; LI,).

As abordagens para o *pruning* podem ser divididas entre as estruturadas, e não-estruturadas. O *pruning* não estruturado remove as conexões menos salientes no modelo (atribuindo o valor 0 a elas), segundo a métrica de escolha (uma métrica usual é a norma L1 dos pesos). O algoritmo não considera a relação entre os pesos na escolha de poda, apenas a métrica utilizada, assim sendo, os pesos podados podem estar em qualquer seção do modelo, e a eliminação de camadas ou canais inteiros não é almejada. Esta abordagem permite remover uma grande número de parâmetros do modelo com pequeno impacto no desempenho de generalização do modelo. No entanto, seus efeitos na latência e poder de processamento exigido pelo modelo são limitados, dependendo do hardware utilizado, pois muitas plataformas não são capazes de realizar cálculos envolvendo matrizes esparsas de forma eficiente (GHOLAMI *et al.*, 2021).

O *pruning* estruturado, por sua vez, envolve a remoção seletiva de estruturas maiores da rede neural, como uma camada ou um canal inteiro. O ganho de eficiência desta abordagem tende a ser mais pronunciado, contudo, os impactos no desempenho do modelo podem ser severos.

As abordagens de *pruning* também podem ser diferenciadas quanto ao momento de sua aplicação. No *pre-pruning*, a remoção de pesos é realizada durante o treinamento do modelo, correspondendo a uma regularização dos pesos. No *post-pruning*, a remoção é realizada sobre o modelo já treinado. O modelo pode, opcionalmente, ser re-treinado em um número menor de iterações (processo de ajuste fino), a fim de melhorar sua *Precision*.

2.3.4 Quantização

A técnica de quantização em Aprendizado Profundo é uma abordagem usada para reduzir o tamanho e a complexidade de modelos de redes neurais, resultando em modelos mais compactos e eficientes, porém as custas de maior erro no modelo. Essa técnica busca representar os parâmetros do modelo (por exemplo, os pesos das conexões entre neurônios), usando menos bits de precisão do que normalmente utilizados em representações de ponto flutuante de 32 bits.

Assim como no *pruning*, a quantização pode ser realizada durante o treinamento do modelo (QAT, ou *quantization aware training*) ou após o seu término (PQT, *post training quantization*). No QAT, o treinamento do modelo, incluindo os passos de propagação e retropropagação, são realizados com a representação usual de ponto flutuante, mas em seguida os pesos passam pelo processo de quantização. Este método tende a obter melhor precisão que o PQT, porém aumenta o tempo de treinamento, razão pela qual é menos utilizado. Com o segundo método, os pesos são quantizados apenas uma vez, sem necessitar o re-treinamento do modelo (GHOLAMI *et al.*, 2021).

A quantização pode ser efetuada com um intervalo simétrico ou assimétrico, sendo esta alternativa preferível para distribuições de valores não centradas em 0. Os intervalos de quantização podem ser fixos, definidos após o treinamento do modelo (quantização estática), ou dinâmicos, definidos a cada nova ativação (quantização dinâmica). O segundo método apresenta maior precisão, porém exige mais recursos computacionais. Finalmente, a quantização pode ser efetuada uniformemente (o passo dos degraus de quantização é constante) ou não-uniforme (os intervalos com maior ocorrências de valores possuem degraus com menor espaçamento). Novamente, a segunda alternativa é menos empregada devido a sua maior complexidade e custo computacional.

A implementação da quantização pode ser um desafio em algumas arquiteturas de rede neural e depende da disponibilidade de bibliotecas de software ou hardware que suportem operações de baixa precisão. Ainda assim, a quantização é uma técnica relevante, especialmente em cenários onde a eficiência do modelo é fundamental, como em dispositivos móveis, dispositivos embarcados e aplicações de *edge computing*.

2.4 Estado da Arte

Esta seção apresenta os trabalhos tidos como Estado da Arte na detecção de defeitos em PCBs(técnicas clássicas e por Aprendizado Profundo).

É apresentado o desempenho do modelo (se possível, em mAP(IoU = 0,5)) a fim de comparação.

2.4.1 Detecção de Defeitos em PCBs

2.4.1.1 Técnicas Clássicas

A Tabela 1 lista alguns dos trabalhos recentes de destaque na detecção de defeitos em PCBs utilizando técnicas clássicas de visão computacional e Aprendizado de Máquina.

Ao comparar os resultados obtidos, se deve atentar para o fato dos conjuntos de dados e das métricas não serem as mesmas, dificultando uma comparação direta. Assim, as me´tricas utilizadas por cada trabalho serão sempre apresentadas, e os trabalhos que utilizam a mesma base de dados serão sinalizados.

Trabalho	Métrica utilizada	Resultados obtidos
(CHAUDHARY; DAVE; UPLA, 2017)	Accuracy	100%
(LIU; QU, 2021)	Accuracy	94%

Tabela 1 – Trabalhos recentes abordando a detecção de defeitos em PCBs empregando técnicas clássicas.

Fonte - o Autor, 2023.

(CHAUDHARY; DAVE; UPLA, 2017) abordam um modelo completo de Inspeção Óptica Referencial (isto é, comparando a PCB a um modelo de referência). O trabalho detalha as etapas de tratamento das imagens, extração de características (por segmentação) e detecção de defeitos, por meio da comparação de características da amostra e do modelo, por subtração de píxeis, e aplicando uma heurística de classificação.

(LIU; QU, 2021) apresentam um sistema de detecção de defeitos baseado em um algoritmo de detecção de aberrações em imagem. As imagens, binarizadas em preto e branco e pré-tratadas (submetidas a suavização e aumento de contraste), passam por um processo que combina a comparação com um *template* e a aplicação do reconhecimento de padrões e morfologias matemáticas, obtendo assim maior taxa de acerto do que com cada uma das técnicas isoladas.

É possível obter, através das técnicas clássicas de visão computacional e/ou Aprendizado de Máquina, taxas de acerto elevadas. Estas técnicas, no entanto, exigem uma etapa de pré-processamento extensa, destinada a extrair as características relevantes da imagem (BHATTACHARYA; CLOUTIER, 2022), além de frequentemente exigirem a

utilização de um *template* de placa sem defeitos. Assim, a aplicação a diferentes tipos de placa pode exigir adaptações complexas no sistema.

2.4.1.2 Aprendizado Profundo

A Tabela 2 lista alguns dos trabalhos recentes de destaque na detecção de defeitos em PCBs utilizando técnicas de Aprendizado Profundo.

Trabalho	Métrica utilizada	Resultados obtidos
(SILVA *et al.*, 2019)	Accuracy	89%
(ZHANG; JIANG; LI, 2021)	Sensitivity	89%
(GHOSH *et al.*, 2018)	Accuracy	91,125%
(VOLKAU *et al.*, 2019)	Sensitivity	>90%
(MUJEEB *et al.*, 2019)	TPR	89%
(YOU, 2022)	F1	96,66 - 95,72
(HU; WANG, 2020)	mAP	95,6%
(TANG *et al.*, 2019)	mAP(IoU = 0.5)	98,6%
(BHATTACHARYA; CLOUTIER, 2022) *	mAP(IoU = 0.5)	98,1%
(WANG; ZHANG; ZHOU, 2021) *	mAP(IoU = 0.5)	97,9%
(JIN *et al.*, 2021) *	mAP(IoU = 0.5)	88.2
(DING *et al.*, 2019) *	mAP(IoU = 0.5)	98,9%
(NIU *et al.*, 2023) *	mAP(IoU = 0.5)	99,1%
(YANG; KANG, 2023) *	mAP(IoU = 0.5)	98,74%

Tabela 2 – Trabalhos recentes abordando a detecção de defeitos em PCBs empregando Aprendizado Profundos.
* Utilizam a base de dados HRIPCB (DING *et al.*, 2019)
Fonte - o Autor, 2023.

São utilizadas diferentes abordagens de Aprendizado Profundo, incluindo a utilização de diferentes classes de modelo. Um exemplo é a (MUJEEB *et al.*, 2019) propõem um modelo de detecção de defeitos utilizando *autoencoders* para a extração de características e classificação binária, comparando a placa em avaliação contra uma placa sem defeitos. Uma das vantagens dessa abordagem é a adaptabilidade do modelo, permitindo a detecção de diferentes defeitos com poucas modificações. Já o trabalho desenvolvido por (ZHANG; JIANG; LI, 2021) apresenta um sistema de detecção de defeitos cosméticos em PCBs, adotando uma rede neural convolucional ResNet modificada, aplicando uma penalidade no processo de otimização para diminuir os problemas associados a classes desbalanceadas.

Múltiplos trabalhos são baseados em versões modificadas das topologias YOLO. O artigo de (NIU *et al.*, 2023) propõe um classificador baseado na topologia YOLOv5 e a base de dados PCB Defects Dataset, porém adotando o método não-supervisionado *k-means* para melhorar a localização das caixas âncora, utilizando a função de perda Focal-EIOU e adicionando um módulo ECA-Net, obtendo aumento na velocidade e *Precision* do modelo. Por sua vez, (YANG; KANG, 2023) utilizam a topologia YOLOv7 e a base

de dados PCB Defects Dataset, modificada com a adição de uma camada convolucional para extração de características locais da PCB (módulo SwinV2_TDD), e introduzindo uma camada convolucional a mais no mecanismo de atenção da rede (utilizado para melhorar a capacidade de generalização do modelo). Já (BHATTACHARYA; CLOUTIER, 2022) propõem um sistema de detecção de defeitos de ponta-a-ponta para ser aplicado à indústria, utilizando uma topologia YOLOv5 modificada com menor número de parâmetros, diminuindo a latência de previsão sem sacrificar a taxa de acertos. Por fim, (WANG; ZHANG; ZHOU, 2021) abordam um modelo de detecção de defeitos baseado na rede CNN YOLOv5 (*you only look once*), porém adicionando uma série de camadas BiFPN (*Weighted Bi-directional Feature Pyramid Network*), visando melhorar o desempenho de classificação para pequenos defeitos. A topologia proposta também oferece uma redução no tempo de treinamento quando comparada ao modelo YOLO puro.

Existem também diversos trabalhos trabalhando com modelos baseados na topologia Faster R-CNN. O trabalho de (DING *et al.*, 2019) propõe uma rede CNN especializada (TDD-Net) para a detecção de pequenos defeitos de fabricação em PCBs, na base de dados PCB Defects Dataset. A topologia do modelo proposto é baseada no paradigma de detecção de defeitos Faster R-CNN, porém com a implementação adicional de camadas de *data augmentation* e fusão de características. Também é utilizada a técnica de aprendizagem por transferência, visto que os pesos das camadas Faster R-CNN foram pré-treinadas na base ImageNet. Já o artigo de (HU; WANG, 2020) apresenta um modelo baseado na arquitetura Faster R-CNN, utilizando uma rede de características piramidal para a extração de características e uma unidade GARPN para ancorar os objetos detectados.

3 Metodologia

O capítulo apresenta o procedimento adotado para o treinamento, validação e escolha de um modelo inteligente de detecção de defeitos em PCBs, bem como as adaptações realizadas para embarcar este modelo em uma plataforma de hardware limitado. O fluxograma (Figura 4) exemplifica o procedimento adotado na elaboração do trabalho.

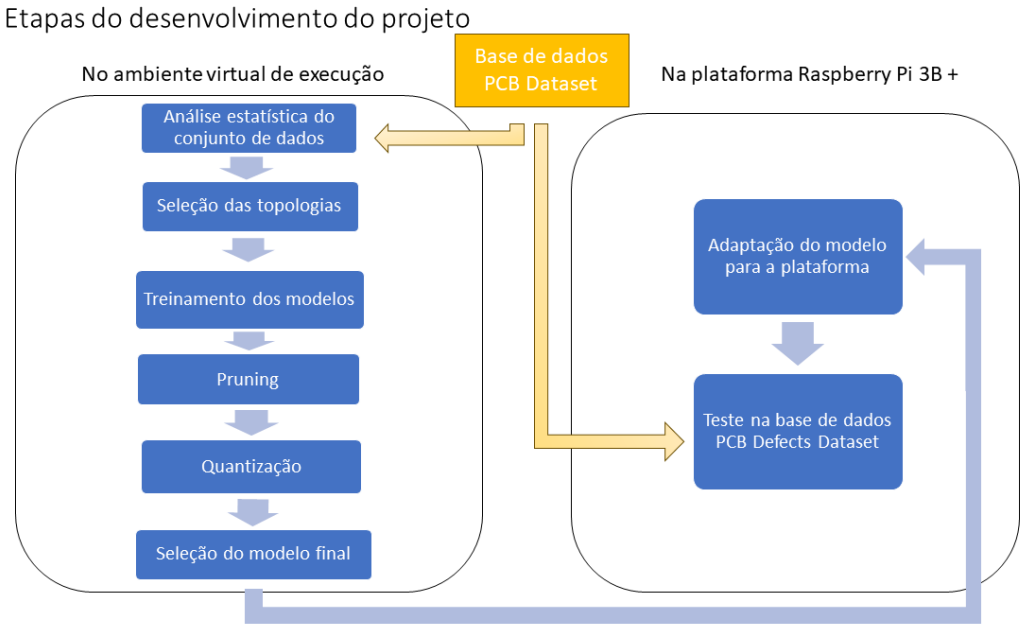

Figura 4 – Etapas de desenvolvimento do projeto
Fonte - o Autor, 2023.

3.1 Materiais utilizados

O trabalho foi desenvolvido utilizando a linguagem de programação de alto nível Python, versão 3.11. Para o tratamento de imagens, é utilizada a biblioteca OpenCV, v4.6.0. São também utilizadas as bibliotecas Pandas, v1.4.3 e Numpy, v1.23.1, e Scikit-Learn, versão 1.1.2, para a manipulação e exibição de dados.

Para a conversão e quantização dos modelos, foram utilizados os *frameworks* ONNX e TFLite. Ambos oferecem ambientes de execução adaptados a uma grande gama de plataformas, desde sistemas de baixo poder de processamento até GPUs de alto desempenho, facilitando a portabilidade dos modelos desenvolvidos.

O compartilhamento dos códigos e scripts desenvolvidos entre os diferentes ambientes de execução foi realizado utilizando a ferramenta Git. As bases de dados utilizadas

durante o treinamento foram disponibilizadas na plataforma de armazenamento e compartilhamento de arquivos virtual *Google Drive*.

3.1.1 Ferramentas computacionais

Para o treinamento, *pruning*, quantização e validação dos modelos, é utilizada a plataforma de computação *Google Colab*, que fornece acesso a um ambiente de execução remoto otimizada para o treinamento de modelos de IA. O *hardware* disponibilizado pela plataforma é uma CPU de 2 *cores*, com 12 Gb de RAM, e um acelerador GPU Nvidia T4, com 16Gb de memória, ou uma Nvidia K80, de 12 Gb de memória RAM, segundo a disponibilidade [1].

3.2 Bases de dados utilizada

A base de dados aberta PCB Defect Dataset (DING *et al.*, 2019), utilizada no decorrer do trabalho, é composta por 12 PCBs distintas, com a inclusão de defeitos feita de forma artificial (por meio da ferramenta Photoshop), totalizando 693 imagens distintas, em formato *jpg*. Cada imagem é acompanhada por um documento *xml*, contendo as caixas delimitadoras das instâncias de defeitos (no formato PASCAL VOC). A base contém 2.953 instâncias de defeitos, distribuídas em 6 categorias distintas (conforme a Tabela 3) [2].

Defeito	Número de Imagens	Número de defeitos
missing hole	115	497
mouse bite	115	492
open circuit	116	482
short circuit	116	491
spurious copper	115	488
spur	116	503

Tabela 3 – Distribuição das classes de defeito nas imagens brutas.
Fonte - o Autor, 2023.

A Figura 5 apresenta um exemplo de imagem bruta.

[1] O serviço Colab permite a visualização do ambiente de execução em uso, mas não a sua escolha

[2] Os nomes das classes de defeito não serão traduzidos para manter a coerência com os rótulos originais da base de dados.

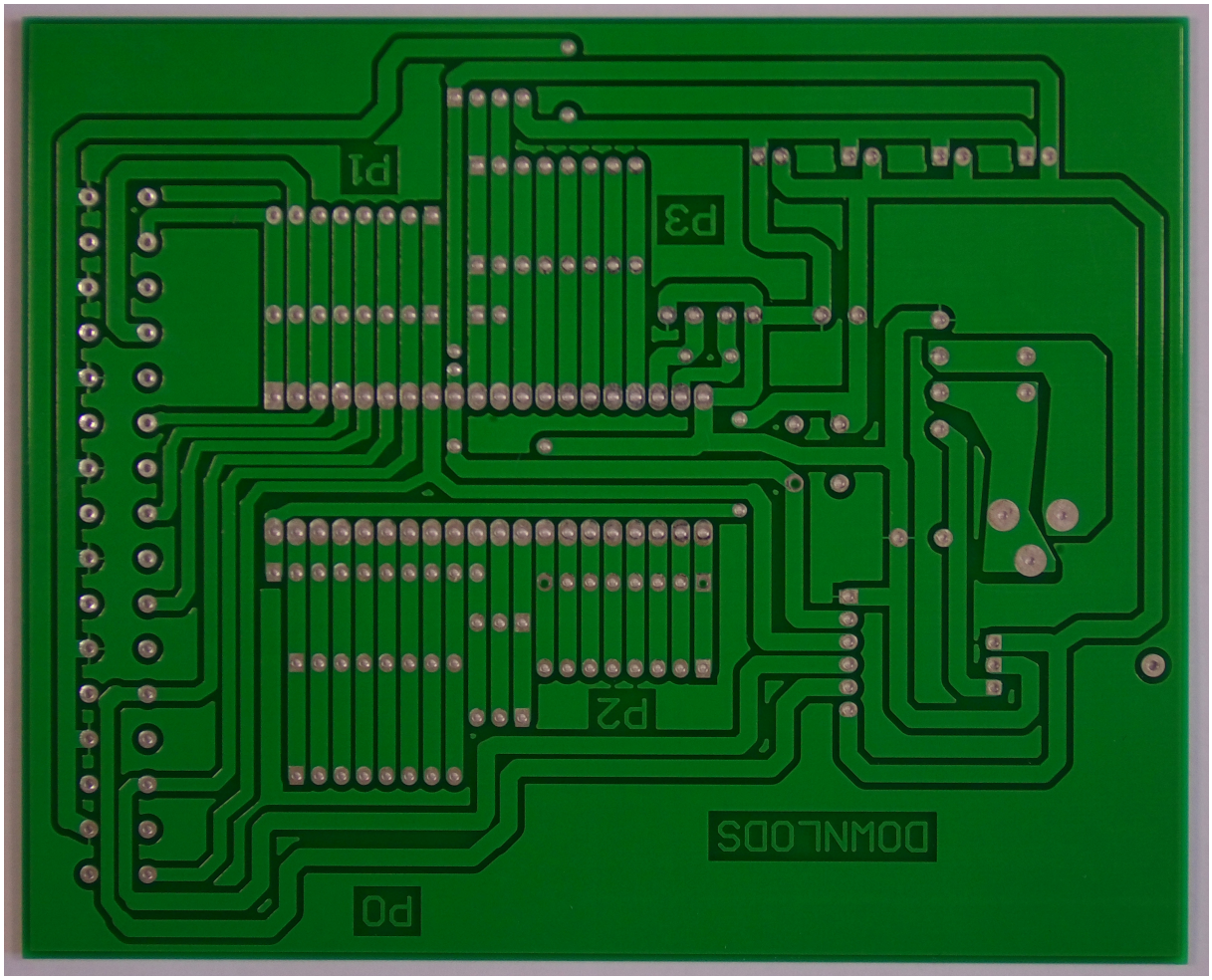

Figura 5 – Exemplo de imagem bruta
Fonte - (DING *et al.*, 2019).

Esta base de dados apresenta algumas limitações, entre elas, o número reduzido de imagens e de instâncias de defeitos, e o fato das imagens não terem seu tamanho padronizado.

A fim de contornar estas limitações, a base de dados comporta um pré-tratamento das imagens, constituído em duas etapas:

- Recorte das imagens em sub-imagens de dimensão 600×600 píxeis,
- *Data augmentation* por meio da aplicação de rotações aleatórias nas imagens.

A Figura 6 apresenta um exemplo de imagem após a aplicação do pré-tratamento, assim como os bounding boxes.

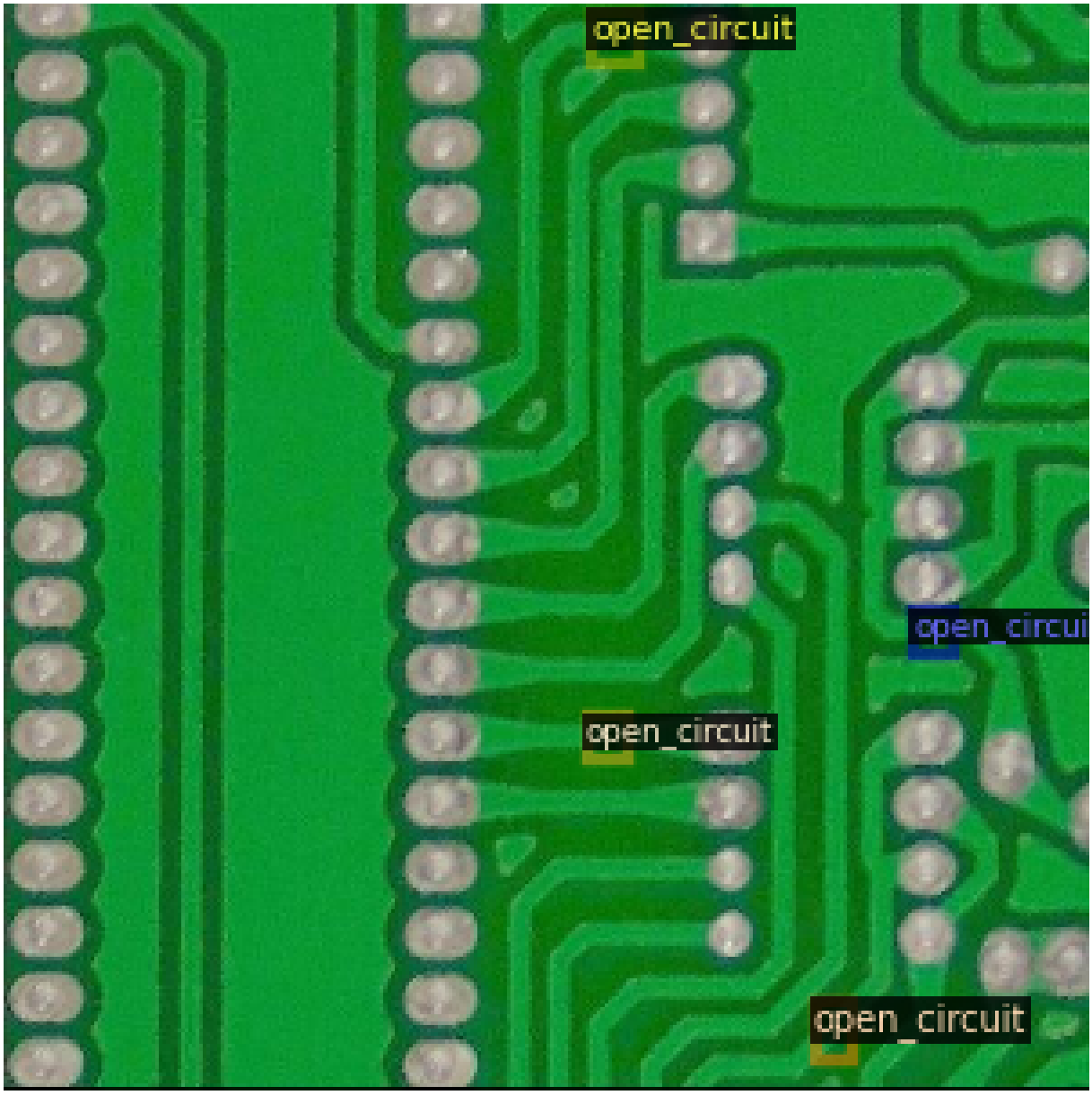

Figura 6 – Exemplo de imagem tratada, apresentando instâncias de defeito circuito aberto.
Fonte - (DING *et al.*, 2019).

A base de dados tratada comporta um total de 10.668 imagens (acompanhadas dos respectivos arquivos *xml*) e 21.664 defeitos.

3.2.0.1 Análise da base de dados

Como etapa preliminar e anterior ao treinamento dos modelos, será realizada uma análise da base de dados, analisando-se a distribuição das instâncias de defeitos e a distribuição espacial de cada instância nas imagens tratadas. Também será traçado o correlograma entre os rótulos nas diferentes imagens, a fim de analisar se as instâncias de defeito estão correlacionadas entre si, em termos de posição ou classe dos defeitos (indicando falta de diversidade nos dados) ou não.

3.2.0.2 Divisão em conjuntos de treinamento, validação e teste

A base de dados já é disponibilizada com a divisão em conjuntos de treinamento, validação e testes. A fim de permitir a comparação com demais trabalhos utilizando essa mesma base, a divisão foi mantida no decorrer do desenvolvimento do projeto.

A base PCB Defect Dataset é dividida da seguinte forma:

- Conjunto de treinamento: composto por 5120 imagens (48% da amostra total);
- Conjunto de validação: composto por 3414 imagens (32% da amostra total);
- Conjunto de teste: composto por 2134 imagens (20% da amostra total).

3.3 Seleção de modelos

A seleção do modelo final, a ser embarcado e testado no Raspberry Pi, foi realizada em três etapas. O procedimento e as métricas de escolha utilizadas em cada uma delas são descritas nesta seção. O procedimento detalhado adotado na execução de cada etapa será descrito nas seções seguintes.

3.3.1 Seleção de topologias e treinamento dos modelos

Os modelos avaliados serão de topologia Tiny-YOLOv4,YOLOv5 e Faster R-CNN, pelo fato de serem modelos relativamente recentes e de desempenho elevado, porém já contando com ampla utilização em trabalhos acadêmicos e na indústria, inclusive na bibliografia estudada. Além disso, a documentação para estes modelos, implementados nos *frameworks* Darknet (Tiny-YOLOv4) e PyTorch (demais topologias), é abrangente. Finalmente, a existência de ferramentas permitindo o *pruning* e quantização destas topologias as torna adequadas para implementações embarcadas.

Nesta etapa, os três modelos serão avaliados primariamente pela métrica mAP(IoU = 0,5), porém, também serão considerados, de forma holística, os critérios da matriz de confusão para cada uma das classes, a curva ROC e a latência média de previsão.

Devido ao tempo de treinamento elevado, atingindo 30 horas (no caso da topologia Faster R-CNN), não foi possível realizar múltiplos treinamentos para cada topologia de modelo, limitando o escopo do trabalho. A semente das variáveis aleatórias foi fixada, de modo que diferentes execuções de treinamento e inferência obtenham o mesmo resultado, apesar de que desta forma a variabilidade decorrente da aleatoriedade durante a etapa de treinamento não será explorada.

3.3.2 Pruning

Dando sequência a etapa anterior, cada um dos modelos treinados, quando possível, será submetido ao *pruning*, com diferentes níveis de esparsidade final (de 0 a 1, com passo de 0.1). A topologia Tiny-Yolov4 não é compatível com o *pruning*, então será feita apenas a quantização.

Será considerada, para cada uma das topologias submetidas ao *pruning*, a Curva de Pareto entre as métricas de desempenho de detecção (mAP) e de complexidade do modelo (número de parâmetros), permitindo avaliar o impacto do *pruning* no desempenho de detecção.

3.3.3 Quantização

Para cada uma das topologias otimizadas e treinadas, e cada nível de *pruning*, serão avaliados diferentes tipos de quantização (ponto flutuante 32, 16, int8).

Assim como na etapa anterior, serão traçadas as Curvas de Pareto considerando as métricas de desempenho de detecção (mAP), esparsidade do modelo e tipo de quantização. As Curvas de Pareto possuem o propósito de permitir a seleção do modelo que melhor se adeque aos requisitos de mAP(IoU=0,5) e latência de inferência para cada aplicação.

3.4 Pré-processamento dos dados

3.4.1 Adequação das anotações

É necessário converter o formato das anotações das bases abertas para o formato padrão de cada modelo.

O formato YOLO (utilizado nos modelos Tiny-YOLOv4 e YOLOv5) exige que cada linha do arquivo de anotações contenha, nesta ordem, a classe, coordenadas do centro da caixa (x e y, medidas a partir da extremidade esquerda superior) e dimensões da caixa (largura e altura), normalizados.

O modelo Faster R-CNN, por sua vez, exige dados no formato COCO, onde as instâncias de objetos são anotadas pelas coordenadas do canto inferior esquerdo (x e y, medidas a partir da extremidade esquerda inferior) e dimensões da caixa (largura e altura), em valor absoluto (em píxeis).

3.5 Treinamento do modelo no conjunto de dados PCB Defect Dataset

Cada classe de modelos permite o ajuste de uma gama diferente de hiperparâmetros de treinamento (incluindo o otimizador utilizado, função de perda, número de épocas/iterações de treinamento, quantidade e tamanho dos lotes).

Cada topologia permite, igualmente, a utilização de diferentes técnicas de *data augmentation*, a fim de introduzir maior variabilidade nos dados de treinamento.

O processo de treinamento para cada classe de modelo será descrito em maior detalhe nas subseções seguintes.

3.5.0.1 YOLOv5

A topologia de modelos de detecção YOLOv5, baseadas no back-end PyTorch, foi considerada devido ao seu bom desempenho de detecção (mAP medido nas bases de dados COCO e Pascal-VOC), sua adoção em larga escala e sua portabilidade, com suporte a implementações embarcadas nos formatos TFLite e ONNX (ULTRALYTICS, 2022).

Devido ao tempo de treinamento elevado do modelo (8 horas, com a utilização de GPU), não foi possível realizar o ajuste por evolução dos hiperparâmetros do modelo, sendo utilizados os parâmetros indicados na documentação do pacote YOLOv5 para bases de dados de extensão média.

Desta forma, o treinamento foi realizado por *transfer learning*, utilizando os pesos pré-treinados yolov5s. Os pesos da camada extratora de características, correspondente ao *backbone* da rede (as 10 primeiras camadas convolucionais) foram congelados, com o treinamento sendo efetuado apenas para as camadas finais do modelo.

As técnicas de *data augmentation*, implementadas de forma padrão na biblioteca e servindo de suplemento às técnicas já aplicadas pelos autores da base PCB Defect Dataset (DING *et al.*, 2019). As técnicas são aplicadas de forma probabilística. Em transformações binárias, cada imagem do conjunto de treinamento possui probabilidade p de ser submetidas à transformação. Em transformações que dependem de parâmetros contínuos, como a translação e a rotação, o parâmetro x representa o valor máximo atingido pela transformação (em fração das dimensões da imagem e em graus, respectivamente), e o valor aplicado a cada imagem obedece a uma distribuição uniforme $U[-x, x]$. As transformações aplicadas são listadas na Tabela 4.

Transformação	Tipo de parâmetro	Valor
Aumento de tonalidade	fração	0.015
Saturação	fração	0.7
Aumento de valor	fração	0.4
Rotação	grau	0.0
Translação	fração	0.1
Mudança de escala	ganho	0.9
Torção	grau	0.0
Mudança de perspectiva	fração	0.0
Espelhamento vertical	probabilidade	0.0
Espelhamento horizontal	probabilidade	0.5
Mosaico	probabilidade	1.0
Mistura	probabilidade	0.1
Segmentação (cópia e colagem)	probabilidade	0.1

Tabela 4 – Parâmetros de *data augmentation* do modelo YOLOv5
Fonte - o Autor, 2023

A Figura 7 demonstra uma série de imagens do conjunto de treinamento após a aplicação das técnicas de *data augmentation*.

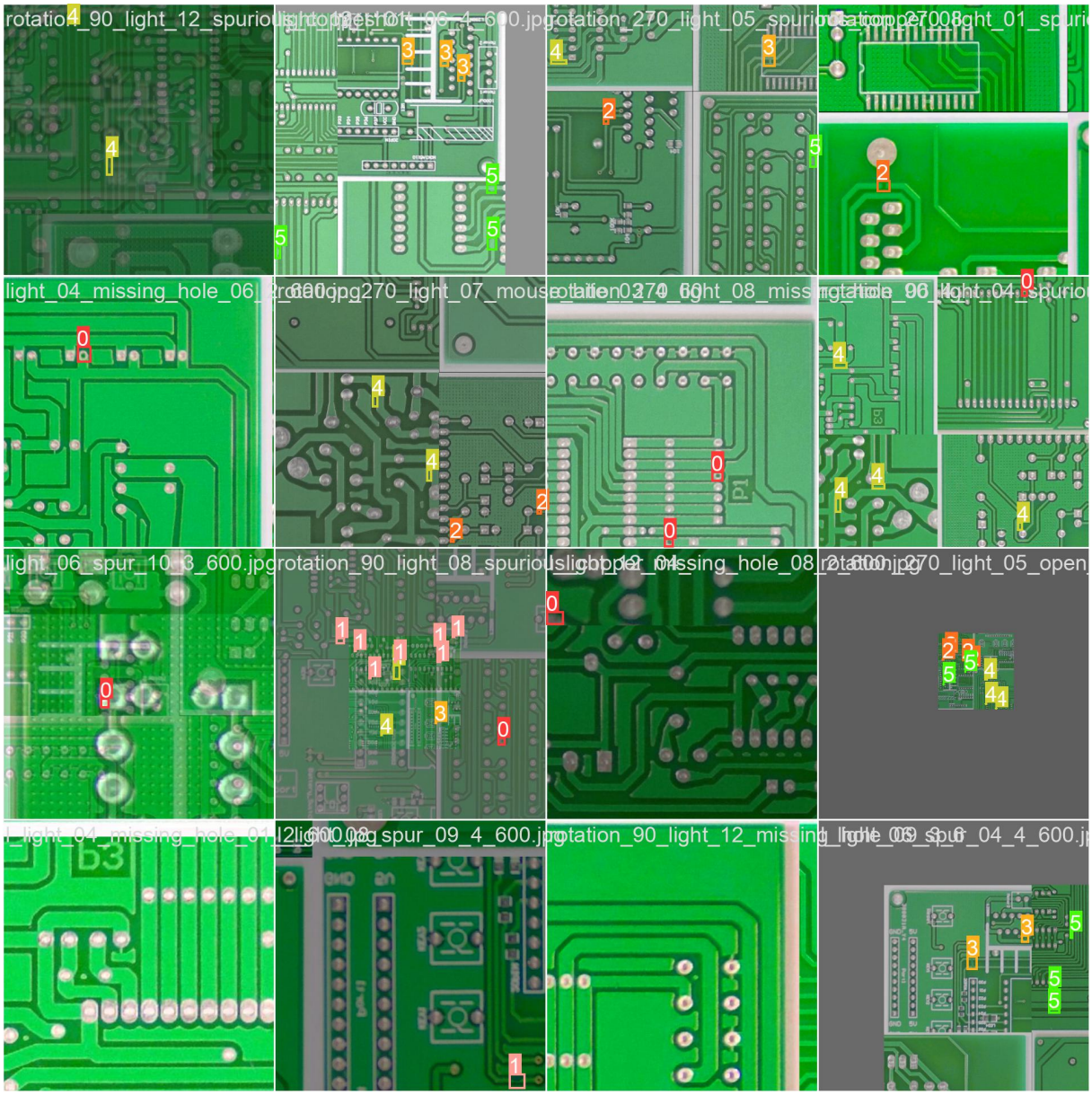

Figura 7 – Exemplo de imagens após aplicação de *data augmentation* . **Fonte -** (DING *et al.*, 2019).

O treinamento foi realizado em 100 épocas, com tamanho dos *batches* de 128 amostras cada.

3.5.0.2 Faster R-CNN

A topologia de modelos Faster R-CNN foi considerada devido ao bom desempenho que ela apresenta na bibliografia estudada. A implementação escolhida foi a da biblioteca Detectron2, baseada no *framework* PyTorch (WU *et al.*, 2019).

Devido ao tempo de treinamento elevado do modelo (30 horas), também não foi possível realizar o ajuste por evolução dos hiperparâmetros do modelo, sendo utilizados os valores indicados na documentação da biblioteca.

As técnicas de *data augmentation*, implementadas no *framework* Detectron2, são descritas na Tabela 5. A mudança de escala sempre é aplicada (não listada na tabela por este motivo), para obter um tamanho de imagem mínimo de 640 píxeis e máximo de 1333 (uma exigência do *framework* utilizado).

Transformação	Tipo de parâmetro	Valor
Espelhamento horizontal	probabilidade	0.5
Rotação	grau	0.1
Translação	fração	0.1

Tabela 5 – Parâmetros de *data augmentation* do modelo Faster R-CNN
Fonte - o Autor, 2023

O treinamento foi realizado 10.000 iterações, com tamanho dos *batches* de 64 amostras cada. O modelo foi inicializado com os pesos já treinados na base de dados COCO.

3.5.0.3 Tiny-YOLOv4

A topologia Tiny-YOLOv4 também foi considerada devido a sua adaptabilidade para aplicações em plataformas de hardware limitado. Trata-se de uma adaptação da topologia YOLOv4 no *framework* Darknet, porém com um número reduzido de componentes, diminuindo seu uso de memória e sua latência de previsão. O treinamento foi realizado em 12.000 iterações, com tamanho dos *batches* de 64 amostras cada. O modelo foi inicializado com os pesos pré-treinados na base de dados COCO, porém todas as camadas foram re-treinadas.

3.6 Pruning

A técnica de *pruning* utilizada, tanto no modelo YOLOv5 como no Faster R-CNN, é o *pruning* não estruturado. Nela, uma determinada proporção dos pesos do modelo é aleatoriamente selecionada e modificada, para que o valor final do peso seja anulado (ou seja, esse peso deixa de influenciar na saída da camada).

Foi utilizada o módulo torch.nn.utils.prune, implementado como parte da biblioteca PyTorch. O *pruning* foi aplicado apenas nas camadas de tipo nn.Conv2d, pois elas contêm a maioria dos parâmetros treináveis do modelo, entre pesos e vieses (6.987.136 de 7.035.811 para o modelo YOLOv5s, 83.770.560 de 104.405.230 para o Faster R-CNN). A técnica de *pruning* escolhida foi o *pruning* não-estruturado segundo a métrica L1 (os pesos de menor valor absoluto são eliminados), por meio da função *prune.l1_unstructured* (MICHELA,).

A escolha pelo *pruning* não-estruturado considera seu menor impacto no desempenho de detecção do modelo e a redução no espaço de armazenamento ocupado pelos

pesos comprimidos do modelo (tendo em vista o armazenamento limitado na plataforma Raspberry Pi).

Cada método de *pruning* foi avaliado para níveis de esparsidade variando entre 0 e 1, com passo de 0,1. Caso o modelo passe a fornecer saída nula ou o mAP(IoU=0,5) atinja valor inferior a 0,01, o teste é interrompido.

Os critérios avaliados nesta etapa são a relação esparsidade vs mAP(IoU=0,5) e esparsidade vs latência de previsão, calculados no conjunto de validação.

3.7 Quantização

Esta etapa ocorre imediatamente na sequência do *pruning*. Para cada modelo reduzido obtido na etapa anterior, foi realizada a quantização e conversão em um modelo embarcado nos formatos ONNX (YOLOv5 e Faster R-CNN) e TFLite (YOLOv5 e Tiny-YOLOv4).

Os pacotes TFLite (ABADI *et al.*, 2015) e ONNX (DEVELOPERS, 2021) permitem a quantização de um modelo Tensorflow/PyTorch com três diferentes níveis de precisão:

- ponto flutuante 32 bits
- ponto flutuante 16 bits (precisão mista [3])
- int 8

Os dois pacotes utilizados também incluem um ambiente de execução, permitindo utilizar o modelo quantizado em diferentes plataformas, incluindo sistemas com baixo poder de processamento.

Todas as combinações possíveis de modelo e tipo de quantização foram avaliadas. Contudo, alguns tipos de modelo são incompatíveis com certos níveis de quantização, uma vez que sua implementação interna é diferente.

Novamente, a implementação desta conversão e quantização se dá separadamente para cada classe de modelos, devido às suas especificidades. Os critérios avaliados nesta etapa são a relação esparsidade vs modo de quantização vs mAP(IoU=0,5), esparsidade vs modo de quantização vs latência de previsão e esparsidade vs modo de quantização vs tamanho do modelo, calculados no conjunto de validação. O tamanho do modelo é definido como o espaço de armazenamento, mensurado em bytes, ocupado pelos pesos do modelo quantizado, comprimidos no formato zip (CHIAO, 2023).

[3] O modelo contém parâmetros em 16 e 32 bits, o formato de 32 bits é utilizado apenas nos parâmetros cuja redução da precisão impactaria severamente a *accuracy* do modelo.

3.8 Análise dos resultados finais e comparação com o Estado da Arte

Nesta etapa, os resultados obtidos com as topologias estudadas, antes do *pruning* e quantização, são analisados e comparados aos resultados obtidos pelos estudos representando o Estado da Arte, em termos de mAP(IoU=0,5) calculado no conjunto de teste. Devido à diferença nos ambientes de execução e *hardware* utilizados, não é considerada a comparação da latência de previsão com o Estado da Arte. Na sequência, a comparação com o Estado da Arte é repetida com os modelos quantizados (no caso de quantização bem-sucedida), sendo feita a escolha de um modelo por topologia segundo o critério de mAP(IoU=0,5) máximo.

Para fins de validar a robustez do modelo face à entrada de PCBs com diferentes formatos e tamanho, também são realizados testes com as placas inteiras que compõem a base PCB Defect Dataset, sem seccionamento. O conjunto de testes utilizado nesta avaliação comporta 67 imagens, baseadas nas 12 PCBs que compõem a base. As imagens possuem dimensões distintas, necessitando, portanto, a conversão para o formato de entrada (de 608 x 608 píxeis para os modelos Tiny-YOLOv4 e 640 x 640 para o Faster R-CNN), exceto no caso da topologia YOLOv5, que aceita imagens em formato maior (de 1280 x 1280 píxeis).

3.9 Estudo de caso: Implementação do sistema de detecção embarcado na plataforma Raspberry Pi

Esta etapa compreende a implementação e os testes realizados na plataforma Raspberry Pi. O modelo quantizado, cuja topologia, esparsidade e método de quantização foram escolhidos em função dos resultados anteriores, é adaptado para realizar a inferência das imagens do conjunto de testes rodando de forma autônoma no Raspberry Pi.

3.9.1 Materiais utilizados

A plataforma embarcada utilizada para a avaliação dos modelos otimizados é um computador de placa única *Raspberry PI 3B +*, possuindo 1 Gb de memória RAM, um processador Broadcom BCM2837B0 de 64 bits 1,4Ghz e 16 Gb de armazenamento em um cartão SD.

O sistema operacional utilizado é o Raspberry Pi OS (64-bit), uma distribuição Linux. A versão da linguagem Python equipada é a 3.9. A comunicação com o computador pessoal foi efetuada por meio do protocolo ssh, e o compartilhamento de arquivos, pelo protocolo scp.

A plataforma *Raspberry PI 3B+* possui entrada de energia DC 5V. Ela oferece também portas USB e HDMI, permitindo sua conexão a um monitor, mouse e teclado e operação autônoma. A conexão com a Internet (e com o computador pessoal utilizado) se dá pelo protocolo WiFi (o *Raspberry Pi*) oferece suporte para WiFi 2,4 GHz e 5GHz, além de uma entrada Ethernet). A plataforma pode ser conectada a uma câmera por meio de uma porta serial. No entanto, a captura de imagens de PCBs e inferência sobre as mesmas não são exploradas neste trabalho.

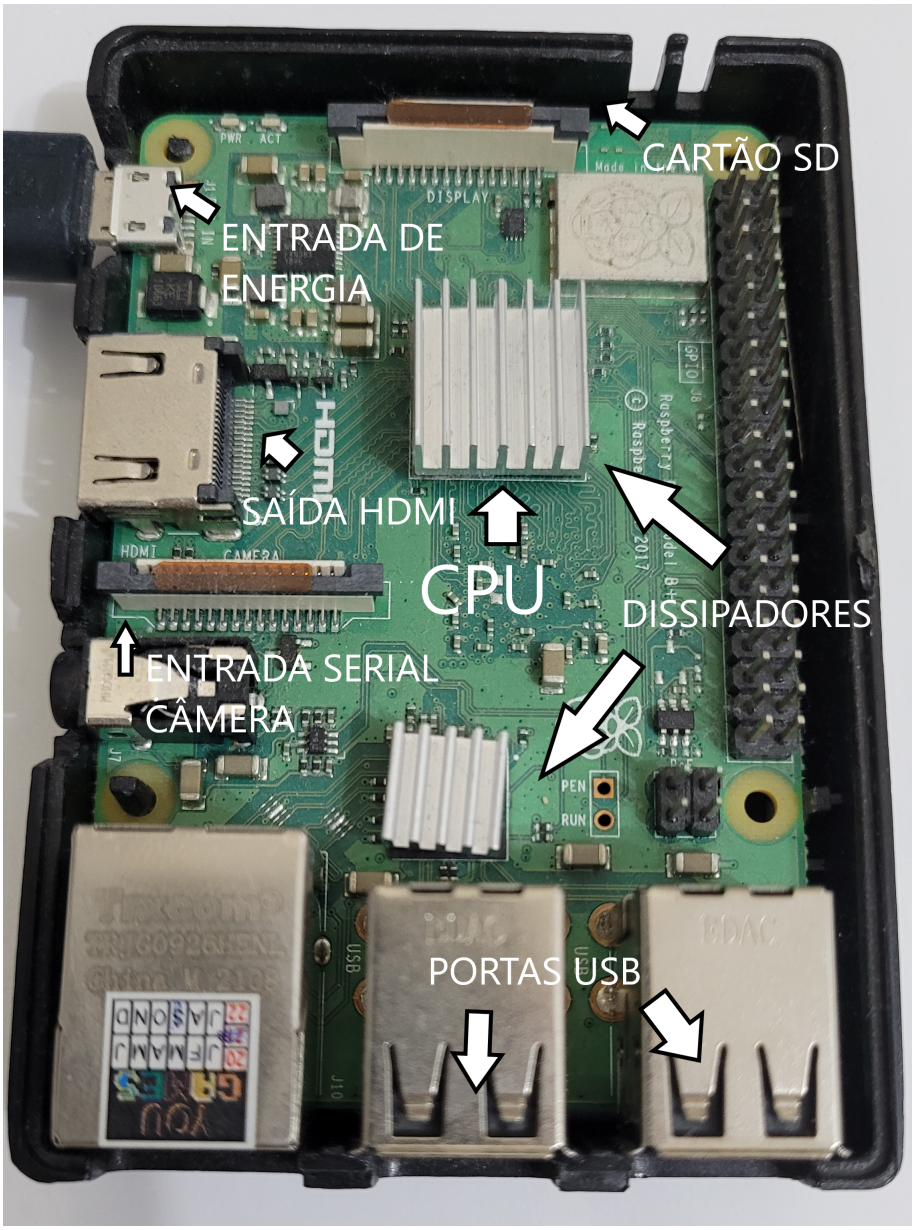

Figura 8 – Computador de placa única Raspberry Pi 3B+.
Fonte - o Autor.

3.9.2 Seleção do modelo embarcado

A seleção de um modelo quantizado final para os testes no Raspberry Pi tem o mAP(IoU=0,5) como critério de escolha, e a latência como critério de desempate. É

adotada a restrição de um tempo de processamento (em CPU, na plataforma *Google Colab*) inferior a 1 segundo.

3.9.3 Testes realizados

Nesta etapa, serão avaliados o desempenho de classificação (segundo a métrica mAP(IoU=0,5) e a latência média de previsão (compreendendo a inferência propriamente dita, somada ao tempo de pré e pós-processamento das amostras). É analisada a evolução destas métricas em função de cada etapa da adaptação do modelo ao Raspberry Pi (treinamento inicial, *pruning*, quantização e implementação).

Por fim, foi realizada a análise de utilização do *hardware*, compreendendo o estudo da memória RAM e capacidade da CPU ocupadas durante a inferência. Estes resultados serão comparados às medições efetuadas na plataforma *Google Colab*, para o mesmo modelo. Em ambos os casos, esta medida foi realizada por meio das funções *cpu_percent()* e *virtual_memory()*, da biblioteca *psutil*.

4 Resultados

Nesta seção, são apresentados os resultados obtidos a partir de ensaios com a base de dados PCB Defect Dataset, as topologias YOLOv5, Tiny-YOLOv4 e Faster-RCNN e a aplicação das técnicas de *pruning* e quantização. São igualmente apresentados os resultados obtidos após testes na plataforma Raspberry Pi 3B+.

4.1 Análise da base de dados

Foi realizada uma análise da base de dados. Foram traçados histogramas com a frequência de cada tipo de defeito, bem como a correlação entre cada classe (em termos das coordenadas e dimensões de cada instância de defeito).

A partir da Tabela 6, é possível concluir que a base de dados apresenta distribuição equilibrada entre as diferentes classes de defeitos, característica importante para evitar que, durante o treinamento, o modelo se torne enviesado para uma determinada classe. O histograma (Figura 9) corrobora esta observação.

Defeito	Número de Imagens	Número de defeitos
missing hole	1832	3612
mouse bite	1852	3684
open circuit	1740	3548
short circuit	1732	3508
spurious copper	1752	3636
spur	1760	3676

Tabela 6 – Distribuição das classes de defeito nas imagens pré-tratadas.
Fonte - (DING *et al.*, 2019).

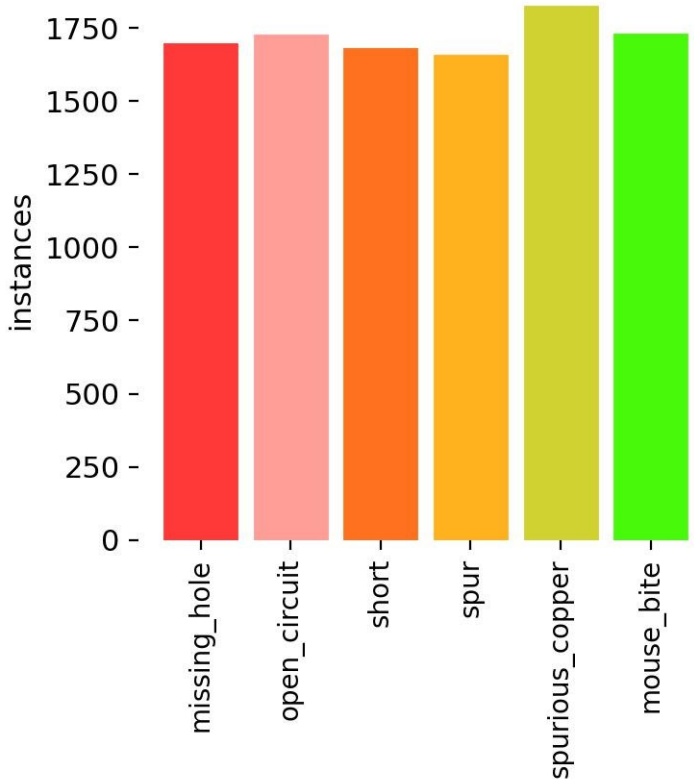

Figura 9 – Histograma dos rótulos.
Fonte - o Autor, 2023.

O correlograma (Figura 10) contém os histogramas unidimensionais e bi-dimensionais das localizações de todas as instâncias de defeito presentes nas imagens, representadas no espaço *x*, *y*, *height* e *width* (localização do canto superior esquerdo de cada caixa delimitadora ao longo dos eixos x e y, considerando como origem a extremidade superior esquerda da imagem, sua largura e altura). É possível observar que as instâncias de defeito estão bem espacialmente distribuídas ao longo das imagens, com uma distribuição aproximadamente uniforme. Também se observa que a maioria das caixas delimitadoras possuem dimensões (normalizadas) da ordem de 0,05 (correspondendo a 30 píxeis, considerando o tamanho das imagens de 600 píxeis) ou menos (Gráfico *height* vs *width*).

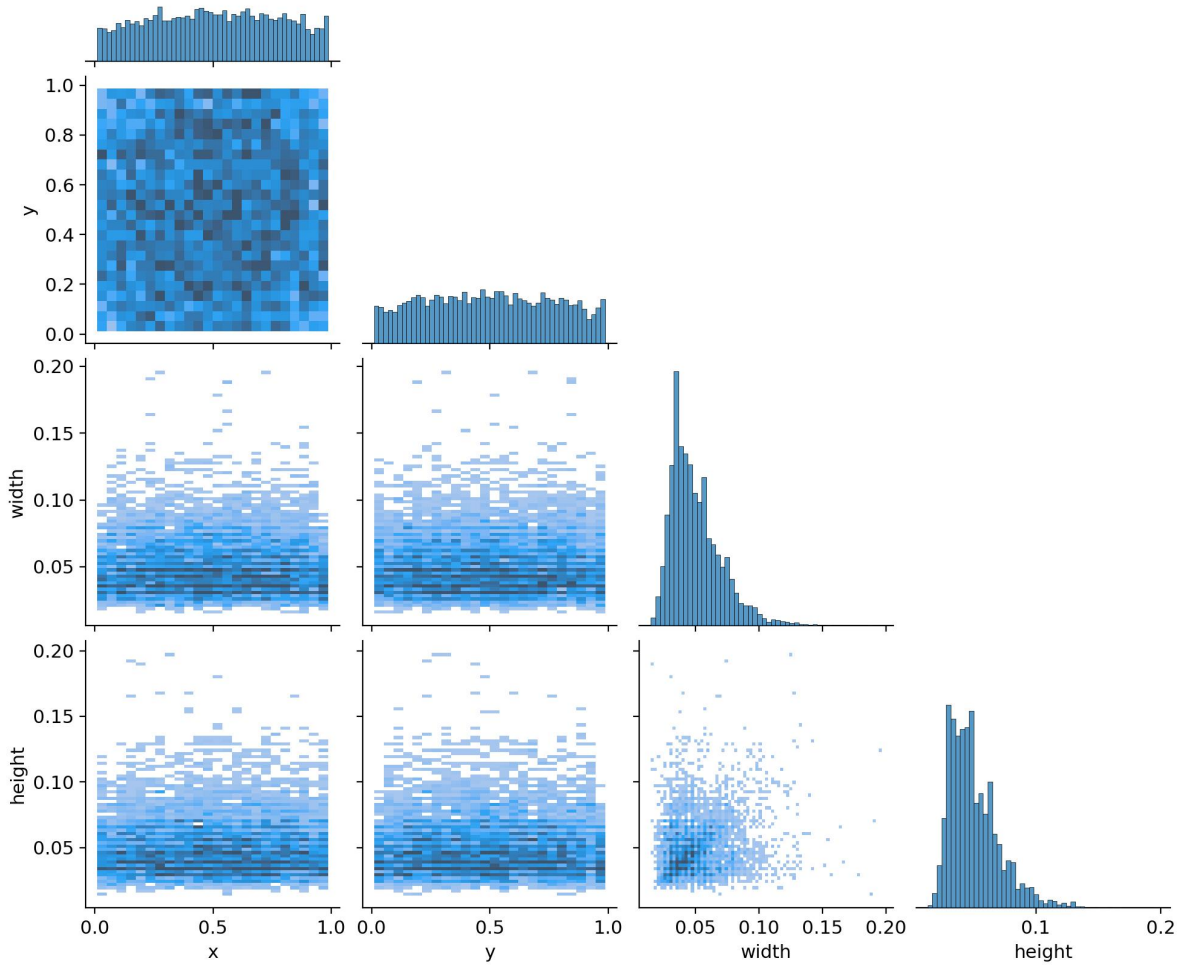

Figura 10 – Correlograma dos rótulos.
Fonte - o Autor, 2023.

Uma limitação da base de dados utilizada é o fato de cada imagem apresentar apenas um tipo de defeito, o que não corresponde com a ocorrência de defeitos na realidade. Esta limitação é, em parte, superada pelo uso de técnicas de *data augmentation*, em específico o *mixup* e mosaico, que geram combinações de diferentes imagens (podendo conter, portanto, diferentes classes de defeito).

4.2 Treinamento inicial dos modelos

Nesta seção, são descritos os resultados obtidos a partir do treinamento inicial de cada uma das topologias, assim como um comparativo entre elas.

4.2.1 Tiny-YOLOv4

Os resultados de treinamento obtidos pelo modelo Tiny-YOLOv4 são descritos na Tabela 7.

Métrica	Validação	Teste
mAP(0,5)	0,978	0,979
Tempo de inferência (por imagem) (CPU) [s]	2,727	2,708
Tempo de inferência (por imagem) (GPU) [s]	0,03603	0,03649

Tabela 7 – Resultados de validação e teste para o treinamento inicial Tiny-YOLOv4.
Fonte - o Autor, 2023.

Observa-se que as principais métricas de acerto apresentam resultado satisfatório (mAP(IoU=0,5) de 0,978 no conjunto de teste), apenas ligeiramente inferior ao obtido nos trabalhos considerados neste trabalho como pertencentes ao Estado da Arte (listados na Seção 2.4.1) (0,991, obtido por (NIU *et al.*, 2023)). O tempo de inferência utilizando um ambiente de execução equipado com GPU é aproximadamente 75 vezes inferior ao obtido apenas com CPU.

A métrica AP por classes, obtida para um *threshold* de IoU de $0,5$ e nível de confiança de $0,5$ é apresentada na Tabela 8. Observa-se que o mAP(0,5) apresenta variação em função da classe de defeito, tendo resultados inferiores para as classes *short_circuit* e *missing_hole*, e resultado superior para a classe *spurious_copper*.

Classe	mAP(0,5) de Validação	mAP(0,5) de Teste
missing_hole	0,959	0,964
mouse_bite	0,989	0,983
open_circuit	0,989	0,989
short_circuit	0,969	0,961
spurious_copper	0,991	0,991
spur	0,975	0,977

Tabela 8 – Resultados por classe para o treinamento inicial Tiny-YOLOv4.
Fonte - o Autor, 2023.

4.2.2 YOLOv5

Os resultados de treinamento obtidos pelo modelo YOLOv5 são descritos na Tabela 9.

Métrica	Validação	Teste
mAP(0,5)	0,966	0,967
Tempo de inferência (por imagem) (CPU) [s]	0,3965	0,4112
Tempo de inferência (por imagem) (GPU) [s]	0,0062	0,0064

Tabela 9 – Resultados de validação e teste para o treinamento inicial YOLOv5.
Fonte - o Autor, 2023.

Observa-se, novamente, que as principais métricas de acerto apresentam valores elevados. Em comparação à topologia Tiny-YOLOv4, a YOLOv5 apresenta latência significativamente menor, tanto com a utilização de GPU (0,0062 contra 0,03603 segundos no conjunto de validação) quanto apenas com CPU (0,3965 contra 2,727 segundos no conjunto de validação). Esta diferença pode ser explicada pelos diferentes *backends* utilizados (Darknet e PyTorch, respectivamente).

A métrica AP por classes, obtida para um *threshold* de IoU de $0,5$ e nível de confiança de $0,5$ é apresentada na Tabela 10.

Classe	mAP(0,5) de Validação	mAP(0,5) de Teste
missing_hole	0,993	0,988
mouse_bite	0,967	0,956
open_circuit	0,970	0,974
short_circuit	0,957	0,971
spurious_copper	0,962	0,968
spur	0,949	0,944

Tabela 10 – Resultados por classe para o treinamento inicial YOLOv5
Fonte - o Autor, 2023.

A curva de perdas do modelo durante o treinamento (Figura 11), calculadas sobre os conjuntos de treinamento e validação, mostram a evolução do modelo. Observa-se que não há *overfitting* no conjunto de treinamento e, na realidade, as perdas de validação são ligeiramente inferiores durante grande parte do treinamento (o que pode ser decorrência da aplicação de técnicas de regularização no conjunto de treinamento). Desta forma, é possível concluir que o modelo apresenta boa generalização. Observa-se que, no fim do treinamento, as perdas apresentam ainda uma tendência de lento decréscimo.

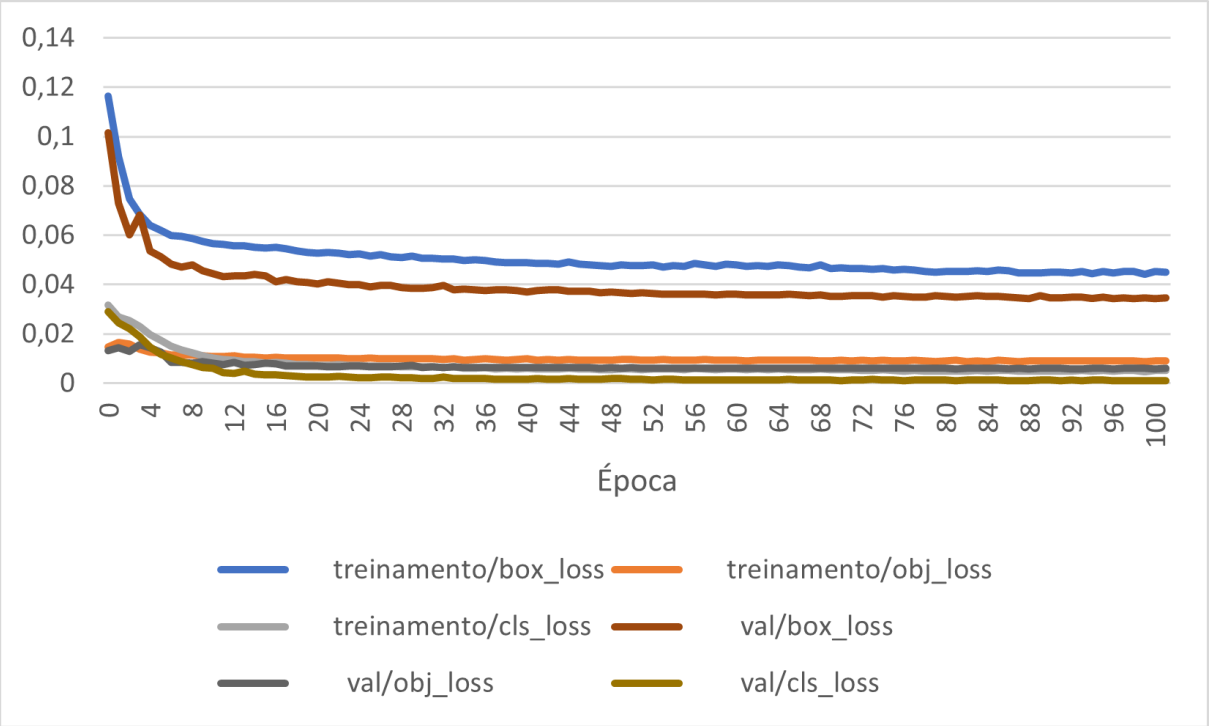

Figura 11 – Perdas de classificação, caixa delimitadora e confiança de presença de objeto (respectivamente, *class*, *box* e *obj loss*) nos conjuntos de treinamento e validação durante o treinamento do modelo, em função da época.
Fonte - o Autor, 2023.

4.2.3 Faster R-CNN

Os resultados de treinamento obtidos pelo modelo Faster R-CNN são descritos na Tabela 11.

Métrica	Validação	Teste
mAP(0,5)	0,971	0,968
Tempo de inferência (por imagem) (GPU) [s]	0,1841	0,1836

Tabela 11 – Resultados de validação e teste para o treinamento inicial Faster R-CNN.
Fonte - o Autor, 2023.

A mAP(IoU=0,5) apresentado pela topologia Faster R-CNN é bastante próximo ao obtido pelas duas topologias anteriores (todas situadas entre 0,96 e 0,98) e pelos trabalhos considerados como pertencentes ao Estado da Arte. Observa-se, porém, que a latência média de previsão é bastante elevada quando comparada à dos outros modelos (utilizando GPU) (0,1841 segundos no conjunto de validação, comparado a 0,03603 para a topologia Tiny-YOLOv4 e 0,0062 para a YOLOv5). Não foi possível realizar esta comparação em CPU, pois o ambiente de execução do pacote *Detectron2* exige suporte para GPU.

A métrica AP por classes, obtida para um *threshold* de IoU de $0,5$ e nível de confiança de $0,5$ é apresentada na Tabela 12.

Classe	mAP(0,5) de Validação	mAP(0,5) de Teste
missing_hole	0,993	0,990
mouse_bite	0,978	0,972
open_circuit	0,956	0,972
short_circuit	0,957	0,955
spurious_copper	0,983	0,980
spur	0,962	0,962

Tabela 12 – Resultados por classe para o treinamento inicial Faster R-CNN.
Fonte - o Autor, 2023.

4.3 Pruning dos modelos

4.3.1 YOLOv5

Os resultados da avaliação do modelo YOLOv5 submetido ao *pruning* são sumarizados na Tabela 13. Após o nível de esparsidade das camadas convolucionais de 0,6 ser ultrapassado, a saída do modelo passa a ser nula, e o experimento é interrompido. Observa-se que o tempo de inferência médio por imagem apresenta pouca variação em função do nível de *pruning*, e não existe uma correlação aparente. Isto decorre do fato de não existirem otimizações, ao nível do *software* e *hardware* utilizados no experimento, para operações com matrizes esparsas (GHOLAMI *et al.*, 2021).

Esparsidade	mAP(IoU=0,5)	Latência média (GPU) [s]
0	0,966	0,0063
0,1	0,967	0,0067
0,2	0,956	0,0068
0,3	0,898	0,0067
0,4	0,643	0,0068
0,5	0,093	0,0067
0,6	0,015	0,0065

Tabela 13 – Resultados para o *pruning* YOLOv5.
Fonte - o Autor, 2023.

A Figura 12 permite visualizar a evolução da taxa de acerto do modelo (mAP(IoU=0,5)) em função do nível de esparsidade. Observa-se que, inicialmente, a taxa de acerto não apresenta variação significativa. Ocorre mesmo um leve aumento no mAP(IoU=0,5), possivelmente em decorrência da redução do *overfitting* com decréscimo no número de parâmetros. A partir do nível de esparsidade 0,3, no entanto, o modelo passa a apresentar redução significativa, e rápida, no nível de taxa de acerto.

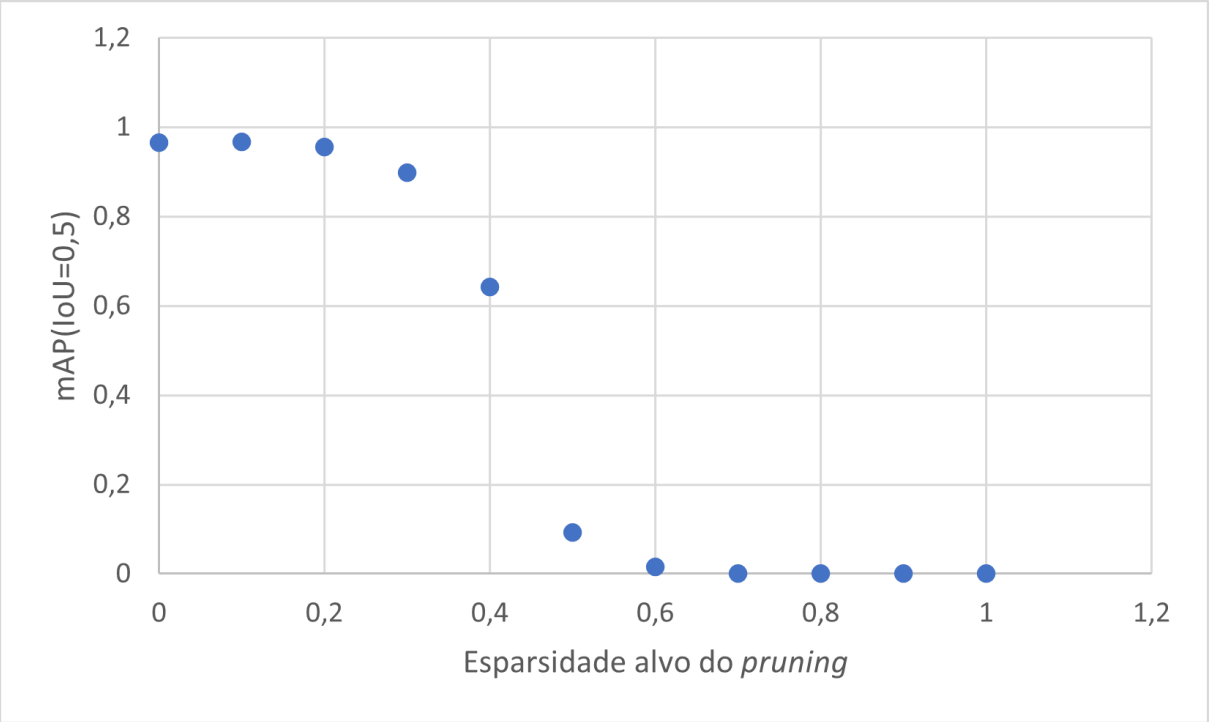

Figura 12 – Evolução do mAP(IoU=0,5) para YOLOv5, em função da esparsidade alvo do *pruning*.

Fonte - o Autor, 2023.

4.3.2 Faster R-CNN

Assim como na topologia anterior, os resultados do *pruning* na topologia Faster R-CNN, sumarizados na Tabela 14, demonstram pouco impacto do procedimento na latência de inferência, apenas uma ligeira redução correspondente a esparsidade. O experimento é interrompido com um nível de esparsidade de 0,7.

Esparsidade	mAP(IoU=0,5)	Latência média (GPU) [s]
0	0,971	0,1664
0,1	0,973	0,1651
0,2	0,971	0,1652
0,3	0,969	0,1649
0,4	0,964	0,1645
0,5	0,925	0,1640
0,6	0,436	0,1630
0,7	0,043	0,1628

Tabela 14 – Resultados para o *pruning* Faster R-CNN.

Fonte - o Autor, 2023.

A Figura 13 permite visualizar a evolução da taxa de acerto do modelo (mAP(IoU=0,5)) em função do nível de esparsidade. Observa-se que o desempenho do modelo é menos impactado em decorrência do nível de esparsidade, quando comparado a topologia YOLOv5. A taxa de acerto do modelo só passa a diminuir rapidamente a partir de um nível de

esparsidade de 0,5. Contribui para este comportamento o fato do *pruning* ter sido efetuado apenas nas camadas convolucionais, e a proporção de parâmetros contida nestas camadas é inferior na topologia Faster R-CNN (em relação a YOLOv5).

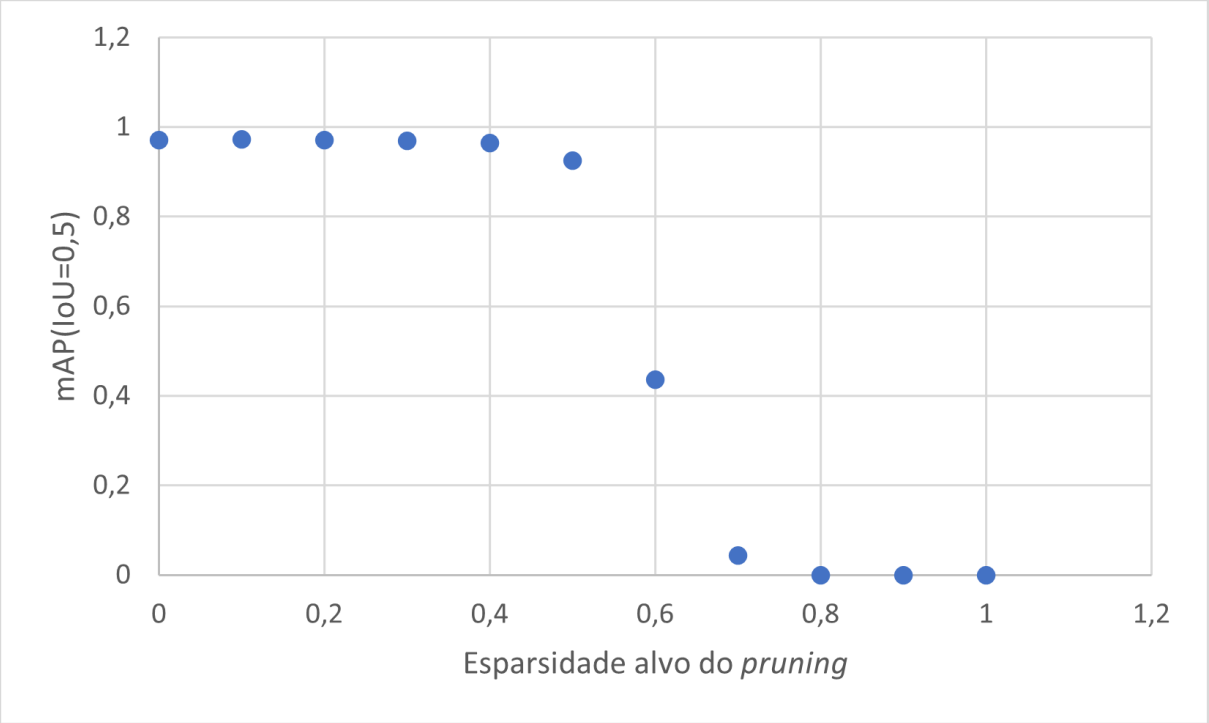

Figura 13 – Evolução do mAP(IoU=0,5) para Faster R-CNN em função da esparsidade alvo do *pruning*.

Fonte - o Autor, 2023.

Observa-se, portanto, que no caso específico estudado, o *pruning* deixa de ser vantajoso para níveis de esparsidade acima de 0,2, pois acarreta redução do mAP atingido pelo modelo sem apresentar melhorias na latência. Caso sejam empregadas otimizações em *hardware* capazes de otimizar a inferência em modelos esparsos, no entanto, o *pruning* com esparsidades superiores poderia ser uma abordagem válida.

4.4 Quantização dos modelos

4.4.1 Tiny-YOLOv4

A Tabela 15 sumariza os resultados obtidos no conjunto de validação com o modelo Tiny-YOLOv4 quantizado no formato TFLite, com precisão de 16 bits ponto flutuante, em termos de taxa de acerto, latência e tamanho do modelo. As medições foram realizadas no ambiente *Colab*, utilizando CPU. Observa-se que o mAP(IoU=0,5) apresenta um forte decréscimo em relação ao obtido com o modelo original. Possivelmente, a quantização com precisão de 32 bits ofereceria resultados melhores, porém o modelo Tiny-YOLOv4 não é compatível com este formato (assim como não é compatível com o formato int 8 bits).

mAP(0,5)	Latência média (CPU) [s]	Tamanho do modelo [Mb]
0,0032	0,731	10,93

Tabela 15 – Resultados para a quantização Tiny-YOLOv4, considerando o modelo não submetido ao *pruning*

Fonte - o Autor, 2023.

4.4.2 YOLOv5

Os resultados no conjunto de validação obtidos para a quantização, nos formatos ONNX (32 bits ponto flutuante, 16 bits ponto flutuante, int 8 bits) e TFLite (16 bits ponto flutuante, int 8 bits), são resumidos na Tabela 16. As medições foram realizadas no ambiente *Colab*, utilizando CPU. Observa-se que o *pruning* e a quantização surtem o efeito esperado no tamanho dos modelos, pois, considerando separadamente os formatos TFLite e ONNX, observa-se uma correlação negativa entre a esparsidade e o tamanho dos modelos, assim como entre o nível de bits da representação quantizada e o tamanho. Observa-se, porém, que também existe uma correlação negativa entre a taxa de acerto do modelo e o grau de *pruning*/quantização (a exceção do formato ONNX int 8 bits, com tamanho maior que a representação em 16 bits).

Modo de quantização	Esparsidade	mAP(0,5)	Latência média(CPU) [s]	Tamanho [Mb]
ONNX pf 32	0	0,966	0,360	24,21
ONNX pf 32	0,1	0,969	0,351	22,94
ONNX pf 32	0,2	0,961	0,343	21,29
ONNX pf 32	0,3	0,932	0,345	19,46
ONNX pf 32	0,4	0,662	0,409	17,47
ONNX pf 32	0,5	0,099	0,416	15,30
ONNX pf 32	0,6	0,011	0,404	12,10
ONNX pf 32	0,7	0,002	0,400	10,55
ONNX pf 16	0	0,969	0,448	49,34
ONNX pf 16	0,1	0,969	0,477	49,04
ONNX pf 16	0,2	0,962	0,491	47,59
ONNX pf 16	0,3	0,931	0,504	45,11
ONNX pf 16	0,4	0,662	0,502	41,61
ONNX pf 16	0,5	0,099	0,504	37,54
ONNX int 8	0	0,961	0,378	13,10
ONNX int 8	0,1	0,963	0,410	12,60
ONNX int 8	0,2	0,962	0,408	11,79
ONNX int 8	0,3	0,916	0,409	10,84
ONNX int 8	0,4	0,629	0,409	97,98
ONNX int 8	0,5	0,095	0,409	86,92
TFLite pf 16	0	0,969	0,629	13,10
TFLite pf 16	0,1	0,969	0,630	12,61
TFLite pf 16	0,2	0,962	0,621	11,81
TFLite pf 16	0,3	0,932	0,621	10,89
TFLite pf 16	0,4	0,662	0,623	98,69
TFLite pf 16	0,5	0,099	0,620	87,86
TFLite int 8	0	0,792	0,455	61,15
TFLite int 8	0,1	0,801	0,442	60,11
TFLite int 8	0,2	0,871	0,442	58,19
TFLite int 8	0,3	0,819	0,444	55,08
TFLite int 8	0,4	0,551	0,441	50,93
TFLite int 8	0,5	0,067	0,441	45,89

Tabela 16 – Resultados para a quantização YOLOv5
. **Fonte -** o Autor, 2023.

As Figuras 14, 15 e 16 permitem observar a evolução destas três métricas (mAP(IoU=0,5), tamanho do modelo e latência de previsão) em função da esparsidade e tipo de quantização. É possível observar que o formato TFLite apresenta, de forma geral, resultados piores nas três métricas, considerando esparsidade e precisão da quantização equivalentes. A quantização no formato 16 bits apresenta taxa de acerto ligeiramente inferior e redução significativa no tamanho do modelo, as custas, porém, de um tempo de inferência maior.

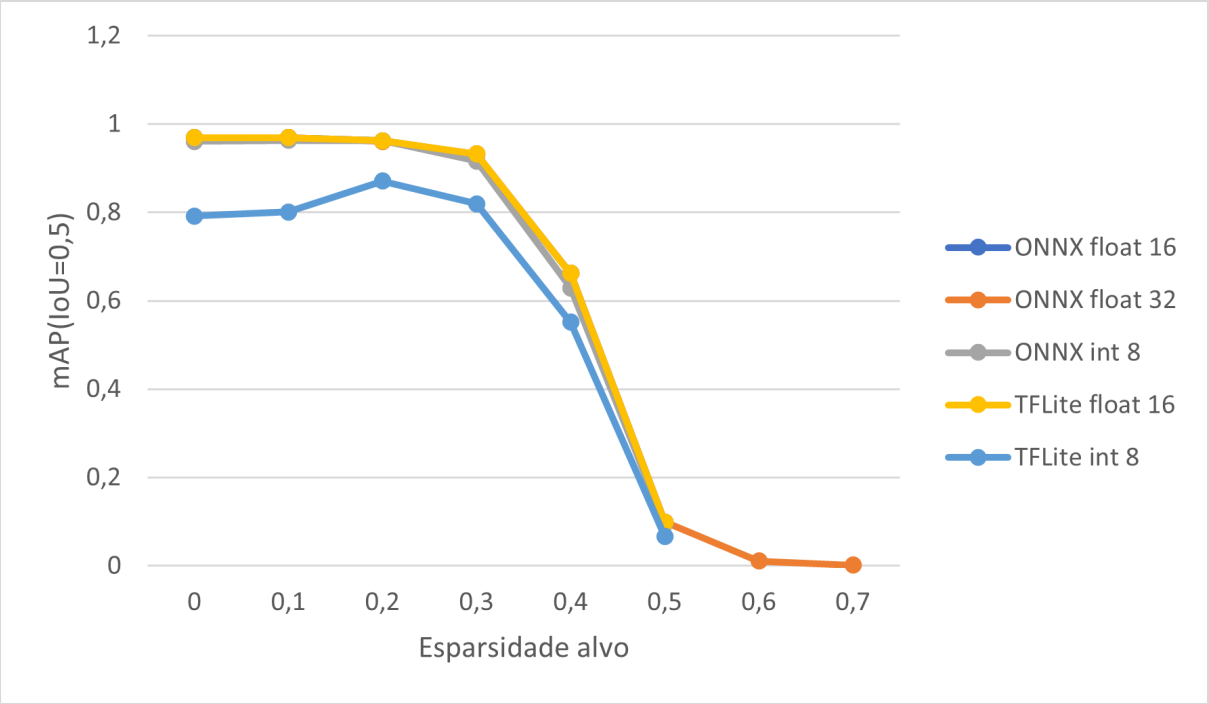

Figura 14 – Evolução do mAP(IoU=0,5) para o modelo YOLOv5 em função da esparsidade e método de quantização.

Fonte - o Autor, 2023.

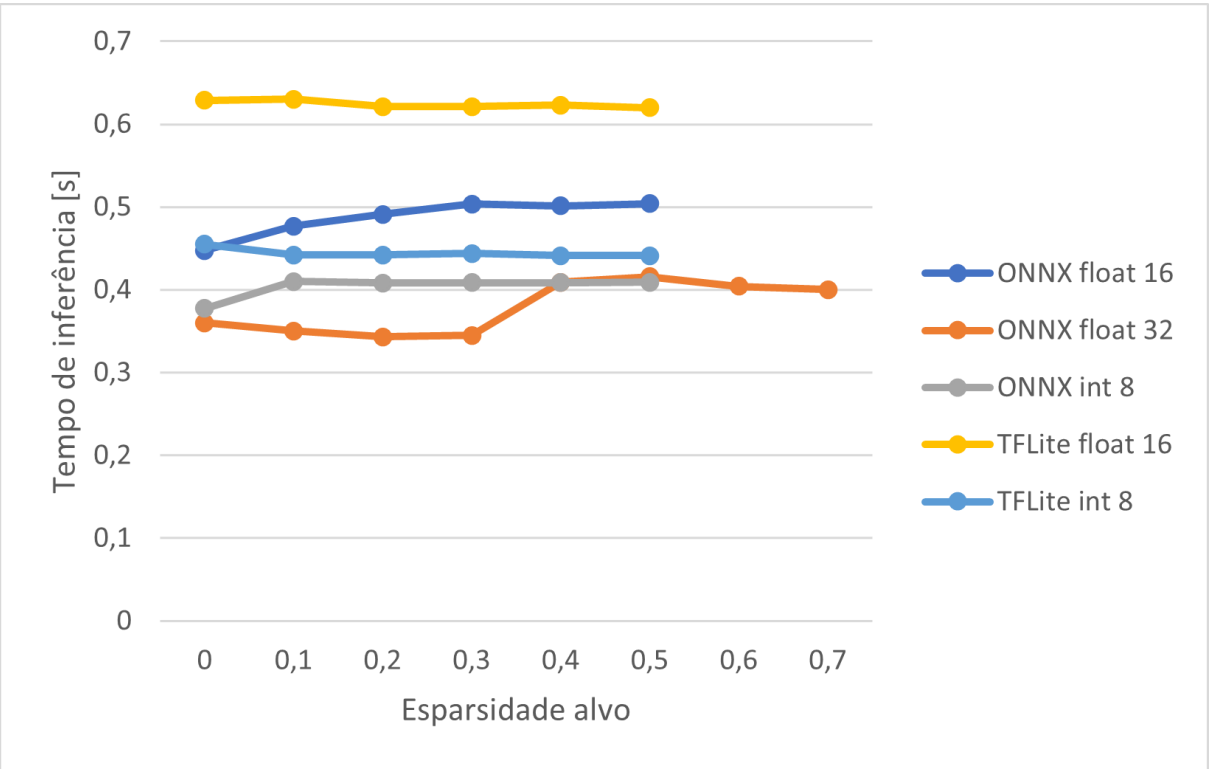

Figura 15 – Evolução do tempo de inferência para o modelo YOLOv5 em função da esparsidade e método de quantização.

Fonte - o Autor, 2023.

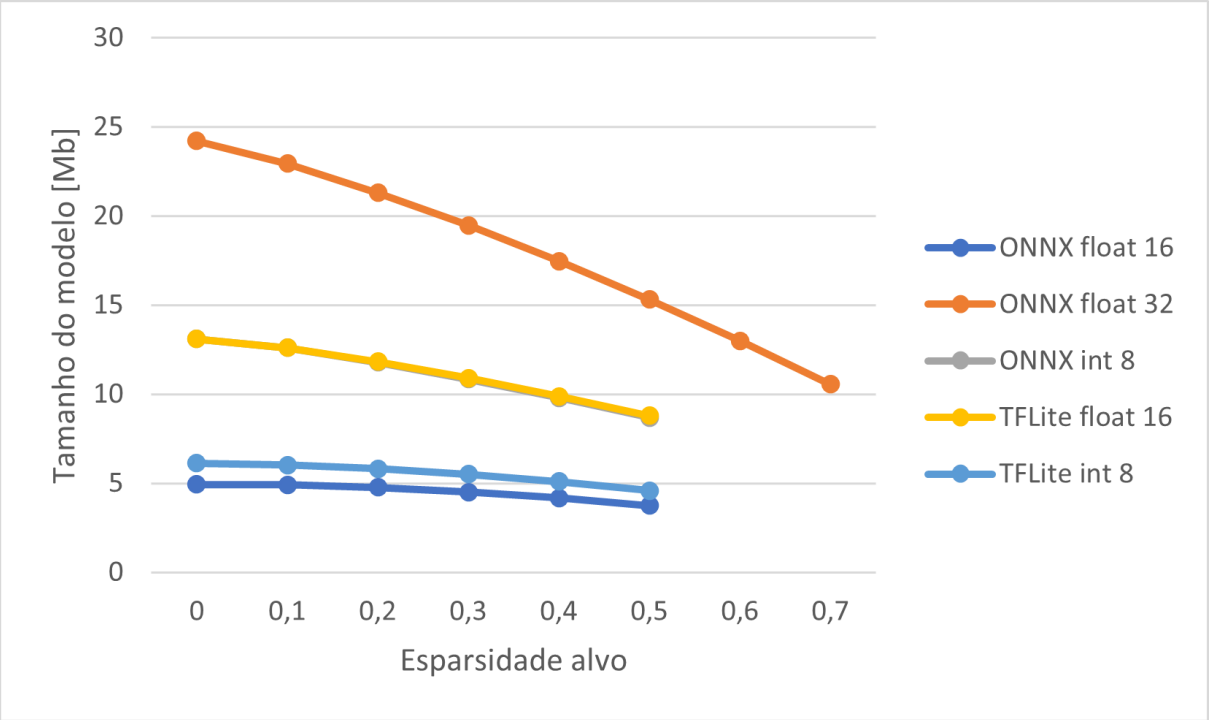

Figura 16 – Evolução do tamanho de modelo para YOLOv5 em função da esparsidade e método de quantização.

Fonte - o Autor, 2023.

4.4.3 Faster R-CNN

As medições foram realizadas no ambiente *Colab*, utilizando GPU, apenas no formato ONNX 32 bits ponto flutuante. Também foi realizada uma avaliação dos resultados utilizando CPU, porém, a alta latência de previsão (média superior a 7,83 segundos para as 100 primeiras amostras do conjunto de validação) inviabilizaram estas medições para o conjunto inteiro. O modelo convertido em formato int 8 também apresentou latência elevada (da ordem de 7 segundos), impedindo um estudo aprofundado. O formato 16 bits ponto flutuante, por sua vez, não era compatível com os módulos empregados na topologia Faster R-CNN.

Os resultados obtidos no teste são sumarizados na Tabela 17. Observa-se uma tendência de diminuição da taxa de acerto e tamanho do modelo em função da esparsidade de *pruning*. O tempo de inferência também apresenta tendência de diminuição, porém menos pronunciada. Comparado as demais topologias, a Faster R-CNN apresenta um tamanho maior e latência de inferência superior, relacionados com um número maior de parâmetros. É uma característica dos modelos de dois estágios apresentar maior custo computacional em relação a arquiteturas de estágio único (como a YOLO) (JIAO *et al.*, 2019).

Esparsidade	mAP(0,5)	Latência média(GPU) [s]	Tamanho [Mb]
0	0,977	0,1831	390,09
0,1	0,977	0,1820	368,66
0,2	0,976	0,1819	343,31
0,3	0,976	0,1818	315,86
0,4	0,969	0,1816	286,85
0,5	0,929	0,1792	256,49
0,6	0,440	0,1749	224,90
0,7	0,040	0,1772	191,96

Tabela 17 – Resultados para a quantização Faster R-CNN.
Fonte - o Autor, 2023.

As Figuras 17, 18 e 19 demonstram a evolução destas métricas segundo o nível de esparsidade alvo.

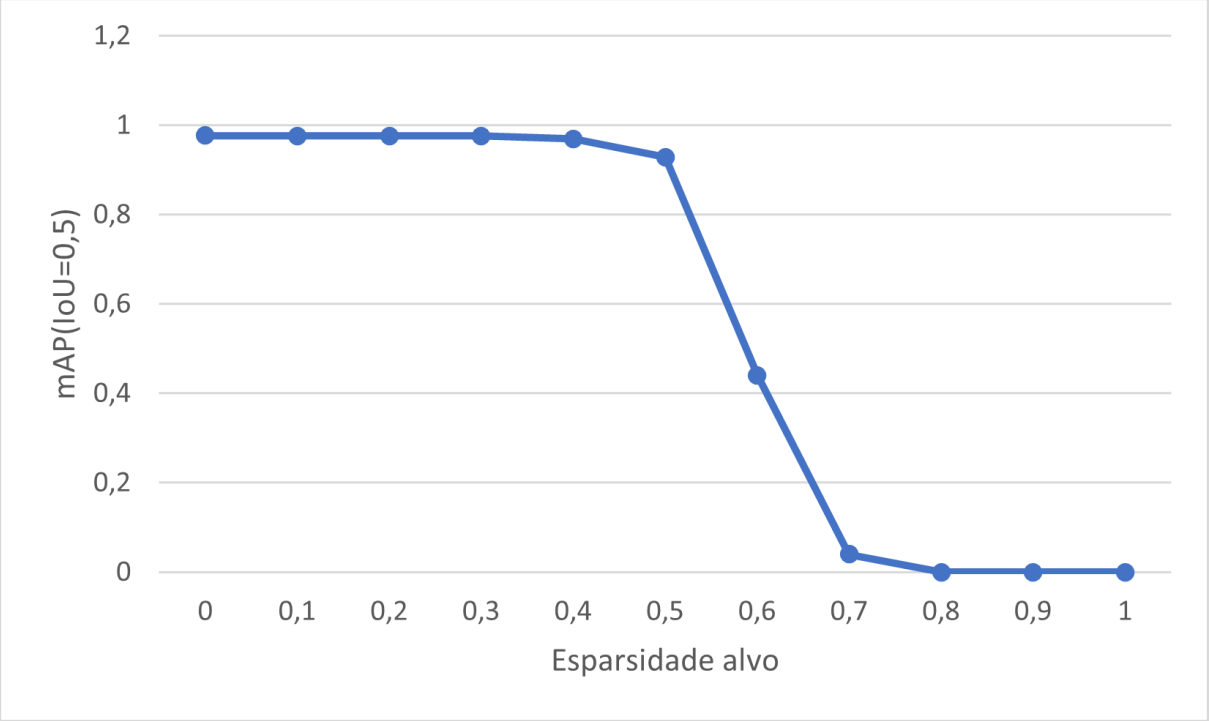

Figura 17 – Evolução do mAP(IoU=0,5) para o modelo Faster R-CNN em função da esparsidade e quantização ONNX ponto flutuante 32.
Fonte - o Autor, 2023.

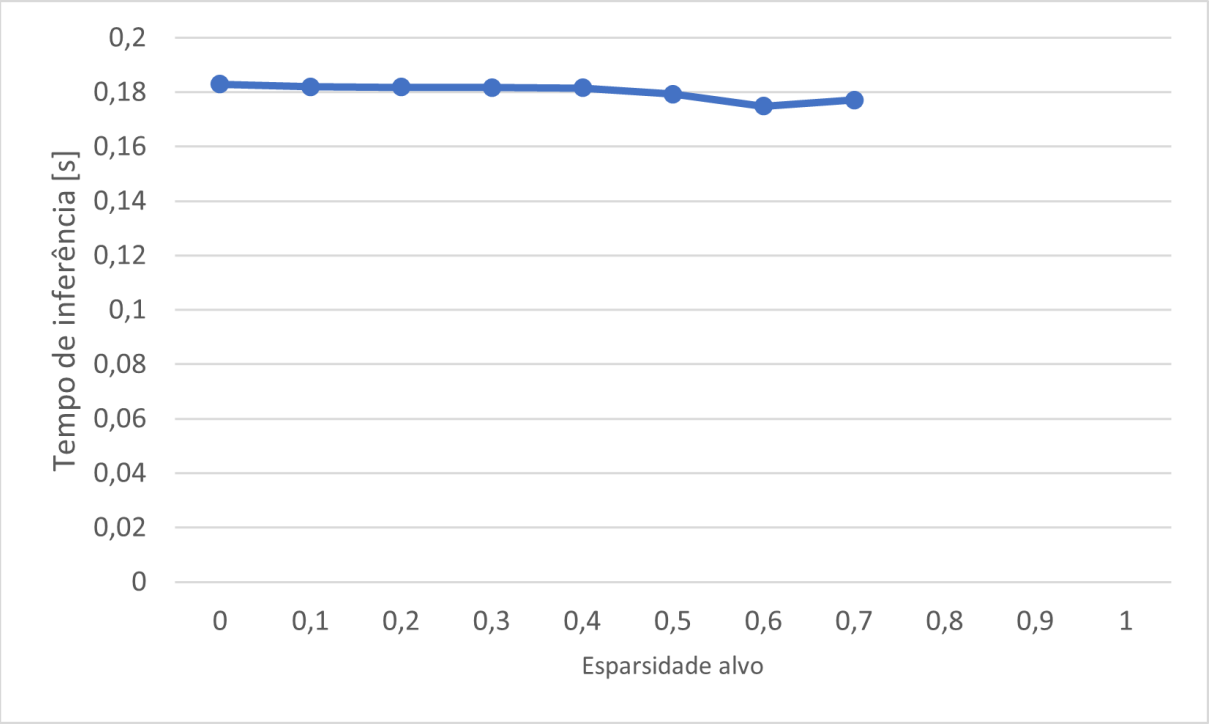

Figura 18 – Evolução do tempo de inferência para o modelo Faster R-CNN em função da esparsidade e quantização ONNX ponto flutuante 32.
Fonte - o Autor, 2023.

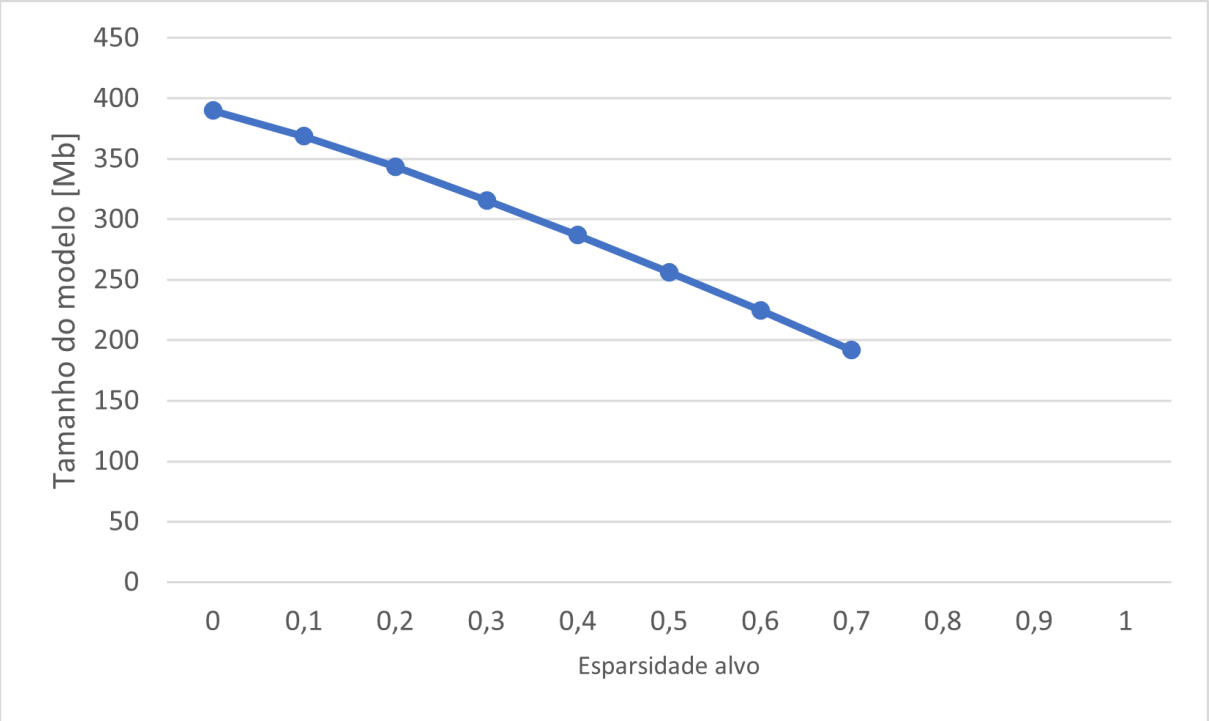

Figura 19 – Evolução do tamanho de modelo para Faster R-CNN em função da esparsidade e quantização ONNX ponto flutuante 32.
Fonte - o Autor, 2023.

4.5 Análise final dos resultados

4.5.1 Comparação com o Estado da Arte

4.5.1.1 Modelos originais

A Figura 20 demonstra a comparação entre os resultados (em função de mAP(0,5)) dos três modelos treinados, avaliados no conjunto de testes, em comparação aos trabalhos avaliados no Estado da Arte que utilizam a mesma base de dados (PCB Defect Dataset).

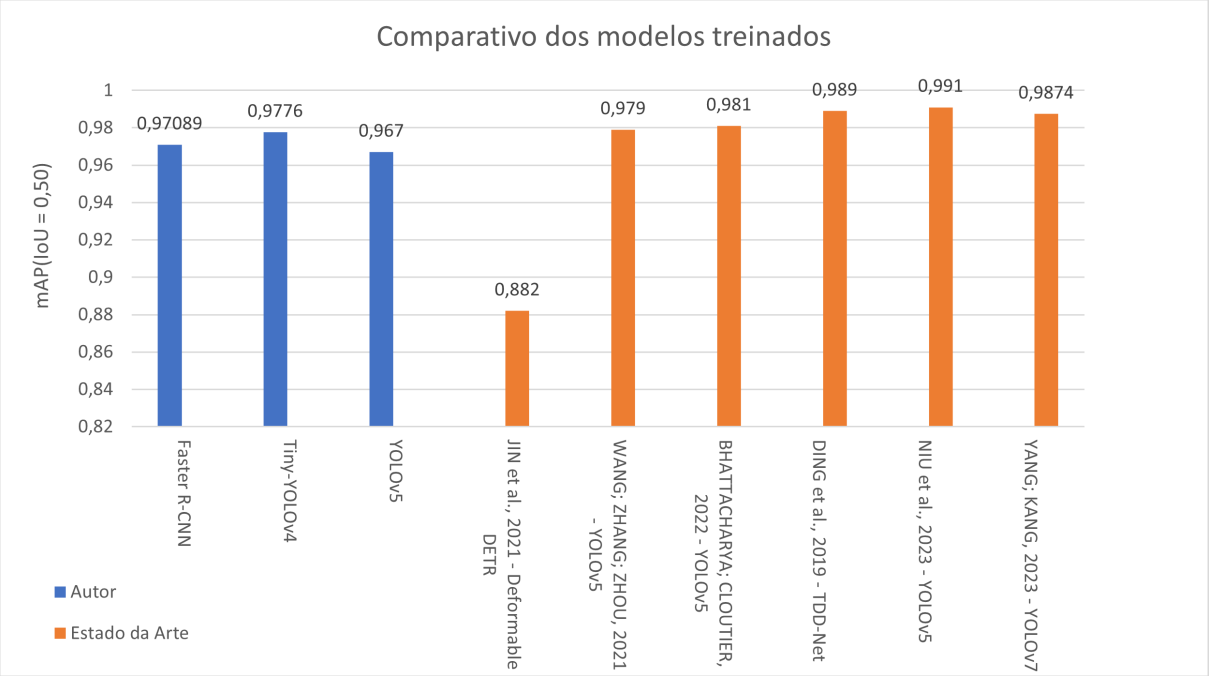

Figura 20 – Comparativo dos modelos treinados frente ao Estado da Arte.
Fonte - o Autor, 2023.

Observa-se que os resultados são comparáveis aos observados nos trabalhos tidos como Estado da Arte, com o desempenho, para as três topologias avaliadas, sendo inferior ao obtido pelos demais trabalhos (exceto (JIN *et al.*, 2021)), porém apresentando menos de 0,025 de diferença em relação ao trabalho com maior mAP (NIU *et al.*, 2023). Esse comportamento é esperado, pois os trabalhos apresentam topologias especialmente otimizadas para esta tarefa de detecção em específico. Apesar desta redução no mAP obtido pelas topologias genéricas (Tiny-YOLOv4, YOLOv5 e Faster R-CNN) face às topologias modificadas utilizadas nos demais trabalhos, a sua utilização apresenta como vantagem maior compatibilidade com as bibliotecas utilizadas nas etapas de *pruning* e quantização.

4.5.1.2 Modelos após *pruning* e quantização

A Figura 21 demonstra a comparação entre os resultados (em função de mAP(0,5)) dos modelos finais, avaliados no conjunto de testes, em comparação aos trabalhos avaliados

no Estado da Arte que utilizam a mesma base de dados (PCB Defect Dataset). Em função do baixo desempenho do modelo quantizado, o modelo original foi mantido para a topologia Tiny-YOLOv4. Para ambas as topologias YOLOv5 e Faster R-CNN, os modelos escolhidos para a comparação são de esparsidade 0,1, quantizados em formato ONNX 32 bits ponto flutuante. Esta escolha foi feita baseada no mAP(IoU=0,5) e tempo de inferência no conjunto de validação.

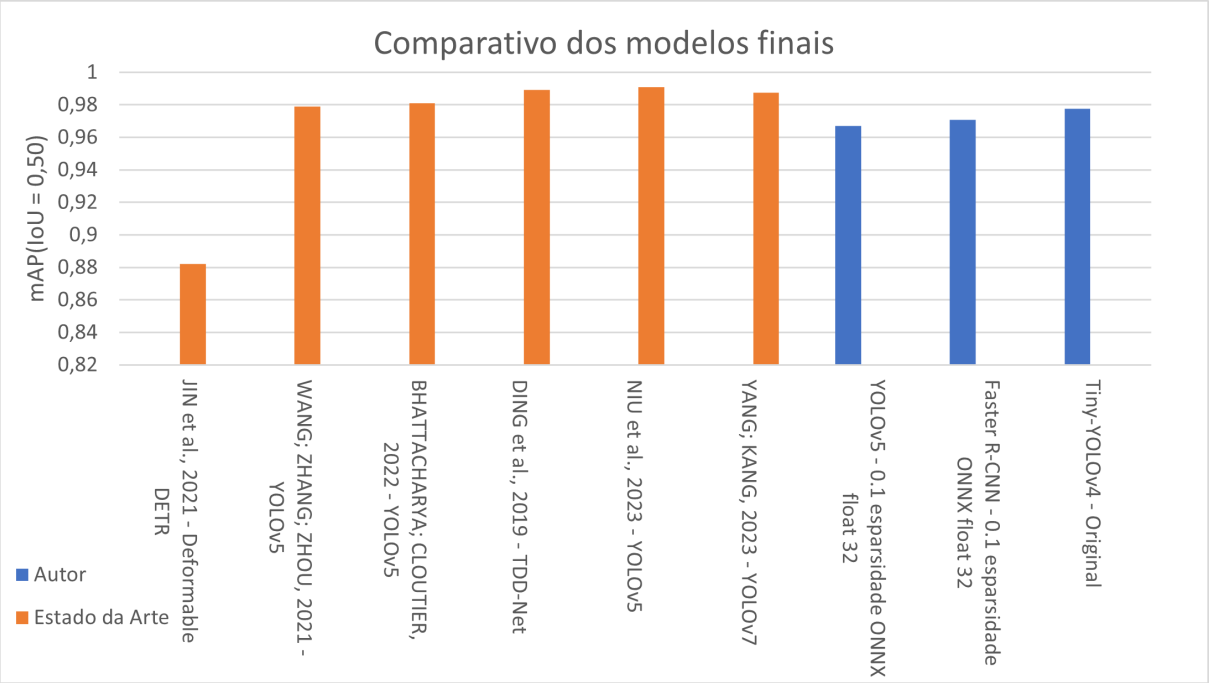

Figura 21 – Comparativo dos modelos quantizados frente ao Estado da Arte.
Fonte - o Autor, 2023.

Novamente, observa-se um pequeno decréscimo de mAP(IoU=0,5) frente a maioria das demais topologias.

4.5.2 Análise da base de dados de placas inteiras

Foi realizado o teste de detecção com as 67 PCBS inteiras (sem passar pelas etapas de seccionamento e *data augmentation*) que compõem o conjunto de teste da base de dados PCB Defect Dataset. A Tabela 18 sumariza os resultados obtidos com cada topologia (utilizando os modelos pré *pruning* e quantização). Percebe-se que, para as topologias Tiny-YOLOv4 e Faster R-CNN, ocorre redução significativa na *mAP* de teste ao utilizar a base de dados com as placas inteiras (em relação aos obtidos com as imagens processadas e seccionadas em formato 600 x 600 píxeis). A topologia YOLOv5 também apresenta redução da taxa de acerto, porém menos pronunciada. Uma possível fonte desta discrepância é o fato da implementação do YOLOv5 permitir o ajuste do formato de imagem esperado, ao passo que as demais topologias forçam a redução do tamanho da imagem.

Topologia	mAP(IoU=0,5) teste	Latência GPU teste [s]
Tiny-YOLOv4	0,775	0,1406
YOLOv5	0,940	0,0245
Faster R-CNN	0,803	0,2490

Tabela 18 – Resultados de testes com as placas inteiras da base PCB Defect Dataset.
Fonte - o Autor, 2023.

As Figuras 22 e 23 mostram, respectivamente, uma amostra do conjunto de teste com placas inteiras e os rótulos das instâncias de defeito, e o resultado da inferência. É considerado um *threshold* de confiança mínimo de 0,001.

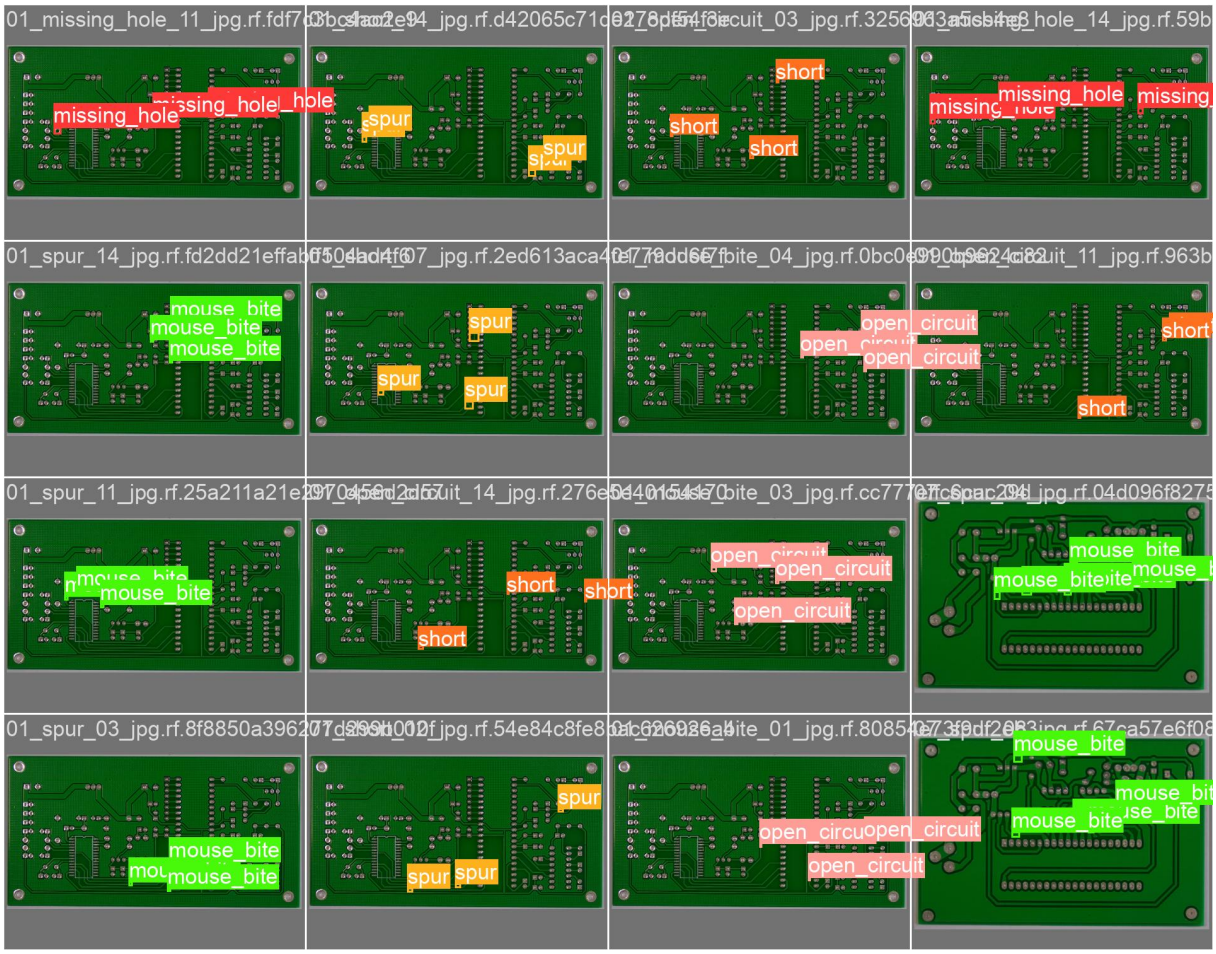

Figura 22 – Rótulos verdadeiros em placas inteiras da base PCB Dataset.
Fonte - o Autor, 2023.

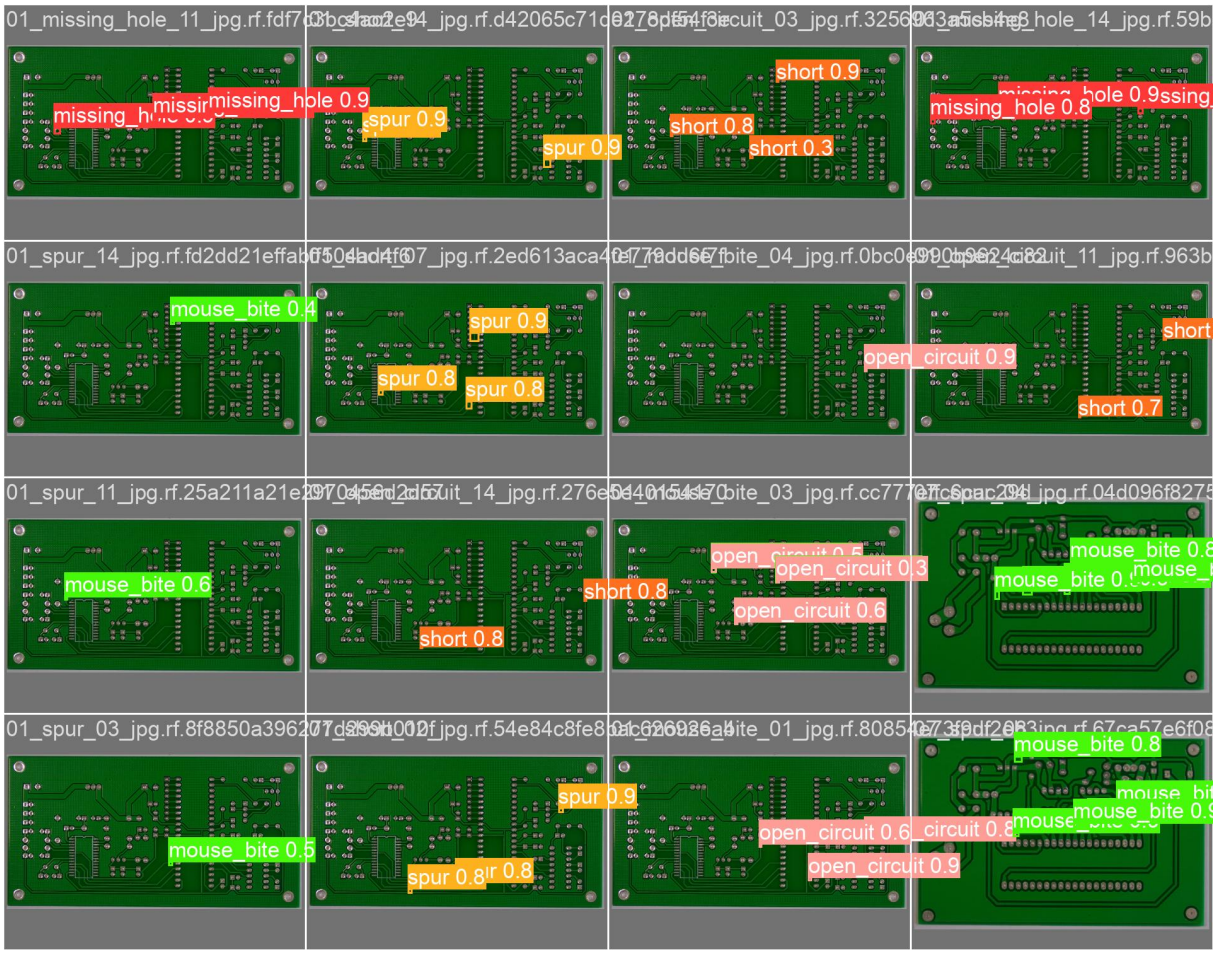

Figura 23 – Resultados da detecção em placas inteiras da base PCB Dataset.
Fonte - o Autor, 2023.

4.6 Estudo de caso - Implementação no Raspberry Pi 3B+

Observando os critérios de *mAP*, tempo de inferência, tamanho do modelo e compatibilidade, o modelo utilizado no estudo de caso é a topologia YOLOv5, com esparsidade de 0,1 e quantização ONNX 32 bits.

4.6.1 Desempenho de classificação

Os resultados finais, obtidos na plataforma Raspberry Pi com o conjunto de teste, são descritos na Tabela 19. Os resultados são apresentados segundo a etapa do experimento (modelo original, após o treinamento inicial, *pruning*, quantização, testes na plataforma). Há um leve aumento desta métrica, indicando uma possível redução do *overfitting* ao longo da simplificação do modelo. Observa-se que os indicadores de taxa de acerto no Raspberry Pi são equivalentes aos obtidos com este mesmo modelo quantizado na plataforma *Google Colab*.

Métrica	Modelo original	*Pruning*	Quantização	Raspberry Pi
mAP(0,5)	0,967	0,967	0,972	0,972
Precision	0,973	0,968	0,964	0,964
Recall	0,931	0,934	0,944	0,944
Latência CPU (por imagem) [s]	0,415	0,679	0,391	1,876
AP(0,5) [missing_hole]	0,988	0,988	0,989	0,989
AP(0,5) [mouse_bite]	0,956	0,948	0,952	0,952
AP(0,5) [open_circuit]	0,974	0,954	0,964	0,964
AP(0,5) [short_circuit]	0,971	0,975	0,975	0,975
AP(0,5) [spurious_copper]	0,968	0,965	0,975	0,975
AP(0,5) [spur]	0,944	0,974	0,980	0,980

Tabela 19 – Comparação dos resultados obtidos no conjunto de teste para cada etapa do experimento com o modelo YOLOv5, com esparsidade de 0,1 e quantização ONNX 32 bits.

Fonte - o Autor, 2023.

A Figura 24 corresponde a matriz de confusão para as classes de defeitos. É constatado que a taxa de verdadeiros positivos apresenta pouca variação entre as diferentes classes.

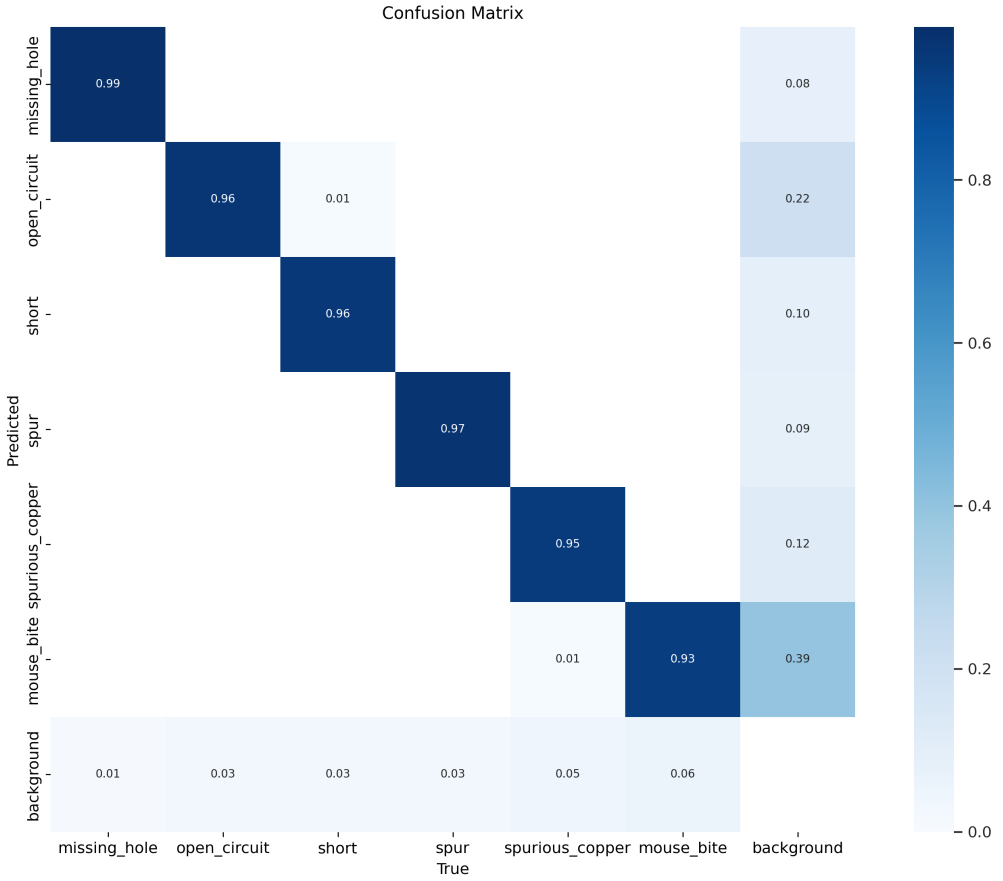

Figura 24 – Matriz de confusão para as previsões do modelo final, no conjunto de teste (valores normalizados por coluna).

Fonte - o Autor, 2023.

As Figuras 25 e 26 mostram, respectivamente, uma amostra do conjunto de teste com os rótulos das instâncias de defeito, e o resultado da inferência, utilizando um *threshold* de confiança mínimo de 0,001.

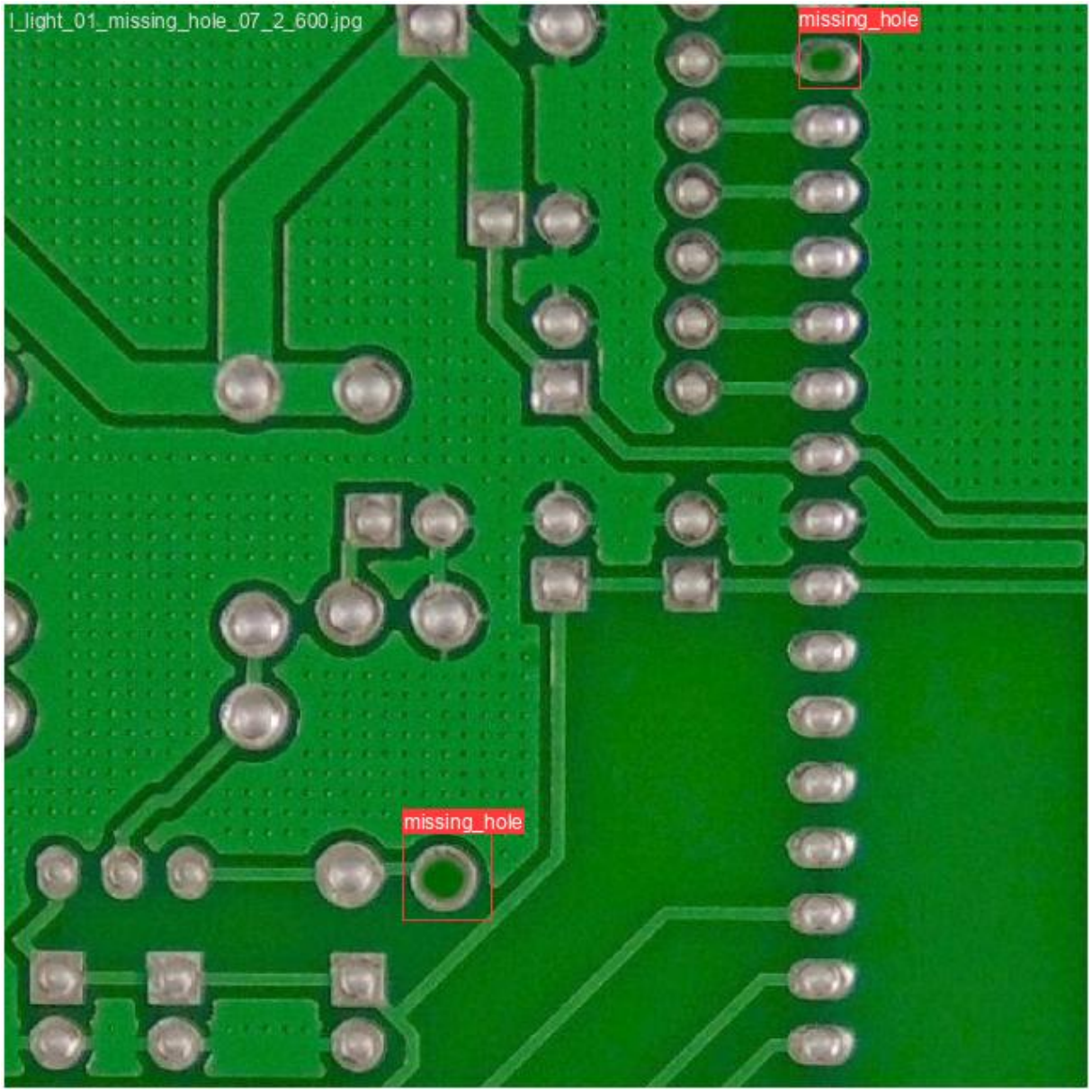

Figura 25 – Rótulos verdadeiros em imagem do conjunto de teste.
Fonte - o Autor, 2023.

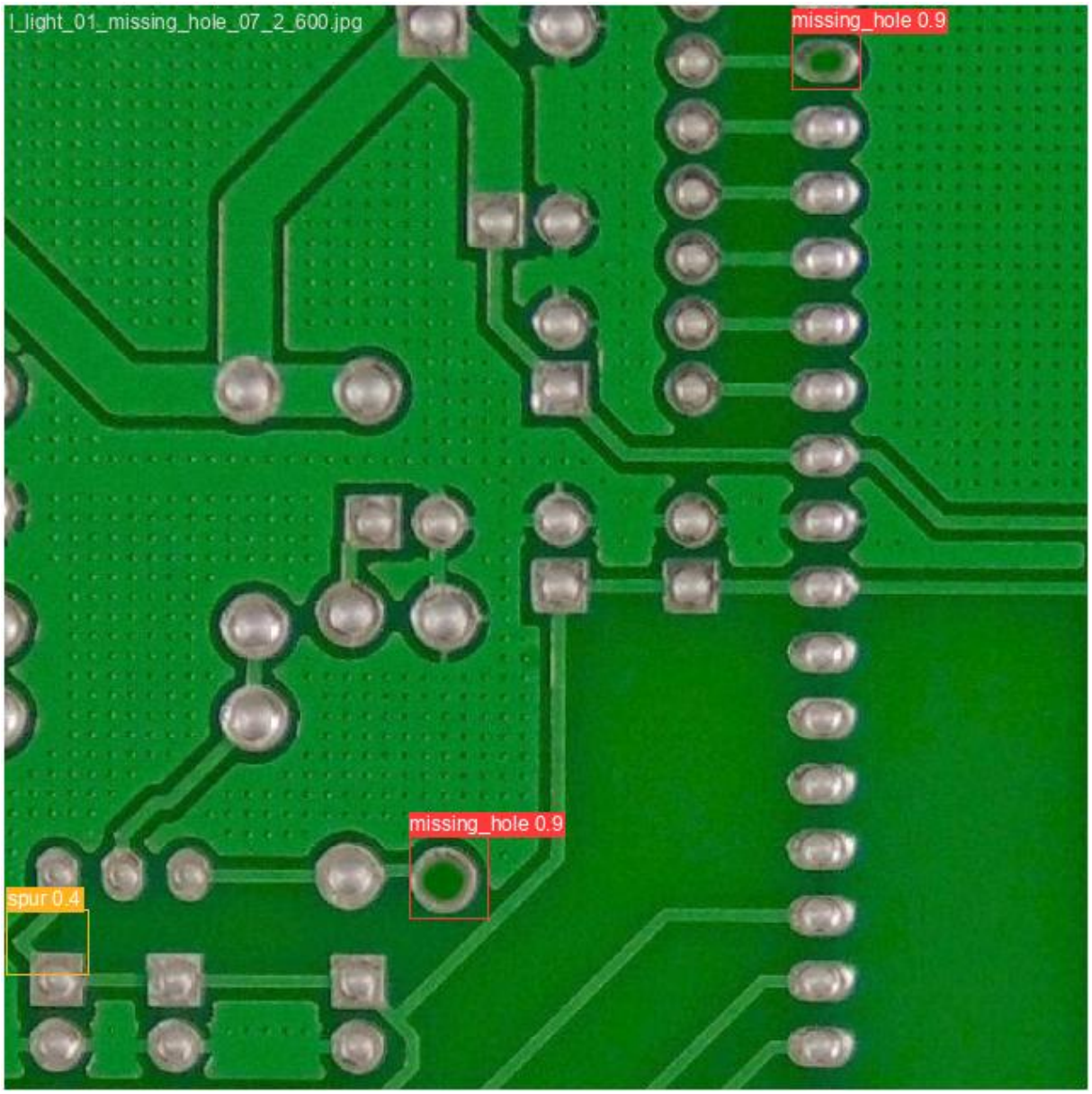

Figura 26 – Resultados da detecção em imagem do conjunto de teste.
Fonte - o Autor, 2023.

4.6.2 Análise do tempo de inferência

Observa-se ainda a partir da Tabela 19 que o tempo de inferência nas três primeiras etapas, realizadas na plataforma *Google Colab* com CPU, não sofre grande alteração. O tempo de inferência no Raspberry Pi, no entanto, é significativamente maior, ultrapassando 1,8 segundos. Embora este tempo de detecção ainda seja viável para detecções esporádicas, isto prejudicaria a implementação do sistema em uma linha de produção ou a realização de detecções em tempo real.

A Tabela 20 apresenta a decomposição do tempo total de inferência em termos de três etapas: o pré-processamento das imagens, a detecção propriamente dica e o pós-processamento (incluindo a etapa de NMS (*non max supression*), responsável por

desconsiderar *bounding boxes* sobrepostos). Somente a detecção é realizada no ambiente de execução ONNX, as outras etapas utilizam as bibliotecas PyTorch e Numpy. Observa-se que a etapa da detecção apresenta duração bastante superior às outras.

Modelo	Ambiente	Pré-processamento	Detecção	Pós-processamento	Total
Original	Colab	0,0035	0,4112	0,0007	0,4154
Quantizado	Colab	0,0072	0,3842	0,0005	0,3919
Original	Raspberry Pi	0,0093	4,8180	0,0043	4,8316
Quantizado	Raspberry Pi	0,0146	1,8549	0,0068	1,8763

Tabela 20 – Decomposição do tempo total de inferência (em segundos).
Fonte - o Autor, 2023.

No ambiente *Google Colab*, existe pouca diferença nos tempos de inferência entre os *frameworks* PyTorch e ONNX. No Raspberry Pi, porém, a diferença é pronunciada, pois o modelo quantizado ONNX alcança um tempo médio de inferência mais de 50% inferior.

4.6.3 Análise do uso de recursos computacionais

A Tabela 21 apresenta os índices de ocupação da CPU (%) e memória RAM (% e Gb), medidos para o modelo quantizado nas plataformas *Google Colab* e no Raspberry Pi. Observa-se que a taxa de utilização da CPU do Raspberry Pi ultrapassa 98%, indicando que este é o fator limitante na velocidade de detecção. A memória RAM, por sua vez, apresenta taxa de utilização inferior a 65%. No ambiente *Colab*, observa-se que as taxas de ocupação de ambos os recursos são inferiores. A memória RAM ocupada, contudo, ultrapassa a memória RAM disponível no Raspberry Pi, indicando que, mesmo sem as limitações em termos de CPU, este seria outro entrave.

Modelo	Ambiente	Uso CPU [%]	Uso RAM [%]	Uso RAM [Gb]
Original	Colab	0,716	0,115	1,220
Quantizado	Colab	0,793	0,108	1,136
Original	Raspberry Pi	0,751	0,687	0,588
Quantizado	Raspberry Pi	0,984	0,629	0,533

Tabela 21 – Decomposição do tempo total de inferência (em segundos).
Fonte - o Autor, 2023

5 Conclusões

O trabalho desenvolvido validou a utilização das topologias de modelo de detecção YOLO e Faster R-CNN na detecção de defeitos de fabricação em PCB, utilizando a base de dados PCB Dataset. A acurácia obtida nos experimentos, superior a 0,96, é comparável ao Estado da Arte na mesma base de dados, e a análise das precisões por classe demonstra que os modelos são capazes de detectar todos os defeitos estudados. A baixa diversidade das amostras nesta base de dados, contudo, limita a possibilidade de extrair conclusões a respeito da generalização do modelo para um número maior de PCBs.

Foi também comprovada a efetividade das técnicas de *pruning* não-estruturado e quantização na redução do tamanho dos modelos e da carga computacional exigida na inferência. O impacto destas técnicas na latência de inferência é limitado, e para atingir melhores resultados neste aspecto, são necessárias simplificações mais profundas na arquitetura do modelo, combinadas a otimizações adequadas ao *hardware* alvo.

O estudo de caso demonstra a possibilidade de implementação de um modelo de topologia YOLOv5, com aplicação de *pruning* e quantizado no formato ONNX 32 bits, em um computador de placa única Raspberry Pi 3b+. O modelo embarcado possui desempenho (mensurado pela métrica mAP(IoU=0,5)) idêntico à obtida utilizando a plataforma *Google Colab*. A latência obtida no Raspberry Pi, superior a 1,8 segundos, torna a detecção esporádica de defeitos viável, porém dificulta o processo de detecção em série de PCBs. Este tempo de processamento elevado deve-se às limitações na plataforma Raspberry Pi. A quantização e conversão ao formato ONNX, porém, mostra-se eficaz ao reduzir o tempo de inferência em mais de 50% em relação ao modelo PyTorch.

5.1 Trabalhos futuros

O desenvolvimento de uma nova base de dados física de PCBs defeituosas representaria uma contribuição importante, permitindo ora uma validação dos modelos já treinados em um caso de testes mais próximo da realidade, ora incrementar a variedade e tamanho do conjunto de treinamento. Esta base de dados deverá conter um número significativo de PCBs, com os defeitos de fabricação estudados introduzidos de forma física nas placas.

Outro estudo passível de ser desenvolvido é uma análise mais aprofundada de técnicas de *pruning* estruturado, visando obter redução na latência do modelo. Este estudo abrangeria o teste de diferentes técnicas de *pruning* estruturados concentradas em uma única topologia (possivelmente a YOLOv5, devido aos bons resultados obtidos com ela).

Uma terceira frente de desenvolvimento é a implementação do modelo em uma plataforma móvel com maior poder de processamento e memória RAM, como o Raspberry Pi 4 ou mesmo um *smartphone*. Isto abriria a possibilidade de implementação de um sistema de AOI, utilizando o modelo na detecção de defeitos em PCBs, seja na fabricação artesanal ou em uma linha de produção de pequena escala.

Referências Bibliográficas

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANé, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIéGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

BHATTACHARYA, A.; CLOUTIER, S. G. End-to-end deep learning framework for printed circuit board manufacturing defect classification. *Scientific Reports*, v. 12, n. 1, p. 12559, jul. 2022. ISSN 2045-2322. Disponível em: <https://doi.org/10.1038/s41598-022-16302-3>.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv, 2020. ArXiv:2004.10934 [cs, eess]. Disponível em: <http://arxiv.org/abs/2004.10934>.

CHAUDHARY, V.; DAVE, I. R.; UPLA, K. P. *Automatic visual inspection of printed circuit board for defect detection and classification*. 2017. 732–737 p.

CHIAO, A. *Pruning in Keras example | TensorFlow Model Optimization*. 2023. Disponível em: <https://www.tensorflow.org/model_optimization/guide/pruning/pruning_with_keras>.

CHOLLET, F. *Deep learning with Python*. Shelter Island, New York: Manning Publications Co, 2018. OCLC: ocn982650571. ISBN 978-1-61729-443-3.

DEVELOPERS, O. R. *ONNX Runtime*. 2021. <https://onnxruntime.ai/>. Version: 1.14.1.

DING, R.; DAI, L.; LI, G.; LIU, H. TDD-Net: A Tiny Defect Detection Network for Printed Circuit Boards. *CAAI Transactions on Intelligence Technology*, v. 4, abr. 2019.

EBAYYEH, A. A. R. M. A.; MOUSAVI, A. A Review and Analysis of Automatic Optical Inspection and Quality Monitoring Methods in Electronics Industry. *IEEE Access*, v. 8, p. 183192–183271, 2020. ISSN 2169-3536. Conference Name: IEEE Access.

GERON, A. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. [S.l.]: O'Reilly Media, Incorporated, 2019. Google-Books-ID: OCS1twEACAAJ. ISBN 978-1-4920-3264-9.

GHOLAMI, A.; KIM, S.; DONG, Z.; YAO, Z.; MAHONEY, M. W.; KEUTZER, K. *A Survey of Quantization Methods for Efficient Neural Network Inference*. arXiv, 2021. ArXiv:2103.13630 [cs]. Disponível em: <http://arxiv.org/abs/2103.13630>.

GHOSH, B.; BHUYAN, M. K.; SASMAL, P.; IWAHORI, Y.; GADDE, P. *Defect Classification of Printed Circuit Boards based on Transfer Learning*. 2018. 245–248 p.

HU, B.; WANG, J. Detection of PCB Surface Defects With Improved Faster-RCNN and Feature Pyramid Network. *IEEE Access*, v. 8, p. 108335–108345, 2020. ISSN 2169-3536. Conference Name: IEEE Access.

IBRAHIM, Z.; AL-ATTAS, S. Wavelet-based printed circuit board inspection algorithm. *Integrated Computer-Aided Engineering*, v. 12, p. 201–213, abr. 2005.

JIAO, L.; ZHANG, F.; LIU, F.; YANG, S.; LI, L.; FENG, Z.; QU, R. A Survey of Deep Learning-Based Object Detection. *IEEE Access*, v. 7, p. 128837–128868, 2019. ISSN 2169-3536. Conference Name: IEEE Access.

JIN, J.; FENG, W.; LEI, Q.; GUI, G.; WANG, W. *PCB defect inspection via Deformable DETR*. 2021. 646–651 p.

KHANDPUR, R. S. *Printed circuit boards: design, fabrication, assembly and testing*. New York: McGraw-Hill, 2006. OCLC: ocm62032512. ISBN 978-0-07-146420-8.

LI, K.; LI, X. *Model Compression, the Pruning Approaches*. Disponível em: <https://www.cs.princeton.edu/courses/archive/spring21/cos598D/lectures/pruning.pdf>.

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; BOURDEV, L.; GIRSHICK, R.; HAYS, J.; PERONA, P.; RAMANAN, D.; ZITNICK, C. L.; DOLLáR, P. *Microsoft COCO: Common Objects in Context*. arXiv, 2015. ArXiv:1405.0312 [cs]. Disponível em: <http://arxiv.org/abs/1405.0312>.

LIU, Z.; QU, B. Machine vision based online detection of PCB defect. *Microprocessors and Microsystems*, v. 82, p. 103807, abr. 2021. ISSN 0141-9331. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0141933120309522>.

MICHELA, P. *Pruning Tutorial — PyTorch Tutorials 2.0.1+cu117 documentation*. Disponível em: <https://pytorch.org/tutorials/intermediate/pruning_tutorial.html#global-pruning>.

MUJEEB, A.; DAI, W.; ERDT, M.; SOURIN, A. One class based feature learning approach for defect detection using deep autoencoders. *Advanced Engineering Informatics*, v. 42, p. 100933, 2019. ISSN 1474-0346. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1474034619301259>.

NIU, J.; LI, H.; CHEN, X.; QIAN, K. An Improved YOLOv5 Network for Detection of Printed Circuit Board Defects. *Journal of Sensors*, v. 2023, p. e7270093, mar. 2023. ISSN 1687-725X. Publisher: Hindawi. Disponível em: <https://www.hindawi.com/journals/js/2023/7270093/>.

PABLO, E. Neural Networks and Deep Learning, Charu C. Aggarwal. *Neural Networks and Deep Learning A Textbook*, jan. 2018. Disponível em: <https://www.academia.edu/42981452/Neural_Networks_and_Deep_Learning_Charu_C_Aggarwal>.

PADILLA, R.; NETTO, S. L.; SILVA, E. A. B. da. A Survey on Performance Metrics for Object-Detection Algorithms. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. [S.l.: s.n.], 2020. p. 237–242. ISSN: 2157-8702.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. arXiv, 2016. ArXiv:1506.01497 [cs]. Disponível em: <http://arxiv.org/abs/1506.01497>.

RICHTER, J.; STREITFERDT, D.; ROZOVA, E. *On the development of intelligent optical inspections*. 2017. 1–6 p.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, v. 65, n. 6, p. 386–408, 1958. ISSN 1939-1471. Place: US Publisher: American Psychological Association.

SILVA, L. H. d. S.; AZEVEDO, G. O. d. A.; FERNANDES, B. J. T.; BEZERRA, B. L. D.; LIMA, E. B.; OLIVEIRA, S. C. *Automatic Optical Inspection for Defective PCB Detection Using Transfer Learning*. 2019. 1–6 p.

TANG, S.; HE, F.; HUANG, X.; YANG, J. *Online PCB Defect Detector On A New PCB Defect Dataset*. arXiv, 2019. ArXiv:1902.06197 [cs]. Disponível em: <http://arxiv.org/abs/1902.06197>.

ULTRALYTICS. *YOLOv5 (6.0/6.1) brief summary · Issue #6998 · ultralytics/yolov5*. 2022. Disponível em: <https://github.com/ultralytics/yolov5/issues/6998>.

VOLKAU, I.; MUJEEB, A.; WENTING, D.; MARIUS, E.; ALEXEI, S. *Detection Defect in Printed Circuit Boards using Unsupervised Feature Extraction Upon Transfer Learning*. 2019. 101–108 p. ISSN: 2642-3596.

WANG, M. *Como os melhores fabricantes de PCB de núcleo de metal desenvolvem o Mcpcbs*. 2021. Disponível em: <https://www.pcbmay.com/pt/fabricantes-de-pcb-com-n%C3%BAcleo-de-metal-desenvolvem-mcpcbs/>.

WANG, X.; ZHANG, X.; ZHOU, N. *Improved YOLOv5 with BiFPN on PCB Defect Detection*. 2021. 196–199 p.

WU, Y.; KIRILLOV, A.; MASSA, F.; LO, W.-Y.; GIRSHICK, R. *Detectron2*. 2019. <https://github.com/facebookresearch/detectron2>.

YANG, Y.; KANG, H. An Enhanced Detection Method of PCB Defect Based on Improved YOLOv7. *Electronics*, v. 12, n. 9, p. 2120, jan. 2023. ISSN 2079-9292. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute. Disponível em: <https://www.mdpi.com/2079-9292/12/9/2120>.

YOHANANDAN, S. *mAP (mean Average Precision) might confuse you!* 2020. Disponível em: <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2>.

YOU, S. *PCB Defect Detection based on Generative Adversarial Network*. 2022. 557–560 p. 2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE).

ZHANG, H.; JIANG, L.; LI, C. CS-ResNet: Cost-sensitive residual convolutional neural network for PCB cosmetic defect detection. *Expert Systems with Applications*, v. 185, p. 115673, 2021. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417421010617>.