UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

JOÃO PEDRO OLIVEIRA SANTIAGO

DETECÇÃO EM TEMPO REAL DE DANOS CAUSADOS PELOS CONSUMIDORES EM PLACAS DE CIRCUITO IMPRESSO USANDO DISPOSITIVOS MÓVEIS E DETECTORES YOLO

QUIXADÁ

2023

JOÃO PEDRO OLIVEIRA SANTIAGO

DETECÇÃO EM TEMPO REAL DE DANOS CAUSADOS PELOS CONSUMIDORES EM PLACAS DE CIRCUITO IMPRESSO USANDO DISPOSITIVOS MÓVEIS E DETECTORES YOLO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Regis Pires Magalhães.
Coorientador: Prof. Dr. Victor Aguiar Evangelista de Farias.

QUIXADÁ

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S226d Santiago, João Pedro Oliveira.
Detecção em tempo real de danos causados pelos consumidores em placas de circuito impresso usando dispositivos móveis e detectores YOLO / João Pedro Oliveira Santiago. – 2023.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Computação, Quixadá, 2023.
Orientação: Prof. Dr. Regis Pires Magalhães.
Coorientação: Prof. Dr. Victor Aguiar Evangilista de Farias.

1. Visão Computacional. 2. Aprendizagem profunda. 3. Placa de circuito impresso. 4. Dispositivos móveis. I. Título.

CDD 621.39

JOÃO PEDRO OLIVEIRA SANTIAGO

DETECÇÃO EM TEMPO REAL DE DANOS CAUSADOS PELOS CONSUMIDORES EM PLACAS DE CIRCUITO IMPRESSO USANDO DISPOSITIVOS MÓVEIS E DETECTORES YOLO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. Regis Pires Magalhães (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Victor Aguiar Evangelista de Farias (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Iago Castro Chaves
Universidade Federal do Ceará (UFC)

Prof. Me. Carlos Igor Ramos Bandeira
Universidade Federal do Ceará (UFC)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada.

AGRADECIMENTOS

A minha família que sempre me apoiou e estiveram comigo durante a vida.

Ao Prof. Dr. Victor Farias, pela orientação e por me apresentar ao LSBD onde tive a oportunidade de fazer parte da equipe de pesquisa em IA e venho aprendendo bastante. Também ao professor Dr. Régis Pires Magalhães que aceitou também me orientar quando o Victor teve que deixar o Campus.

Aos professores participantes da banca examinadora MSc. Iago Chaves e MSc. Carlos Igor Ramos pelo tempo, pelas valiosas colaborações e sugestões.

Aos meus amigos e colegas pelas críticas e sugestões durante o desenvolvimento do projeto.

"Só se pode alcançar um grande êxito quando nos mantemos fiéis a nós mesmos." (Friedrich Nietzsche)

RESUMO

A identificação de danos causados pelo consumidor é essencial para os programas de garantia de fabricantes de eletrônicos. Danos Induzidos pelo Consumidor (DIC) referem-se a quaisquer danos causados por uma pessoa não autorizada, incluindo o próprio consumidor. A garantia do produto não cobre esses danos, evitando despesas no faturamento do fabricante. O processo de garantia para danos causados pelo consumidor geralmente é realizado manualmente por pessoas tecnicamente qualificadas. No entanto, essa tarefa exige muita atenção aos detalhes, pode ser demorada e está suscetível a erros humanos. Com isso em mente, este trabalho apresenta um modelo de detecção de objetos para dispositivos de baixo custo computacional, utilizando métodos de visão computacional e aprendizado profundo com detectores YOLO embarcados em dispositivos móveis para identificar danos causados pelo consumidor em placas de circuito impresso. Foram realizados dezesseis experimentos com quatro arquiteturas de redes neurais YOLO e foi desenvolvido com sucesso um aplicativo móvel para detecção de DIC. O melhor modelo alcançou uma mAP@0.5 de 33,1% e uma média de 5,7 FPS em dispositivos móveis reais.

Palavras-chave: visão computacional; aprendizagem profunda; placa de circuito impresso; dispositivos móveis.

ABSTRACT

The identification of consumer-induced damage is essential for electronics manufacturers' warranty programs. Consumer-Induced Damage (CID) refers to any damage caused by an unauthorized person, including the consumer. The product warranty does not cover these damages, avoiding expenses in the manufacturer's revenue. The warranty process for consumer-induced damage is usually carried out manually by technically qualified individuals. However, this task demands a lot of attention to detail, can be time-consuming, and is susceptible to human errors. With this in mind, this work presents an object detection model for low-computational-cost devices using computer vision and deep learning methods with YOLO detectors embedded in mobile devices to identify consumer-induced damages on printed circuit boards (PCB). Sixteen experiments were conducted with four YOLO neural network architectures, and a mobile application for CID detection was successfully developed. The best model achieved a mAP@0.5 of 33.1% and an average of 5.7 FPS on real mobile devices.

Keywords: computer vision; deep learning; printed circuit board; mobile devices.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

AP	*Average Precision*
AR	*Average Recall*
DIC	Danos Induzidos pelo Consumidor
GT	*Ground Thuths*
IA	Inteligência Artificial
IoU	*Intersection over Union*
mAP	*Mean Average Precision*
MnasNet	*Mobile Neural Architecture Search*
NAS	*Neural Architecture Search*
NASNet	*Neural Architecture Search Network*
PCB	*Printed Circuit Board*
RAG	Redes Adversárias Generativas
RNA	Redes Neurais Artificiais
RNC	Redes Neurais Convolucionais
YOLO	*You Only Look Once*

SUMÁRIO

1 INTRODUÇÃO

Atualmente as fabricantes de dispositivos eletrônicos têm de arcar com grande custo para cobrir a garantia de produtos que foram adquiridos por consumidores. Em muitos casos, o próprio consumidor induz o dispositivo a um ou mais defeitos por mal uso ou manutenção por uma pessoa não especializada (ALVES *et al.*, 2022). Quando o dano é introduzido pelo consumidor, as fabricantes não cobrem os custos de reparo e são conhecidos como DIC.

Uma Placa de Circuito Impresso, em inglês *Printed Circuit Board* (PCB) é fundamental na estrutura interna de muitos dispositivos eletrônicos. A PCB consiste em uma placa constituída de camadas condutoras e isolantes que faz a integração de vários componentes eletrônicos e elétricos (ALVES *et al.*, 2022). Devido a grande importância da PCB na composição dos dispositivos eletrônicos, a avaliação desta estrutura no processo de garantia e controle de qualidade de produtos é de extrema importância. Entretanto, o processo de diagnóstico de PCB majoritariamente é feito de maneira manual, através de técnicos especializados (ADIBHATLA *et al.*, 2020). Esta tarefa é demorada e suscetível a erros, por conta de defeitos que podem ser difíceis de identificar. Devido a estes fatos, a inspeção de PCB de maneira automática é considerada e alguns pesquisadores voltaram seus esforços para realizar a detecção de defeitos em PCB de maneira automática (HE *et al.*, 2020). Recentemente estudos (ALVES *et al.*, 2022) mostram que é possível construir modelos de detecção de defeitos em PCB por meio de técnicas de visão computacional.

A visão computacional vem se tornando cada vez mais evidente em tarefas do dia-a-dia como na classificação de imagens, detecção de objetos, segmentação de objetos e reconhecimento de faces. O principal método que permite que estes tipos de tarefas sejam realizados ocorre através do aprendizado profundo por meio do uso de Redes Neurais Convolucionais (RNC), que fornece bons resultados nestas atividades (GÉRON, 2022). Buscando sempre melhores acurácias e precisão, os modelos de aprendizado profundo geralmente são robustos e possuem alto custo computacional, não sendo eficientes para serem utilizados em dispositivos móveis. No entanto, quando uma PCB chega ao centro de reparo, o técnico responsável pela inspeção da peça tem que verificar quais os defeitos estão ocorrendo e isto deve ser realizado rapidamente por um dispositivo móvel. Um modelo de detecção de defeitos em PCB pode ser utilizado em um servidor na nuvem e acessado através de uma API em um dispositivo móvel, porém, isto remete em mais infraestrutura, latência e a maiores custos financeiros, além de ser necessário possuir conexão de internet disponível para realizar o acesso ao servidor externo,

impactando negativamente na privacidade. Neste contexto, o modelo de detecção de defeitos em PCB deve ser executado diretamente em dispositivos móveis. Dispositivos móveis possuem baixo poder computacional, à vista disso, existem trabalhos que tem o objetivo de desenvolver modelos de aprendizado profundo para a detecção de objetos com maior desempenho e fluidez em dispositivos móveis, a exemplo de *ShuffleNet* (ZHANG *et al.*, 2018), *EfficientDet* (TAN *et al.*, 2020) e *MobileNet* (HOWARD *et al.*, 2017).

O *You Only Look Once* (YOLO) revolucionou o campo da detecção de objetos em visão computacional. Devido à sua alta velocidade e boa precisão, tornou-se uma escolha muito popular para várias aplicações em tempo real. As abordagens do YOLO diferem dos métodos tradicionais ao prever diretamente caixas delimitadoras e probabilidades de classe em uma única etapa, em oposição ao processo de duas etapas de propostas de região e classificação usado em técnicas mais antigas (REDMON *et al.*, 2016). Essa abordagem reduz significativamente a sobrecarga computacional, tornando o YOLO ideal para ambientes com recursos limitados.

Atualmente, existe uma ampla variedade de modelos YOLO baseados na arquitetura original (REDMON *et al.*, 2016; REDMON; FARHADI, 2017; REDMON; FARHADI, 2018; BOCHKOVSKIY *et al.*, 2020; LI *et al.*, 2023; WANG *et al.*, 2023). Cada um desses trabalhos se esforça para melhorar a precisão e o desempenho.

1.1 Objetivos

Este trabalho tem o objetivo geral de desenvolver um modelo de baixo custo computacional para dispositivos móveis que utiliza o Aprendizado Profundo em conjunto com técnicas de Visão Computacional e detectores de objetos YOLO para detectar defeitos causados por consumidores em PCB de maneira automática, sendo uma ferramenta auxiliar para técnicos de centros de reparos de fabricantes eletrônicos. O modelo gerado deve possuir baixo poder de processamento para responder rápido mesmo em dispositivos de hardware limitado. Para isto, um modelo de detecção de objetos será treinado para detectar vários tipos de defeitos em PCB. Possuindo baixa complexidade computacional, este modelo será convertido para tornar-se possível a utilização em dispositivos móveis, através de imagens capturadas pela câmera destes dispositivos.

De forma a alcançar o objetivo geral deste trabalho, os objetivos específicos são listados a seguir.

– Rotular uma base de dados para o treinamento e validação do modelo;

- Realizar o treinamento do modelo;
- Realizar a validação e melhoria do modelo;
- Realizar uma prova de conceito com o modelo treinado em dispositivos móveis.

O trabalho está organizado da seguinte forma. No Capítulo 2, são apresentados todos os fundamentos teóricos importantes para a compreensão do leitor. No Capítulo 3, são apresentados trabalhos relacionados a este tema de pesquisa, e que fornecem suporte para este trabalho. No Capítulo 4 são apresentados os passos realizados para a execução deste trabalho. O Capítulo 5 apresenta os experimentos e resultados obtidos. Por fim, no Capítulo 6 são apresentadas as conclusões obtidas neste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo foram apresentados os principais conceitos para o entendimento da solução proposta neste trabalho.

A Seção 2.1 apresentou o conceito de DIC. Na Seção 2.2, é apresentado o conceito estado da arte de Visão Computacional. Na Seção 2.3, são apresentados os conceitos de Inteligência Artificial, Aprendizado de Máquina e Aprendizado Profundo. Na Seção 2.4 são apresentados os conceitos de Redes Neurais Convolucionais, bem como os principais métodos de Detecção de Objetos para dispositivos de baixo custo computacional.

2.1 Danos Induzidos pelo Consumidor

Danos a componentes eletrônicos, como placas-mãe, podem resultar em mau funcionamento ou falhas nos dispositivos associados. As origens destes danos podem ser variadas, e um corpo considerável de literatura concentra-se predominantemente em problemas decorrentes do processo de fabricação (ALVES *et al.*, 2022). Notavelmente, as empresas de componentes elétricos empregam procedimentos rigorosos de controle de qualidade para mitigar os danos induzidos pela fabricação.

No entanto, outra fonte predominante de danos é o DIC. DIC abrange danos causados por indivíduos não autorizados, incluindo clientes (ALVES *et al.*, 2022). Tais danos podem ter implicações financeiras significativas para as empresas, especialmente em termos de custos de garantia. É imperativo identificar instâncias de DIC, visto que o padrão predominante da indústria normalmente exclui o DIC da cobertura da garantia (ALVES *et al.*, 2022). Mecanismos eficazes de detecção de DIC podem servir como um meio para as empresas otimizarem a economia de custos.

2.2 Visão Computacional

Os humanos têm facilidade em perceber estruturas e reconhecer objetos, a Visão Computacional utiliza técnicas para descrever o mundo em uma ou mais imagens e reconstruir suas propriedades, como forma e distribuição de cores (SZELISKI, 2022).

Visão computacional pode ser aplicada em diversas tarefas, em tarefas de classificação de imagens, é possível analisar imagens de produtos em uma linha de produção para automaticamente classificar estes produtos (GÉRON, 2022). Ainda de acordo com (GÉRON,

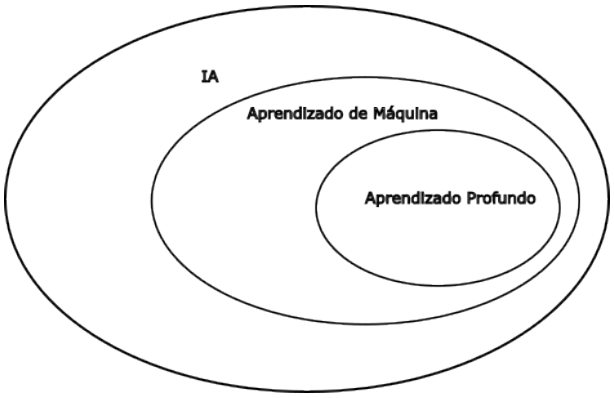
2022), a detecção de objetos tem o objetivo de localizar onde um objeto de interesse está na imagem, permitindo que seja factível detectar tumores em exames cerebrais.

Segundo Gonzalez e Woods (2009) o processamento digital de imagens e a visão computacional estão fortemente relacionados, e fica evidente ao dividir o processamento digital em três paradigmas (nível baixo, médio e alto). No nível baixo, está o pré-processamento de imagens para redução de ruídos, contraste, realce e aguçamento de imagens. No nível médio encontra-se a segmentação de objetos (separação da imagem em regiões ou objetos) e a classificação (reconhecimento) de objetos individuais. Por fim, o nível alto onde estão a análise de imagens e funções cognitivas relacionadas a visão.

2.3 Inteligência Artificial, Aprendizado de Máquina e Aprendizado Profundo

Em meados de 1950 inicia-se o questionamento sobre a possibilidade de máquinas pensarem, surgindo o que é conhecido como Inteligência Artificial. A Inteligência Artificial (IA) preocupa-se em entender e construir entidades inteligentes, máquinas que podem calcular como agir de forma eficaz e segura em uma ampla variedade de novas situações (RUSSELL, 2010). A IA é uma grande área de conhecimento que tem como subáreas o Aprendizado de Máquina e o Aprendizado Profundo. A Figura 1 mostra a representação de IA e seus subcampos.

Figura 1 – Inteligência Artificial e suas subáreas.



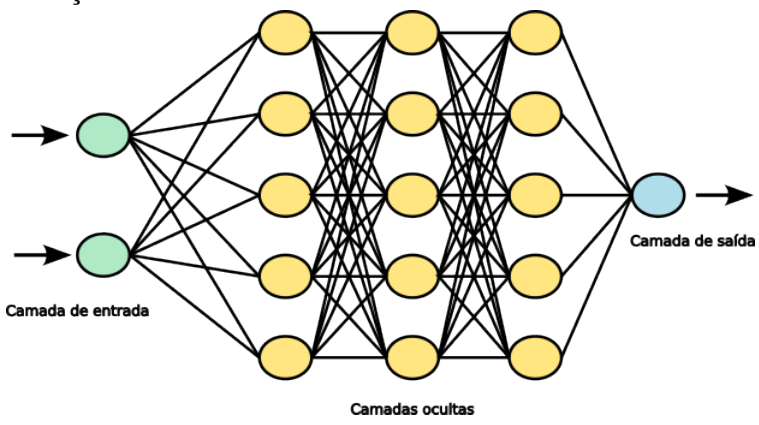
Fonte: adaptada de Chollet (2021)

As dificuldades enfrentadas pelos sistemas que dependem do conhecimento codificado sugerem que os sistemas de IA precisam da capacidade de adquirir seu próprio conhecimento, extraindo padrões de dados brutos (GOODFELLOW *et al.*, 2016). Usando algoritmos de autoaprendizagem do campo de aprendizado de máquina, pode-se transformar esses dados em conhecimento (RASCHKA; MIRJALILI, 2019). Há diferentes abordagens para o aprendizado

de máquina, como o aprendizado supervisionado, aprendizado não supervisionado, aprendizado semi-supervisionado e aprendizado por reforço. No aprendizado supervisionado o treinamento é realizado através de dados previamente categorizados e rotulados, a exemplo de *Random Forest* (BREIMAN, 2001), *Naive Bayes* (RISH *et al.*, 2001), Redes Neurais Artificiais (RNA) (YEGNA-NARAYANA, 2009), etc. Para o aprendizado não supervisionado não há a necessidade de dados rotulados, como os algoritmos de clusterização, *k-means* (GÉRON, 2022), Redes Adversárias Generativas (RAG) (GOODFELLOW *et al.*, 2020), etc. Para o aprendizado semi-supervisionado, podem ser utilizados algoritmos que aprendem a partir de dados já rotulados e aplica a rotulação para dados não-rotulados, como o (ROSENBERG *et al.*, 2005). Já o aprendizado por reforço consiste em aprender o que fazer, e como mapear situações para ações maximizando um sinal de recompensa numérica (SUTTON; BARTO, 2018). Ainda dentro campo de Aprendizado de Máquina, há o Aprendizado Profundo que se torna cada vez mais popular devido a utilizar redes neurais tornando possível tarefas de reconhecimento de fala, processamento de linguagem natural e visão computacional.

Segundo Raschka e Mirjalili (2019), o aprendizado profundo pode ser entendido como um subcampo do aprendizado de máquina que se preocupa em treinar RNA com muitas camadas de forma eficiente. O aprendizado profundo moderno incorpora dezenas ou até centenas de camadas consecutivas de representações, todas as quais são aprendidas automaticamente mediante a exposição aos dados de treinamento. Em contraste, outras abordagens podem apresentar apenas uma ou duas camadas, sendo denominadas como aprendizado superficial (CHOLLET, 2021). Quanto mais profunda é a rede neural, mais camadas de representação ela possui. Na Figura 2 pode ser vista a representação de uma rede neural profunda.

Figura 2 – Representação de uma Rede Neural Profunda.



Fonte: elaborada pelo autor

2.4 Aprendizado Profundo para Detecção de Objetos

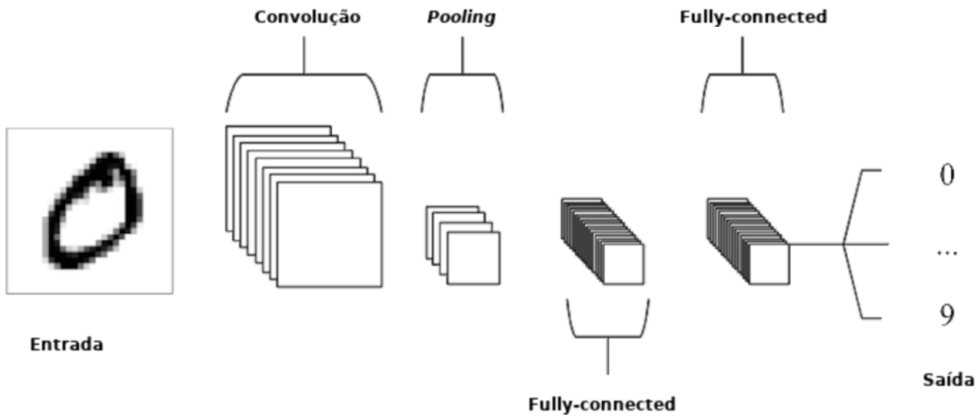
Tarefas como detecção de objetos e classificação de imagens são tipicamente problemas de Visão Computacional. A detecção de objetos localiza e identifica objetos presente em imagens. Já a classificação de imagens identifica classes em que imagens podem pertencer. Nesta seção, são apresentados os principais conceitos de Redes Neurais Convolucionais (RNC) e os métodos de detecção de objetos MnasNet, MobileNet e YOLO.

2.4.1 Redes Neurais Convolucionais

As RNC foram popularizadas por (LECUN *et al.*, 1998), sendo muito utilizadas no campo da visão computacional em processos de reconhecimento de objetos e classificação de imagens, apesar de não serem limitadas apenas a problemas deste tipo.

Redes Neurais Convolucionais realizam uma operação matemática chamada de convolução. De maneira geral, redes convolucionais são redes neurais que utilizam a operação de convolução em vez da multiplicação geral em pelo menos uma de suas camadas (GOODFELLOW *et al.*, 2016). A Figura 3 mostra uma rede neural convolucional simples de apenas cinco camadas.

Figura 3 – Arquitetura de uma RNC simples de apenas cinco camadas.

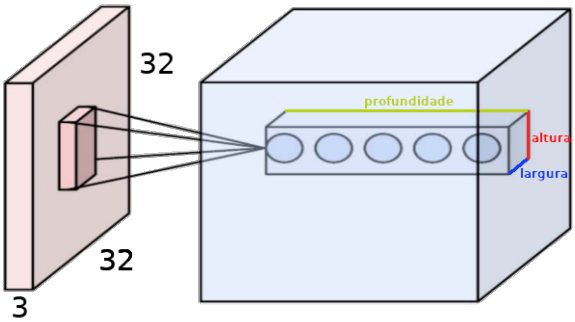


Fonte: adaptada de O'Shea e Nash (2015)

Matrizes podem ter múltiplas dimensões, sendo chamadas de tensores. A operação de convolução ocorre sobre tensores de três dimensões conhecidos como mapas de características, com dois eixos espaciais (altura e largura), e um eixo de profundidade (CHOLLET, 2021).

Durante a operação de convolução são produzidos mapas de características de saída, que é um tensor de três dimensões com largura e altura, entretanto, sua profundidade é arbitrária, uma vez que a profundidade de saída é um parâmetro da camada que representa filtros, ou também conhecidos como *kernels* (CHOLLET, 2021). A Figura 4 mostra a representação de uma RNC.

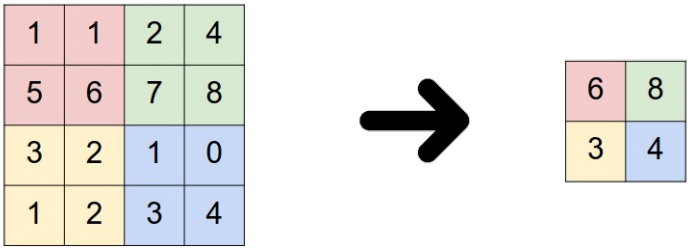
Figura 4 – Múltiplas camadas que correspondem a diferentes filtros na mesma região de imagem.



Fonte: adaptada de Albawi *et al.* (2017)

A camada de convolução aplica vários filtros a suas entradas, sendo capaz de detectar características em qualquer parte de sua entrada. Já camada de *pooling* faz a redução de dimensionalidade espacial da entrada fornecida para reduzir o número de parâmetros e a carga computacional (GÉRON, 2022). A Figura 5 mostra a representação de uma camada de *max-pooling* que é o tipo mais comum de camada de *pooling*. Na operação de *max-pooling*, cada quadrante representado por uma cor na imagem tem o máximo valor utilizado para representar a camada após a redução de dimensionalidade espacial.

Figura 5 – Operação de *Max-Pooling* em uma imagem 4x4 e filtro 2x2.



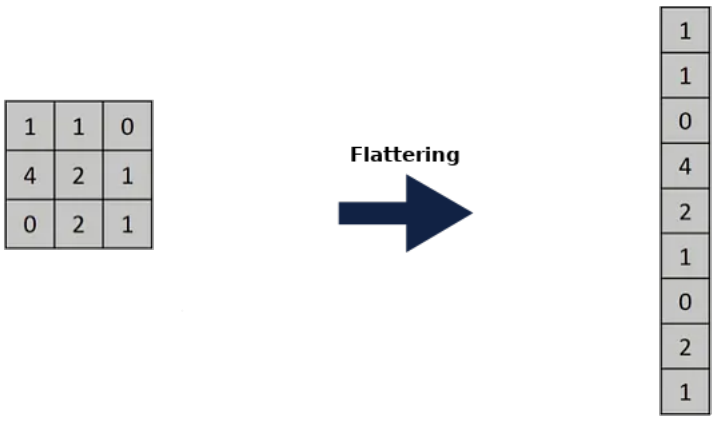
Fonte: adaptada de Andrej Karpathy (2015)

Nas camadas totalmente conectadas, também conhecidas como camadas densas, cada um dos neurônios é conectado a todos os neurônios da camada seguinte.

Entre a camada de *pooling* e a camada densa há a camada de *Flatten*, que é res-

ponsável por converter a dimensão dos dados após a aplicação de *pooling* em um formato unidimensional. A Figura 6 mostra a operação de Flatten.

Figura 6 – Operação de *Flatten*.



Fonte: adaptada de Muhammad Shoaib Ali (2022)

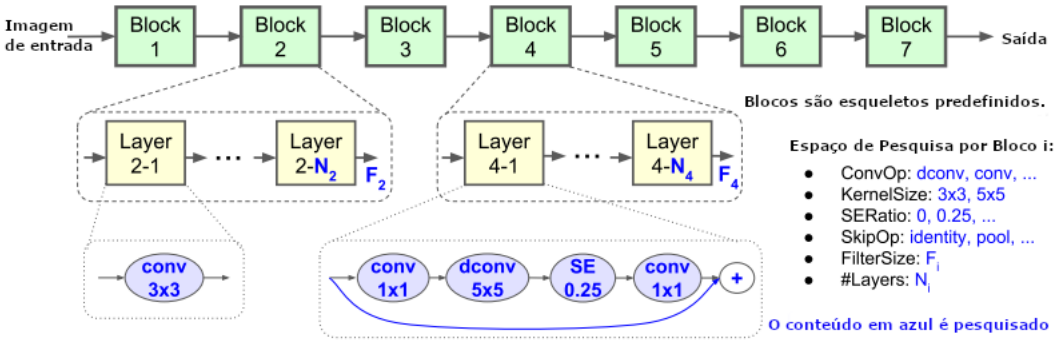
2.4.2 MnasNet

Dispositivos móveis possuem baixo poder computacional, sendo um problema ao utilizar redes neurais profundas que estão cada vez mais complexas, requisitando de grande poder computacional em busca de maior precisão. Isto posto, modelos de redes neurais para dispositivos móveis foram construídos para serem cada vez mais eficientes.

O *Mobile Neural Architecture Search* (MnasNet) equilibra precisão e desempenho para dispositivos móveis. Segundo Tan *et al.* (2019), no projeto do MnasNet é proposta uma abordagem de busca de arquitetura neural automatizada para projetar modelos de RNC móveis. Ainda segundo Tan *et al.* (2019), trabalhos anteriores de abordagem automatizada procuram por um tipo de célula na RNC e empilham repetidas vezes as mesmas células através da rede, simplificando o processo de busca, mas impedindo a diversidade de camadas que é importante para a eficiência computacional. Desta forma, Tan *et al.* (2019) propõem um novo espaço de busca hierárquico fatorado, que permite que as camadas sejam arquitetonicamente diferentes, mas ainda atinge o equilíbrio certo entre flexibilidade e tamanho do espaço de busca. Ademais, o propósito do projeto é formulado como um problema multiobjetivo que considera tanto a precisão quanto a latência de inferência dos modelos RNC, considerando dispositivos móveis reais para medir a latência, ao contrário de trabalhos anteriores como o *Neural Architecture Search Network* (NASNet) (ZOPH *et al.*, 2018) que utiliza FLOPS para obter uma latência de

inferência aproximada (TAN *et al.*, 2019). A Figura 7 mostra o espaço de busca hierárquico fatorizado.

Figura 7 – Espaço de Busca Hierárquico Fatorizado.



Fonte: adaptada de Tan *et al.* (2019)

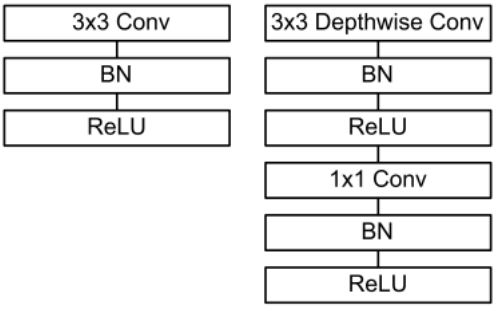
2.4.3 MobileNet

Os MobileNets são uma série de modelos para dispositivos móveis que foram iniciados com o MobileNet (HOWARD *et al.*, 2017) introduzindo convoluções separáveis em profundidade como um substituto eficiente para as camadas de convolução tradicionais. Segundo Howard *et al.* (2019), convoluções separáveis em profundidade fatoram a convolução tradicional, separando a filtragem espacial do mecanismo de geração de características, reduzindo drasticamente a computação e o tamanho do modelo. Já o modelo MobileNetV2 (SANDLER *et al.*, 2018) introduziu uma camada de expansão usando um bloco de expansão-filtragem-compressão, conhecido como bloco residual invertido que permite melhorias de performance. Ainda segundo (HOWARD *et al.*, 2019), o MobileNetV3 adiciona uma camada de *squeeze* e camada de excitação no inicio do bloco de construção retirado do MobileNetV2 e utiliza o *Neural Architecture Search* (NAS) para pesquisar as estruturas de rede global otimizando cada bloco de rede e recorre ao algoritmo NetAdapt para pesquisar por camada o número de filtros. A Figura 8 mostra uma comparação entre uma camada Convolucional Padrão com normalização em *batch* e *ReLU* (à esquerda) e uma Convolução Separável em Profundidade (à direita) com camadas profundas e pontuais seguidas por normalização em *batch* (BN) e *ReLU*.

2.4.4 YOLO

O YOLO foi proposto por (REDMON *et al.*, 2016) e consiste em uma série de detectores que são o estado da arte para a detecção de objetos em aplicações de tempo real, devido

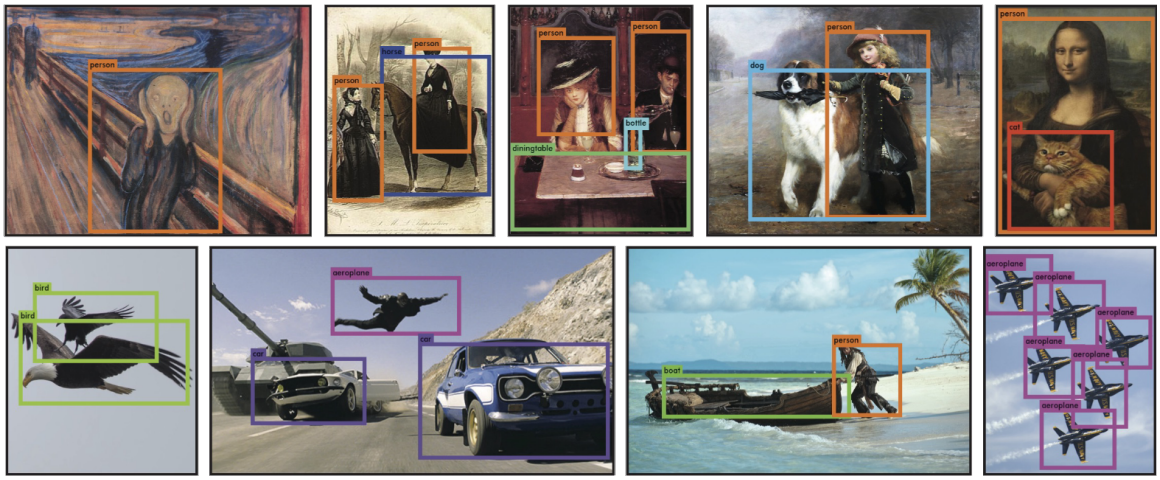
Figura 8 – Camada Convolucional Padrão X Convoluções Separáveis em Profundidade.



Fonte: Howard *et al.* (2017)

ao seu desempenho balanceado com precisão. A detecção no YOLO é alcançada utilizando apenas um estágio de detecção para a extração de características da imagem a partir de uma RNC. Conforme Redmon *et al.* (2016), uma única RNC prevê simultaneamente várias caixas delimitadoras e probabilidades de classes de objetos para cada uma destas caixas. Desta maneira, o YOLO consegue um bom desempenho e precisão para tarefas de detecção de objetos. A Figura 9 mostra a detecção de objetos usando modelos YOLO.

Figura 9 – Detecção de objetos com YOLO.



Fonte: Redmon *et al.* (2016)

YOLOv3 traz melhorias em comparação com seus predecessores, apresentando uma rede extratora de características mais complexa que é um hibrido entre a rede utilizada no YOLOv2, Darknet-19 e uma nova rede residual (REDMON; FARHADI, 2018). Também é possível usar diferentes estruturas de base para a rede neural no YOLOv3, possuindo uma implementação que utiliza o MobileNetV2 como *backbone* da rede para obter um bom desempenho em dispositivos com baixo custo computacional.

O YOLOv6 (LI *et al.*, 2022) traz um melhor *trade-off* em termos de precisão e velocidade de predições. O principal diferencial do YOLOv6 é o fato de ser *anchor-free*, tendo

melhor capacidade de generalização e simplicidade na decodificação dos resultados de previsão e sendo 51% mais rápido que as séries YOLO *anchor-based*. Detecção de objetos baseadas em *anchor* consiste em caixas delimitadoras pré-definidas como propostas da verdade. Ademais, também conta com melhorias pensadas para aplicações industriais, como longos períodos de treinamento, quantização (HAN *et al.*, 2015) para a redução do número de bits dos pesos da rede. O YOLOv6 v3.0 (LI *et al.*, 2023) é uma evolução do trabalho YOLOv6 (LI *et al.*, 2022) e traz um novo projeto para a rede e a estratégia de treinamento. De acordo com Li *et al.* (2023), na prática, houve a integração de recursos para várias escalas, tornando-se um fator crítico e eficaz na detecção de objetos. Este novo projeto de rede em conjunto com as novas estratégias de treinamento impulsionam o YOLOv6 v3.0 para a detecção de objetos em tempo-real e com alta precisão.

O YOLOv7 introduz uma nova abordagem aos métodos de detecção de objetos em tempo real. Em vez de simplesmente otimizar a arquitetura, os métodos propostos pelos autores enfatizam o aprimoramento do processo de treinamento, focando na melhoria de módulos específicos e técnicas de otimização destinadas a aumentar a precisão na detecção de objetos sem aumentar os custos de inferência (WANG *et al.*, 2023). Esses módulos especializados e técnicas de otimização são referidos como *trainable bag-of-freebies*, prometendo um desempenho significativo para aplicações em tempo real sem comprometer a precisão. As melhorias introduzidas pelo *trainable bag-of-freebies* podem incluir técnicas como aumentação de dados, regularização, ou outras modificações que não sobrecarregam excessivamente o modelo durante a fase de inferência, mas contribuem para um aprendizado mais eficaz durante o treinamento.

2.4.5 Métricas de Avaliação

Idealmente, as métricas para a avaliação de modelos de detecção de objetos devem mostrar o melhor modelo com base na aplicação desejada.

2.4.5.1 Saídas na Detecção de Objetos

Os *Ground Thuths* (GT) podem ser compreendidos como verdade fundamental, isto é, os objetos reais nas imagens, que são obtidos durante o processo de rotulação das imagens do conjunto de dados. Durante o treino e teste do modelo de detecção de objetos, os GT são utilizados para verificar o quão próximo o objeto predito está do objeto real (SHAH, 2022). A seguir são listadas as possíveis saídas para tarefas de detecção de objetos.

- **Verdadeiro Positivo:** O modelo prevê um rótulo que corresponde com o a verdade fundamental.
- **Falso Positivo:** O modelo prevê um rótulo que não faz parte da verdade fundamental.
- **Falso Negativo:** O modelo não prevê um rótulo, porém faz parte da verdade fundamental.

2.4.5.2 Intersection over Union (IoU)

Intersection over Union (IoU) é uma métrica comumente utilizada em visão computacional para avaliar o desempenho de algoritmos de detecção de objetos, especialmente em tarefas como segmentação de imagem e localização de objetos. Ela mede a sobreposição entre a caixa delimitadora prevista e a caixa delimitadora verdadeira de um objeto.

O IoU é calculado tomando a razão entre a área de interseção entre as caixas delimitadoras prevista e verdadeira e a área de sua união. A fórmula para calcular o IoU é mostrada na Equação 2.1.

$$IoU = \frac{\text{Área de Interseção}}{\text{Área de União}} \tag{2.1}$$

O resultado do cálculo do IoU é um valor entre 0 e 1. De acordo com Hui (2018), geralmente, um limiar pré-definido de IoU com um valor de 0,5 é utilizado para classificar se a predição é um verdadeiro positivo ou um falso positivo.

2.4.5.3 Precision

Precision (precisão) mede a proporção de positivos reais previstos de maneira correta. Pode ser obtida pela Equação 2.2. Outra forma de compreender a precisão é que ela está associada às instâncias previstas como positivas, e quantas destas instâncias realmente são positivas.

$$P = \frac{\text{Verdadeiro Positivo}}{\text{Verdadeiro Positivo + Falso Positivo}} \tag{2.2}$$

A precisão é importante em situações onde falsos positivos são críticos, isto é, quando é desejado ter certeza de que as instâncias previstas como positivas são realmente positivas.

2.4.5.4 *Recall*

Recall mede a proporção de positivos reais que são previstos de maneira correta. Pode ser obtida pela Equação 2.3. De maneira intuitiva, o *recall* está relacionado às instâncias positivas que realmente são positivas e qual o número destas instâncias que o modelo conseguiu prever corretamente.

$$R = \frac{\text{Verdadeiro Positivo}}{\text{Verdadeiro Positivo + Falso Negativo}} \tag{2.3}$$

O *recall* é crucial em situações em que falsos negativos são mais críticos, como em sistemas de detecção de doenças, onde perder um caso positivo é mais grave do que um falso positivo.

2.4.5.5 *Curva de Precision-Recall e Average Precision(AP)*

Calcular a precisão e o *recall* para cada limite de confiança permite que seja preenchida a curva de *precision-recall*. A área sob esta curva é conhecida como *Average Precision* (AP) (SZELISKI, 2022).

2.4.5.6 *mean Average Precision (mAP)*

O *Mean Average Precision* (mAP) é uma métrica amplamente utilizada em visão computacional, especialmente em tarefas de detecção de objetos e segmentação de imagem. É uma extensão da métrica de AP e é empregada para avaliar o desempenho geral de um modelo de detecção de objetos em várias classes (GÉRON, 2022).

Seja o AP a área sob a curva *precision-recall* e N o número de classes, o mAP pode ser dado pela equação 2.4.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i \tag{2.4}$$

2.4.5.7 *Average Recall*

A métrica *Average Recall* (AR) avalia a capacidade de um modelo de detecção de objetos de reconhecer e recuperar objetos em diversos limiares de IoU. Ela quantifica o desempenho do modelo medindo a contagem de detecções verdadeiras positivas alcançadas

em uma gama de níveis de sobreposição entre as caixas delimitadoras previstas e as caixas delimitadoras de referência (GT).

3 TRABALHOS RELACIONADOS

Nesta seção, são apresentados trabalhos relacionados que auxiliam no desenvolvimento da aplicação proposta neste trabalho.

3.1 *Detecting Customer Induced Damages in Motherboards with Deep Neural Networks*

No trabalho de Alves *et al.* (2022), os autores elaboram um modelo de visão computacional que através de métodos de aprendizado profundo identifica e classifica danos causados pelo consumidor na superfície e componentes de placa mãe de dispositivos eletrônicos. Os autores realizam a aplicabilidade de conceitos abordados em seu trabalho, como redes neurais profundas, arquiteturas de redes *Mask R-CNN* (HE *et al.*, 2017), *Soft Teacher* (XU *et al.*, 2021), e *Swin* (LIU *et al.*, 2021) e danos causados pelos clientes.

Três experimentos foram realizados, considerando cada um dos modelos de redes neurais profundas *Mask R-CNN*, *Soft Teacher* e *Swin*, utilizando como métricas de desempenho da previsão da região de dano o *Intersection over Union* (IoU) em conjunto com *Average Precision* (AP) e *Average Recall* (AR) e verificaram que o modelo *Swin* obteve melhores resultados que as outras arquiteturas, sendo protagonizado pelo Swin-S com AP de 42,4%, AP@0,5 de 80,2% e AP@0,75 de 39,8%.

3.2 *Defect Detection in Printed Circuit Boards Using You-Only-Look-Once Convolutional Neural Networks*

No trabalho de Adibhatla *et al.* (2020), os autores utilizam um algoritmo de aprendizado profundo baseado no Tiny-YOLOv2. Neste estudo, os autores desenvolveram um classificador a partir de RNC que reconhece vários componentes elétricos em uma PCB e, em sequência localizar e detectar defeitos nos componentes do PCB.

Para avaliar o desempenho do modelo, os autores utilizaram a *Fivefold cross validation*, onde os dados foram divididos em cinco segmentos iguais de maneira aleatória, onde quatro destes segmentos foram utilizados para treinamento e o restante para teste. O procedimento foi realizado cinco vezes no conjunto de dados de treinamento e teste. A acurácia do modelo foi de 98,79% para *batch* de tamanho 32, e a precisão constantemente de 0,99.

3.3 *A Real-time PCB Defect Detector Based on Supervised and Semi-supervised Learning*

Diferente dos trabalhos apresentados nas Seções 3.1 e 3.2, o trabalho de He *et al.* (2020) utiliza em conjunto o aprendizado supervisionado através de redes neurais profundas e o aprendizado semi-supervisionado. Conforme os autores, um desafio enfrentado por redes neurais profundas é a grande quantidade de dados necessários para o treinamento, os dados tem que está rotulados e o processo de rotulação é custoso para grande quantidades de dados. Desta maneira, o aprendizado semi-supervisionado é proposto como uma solução para a rotulação de grandes conjuntos de dados não-rotulados.

Os autores utilizaram a Consistência de Peso Médio (TARVAINEN; VALPOLA, 2017) como pilar do aprendizado semi-supervisionado para a rotulação dos dados não-rotulados. Para a detecção de objetos, foi proposto um novo módulo profundo que combina recursos de forma eficiente em diferentes resoluções e faz previsões para detectar defeitos de PCB em várias escalas. O modelo atingiu resultados de 98.6% mAP @ 62 FPS para o conjunto de dados DeepPCB, disponibilizado pelos autores.

3.4 Comparação entre os trabalhos relacionados

O Quadro 1 traz as principais características de cada um dos trabalhos relacionados apresentados e o trabalho proposto. O termo FLOPS corresponde ao número de operações de ponto flutuante por segundo, sendo uma medida da capacidade computacional de um computador. A latência e o FPS presentes na tabela foram obtidos através dos artigos originais de cada um dos trabalhos relacionados, desta forma, os hardwares utilizados para capturar destes resultados não necessariamente são comparáveis por tratarem-se de hardwares diferentes. Informações não obtidas durante a pesquisa são representadas por hífen (-).

Quadro 1 – Quadro comparativo entre os trabalhos relacionados.

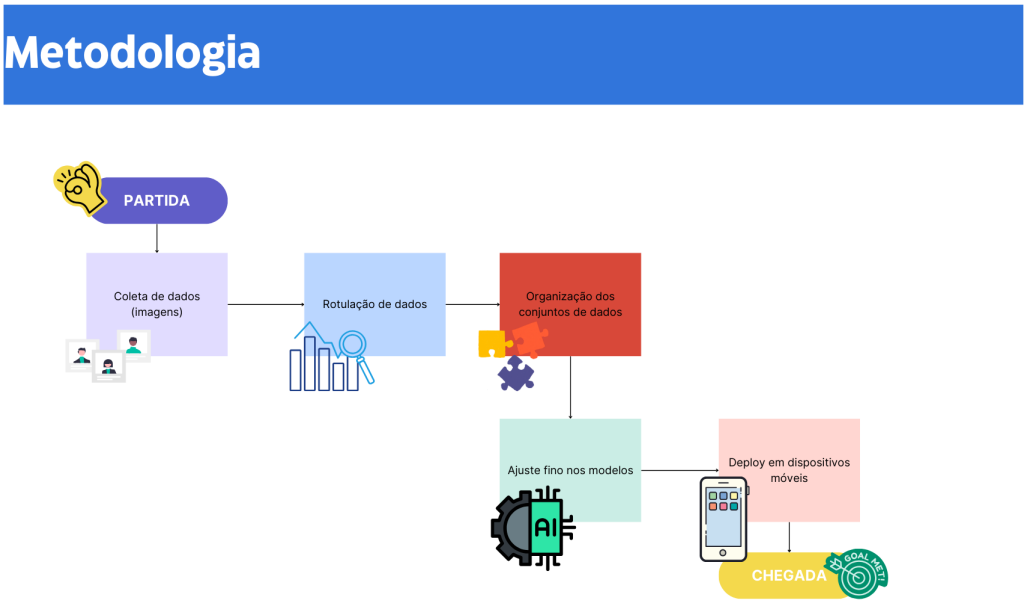
Trabalho	**Arquitetura usada**	**FLOPS (G)**	**Latência (ms)**	**FPS**	**Métrica**
Alves *et al.* (2022)	Swin-S	359	-	437	mAP
Adibhatla *et al.* (2020)	Tiny-YOLO-v2	5,41	-	244	Acurácia
He *et al.* (2020)	VGG16-tiny	15,3	128,62	-	mAP
Trabalho Proposto	YOLOv3-MobileNetV2	-	-	-	mAP
	YOLOv6-Lite-L	0,24	3,63	275,06	mAP
	YOLOv6-N	11,4	0,842	1187	mAP
	YOLOv6-S	45,3	2,07	484	mAP
	YOLOv6-M	85,8	4,42	226	mAP
	YOLOv6-L	150,7	8,62	116	mAP
	YOLOv6-N6	48,9	3,56	281	mAP
	YOLOv6-S6	198,0	9,26	108	mAP
	YOLOv7	-	2,8	161	mAP
	YOLOv7-X	-	4,3	114	mAP
	YOLOv8-N	8,7	80,4	12,44	mAP
	YOLOv8-S	28,6	128,4	7,79	mAP
	YOLOv8-M	78,9	234,7	4,26	mAP
	YOLOv8-L	165,2	375,2	2,66	mAP

Fonte: elaborado pelo autor.

4 MATERIAIS E MÉTODOS

Neste capítulo são apresentados os passos necessários para a elaboração deste trabalho. As seções seguintes descrevem os processos metodológicos, manipulação de dados, técnicas e procedimentos da detecção de objetos voltada para a detecção de defeitos em placas de circuito impresso. A Figura 10 mostra a representação da metodologia proposta neste trabalho.

Figura 10 – Processo metodológico.



Fonte: elaborada pelo autor

4.1 Seleção e métricas

Os critérios para a seleção da arquitetura de redes neurais profundas neste trabalho são os seguintes: i) A métrica de avaliação principal é o mAP, oferecendo uma avaliação abrangente do desempenho do modelo em tarefas de detecção de objetos. ii) O *Average Recall* também faz parte dos critérios por ser uma métrica que mede a habilidade do sistema para identificar corretamente todas as instâncias de um objeto alvo que pertence a uma classe do conjunto de dados. iii) Foi estabelecido um limite de 200G FLOPs para o custo computacional dos modelos treinados durante os experimentos para manter a consistência entre complexidade do modelo e limitação do hardware de dispositivos móveis. Este teto de FLOPs foi definido de maneira experimental relacionado aos hardwares de dispositivos disponíveis para os testes de *deploy*, de modo que romper este limite pode fazer com que os modelos demonstrem experiências

ruins ao executar nos dispositivos utilizados. iv) A quantidade de Parâmetros treináveis da rede que é associado a capacidade e complexidade de aprendizado dos modelos.

4.2 Seleção de técnicas e detecção de objetos

Na Seção 2.4.4, foram introduzidos métodos baseados na arquitetura YOLO, que são considerados estado-da-arte para a detecção de objetos em alguns cenários. Esses métodos foram avaliados no conjunto de dados COCO[1], que é um conjunto de dados comumente utilizado em detecção de objetos com propósitos gerais. As arquiteturas de redes neurais utilizada nos experimentos foram selecionadas baseado em seu desempenho no conjunto de dados COCO e o resultado obtido nas métricas introduzidas na Seção 4.1.

Os modelos escolhidos para os experimentos foram: i) YOLOv3-MobileNetV2, que utiliza o MobileNetV2 como espinha dorsal da rede neural; ii) YOLOv6, que apresenta versões otimizadas para dispositivos móveis; iii) YOLOv7, que promete bom desempenho em aplicações em tempo real; e iv) YOLOv8 (JOCHER *et al.*, 2023), devido à sua recente popularidade.

Os quatro modelos de diferentes arquiteturas baseadas em YOLO foram treinados e validados usando subconjuntos do conjunto de dados rotulado. Foram realizados dezesseis experimentos com diferentes arquiteturas de modelos, um para YOLOv3-MobileNetV2, nove com YOLOv6 (variações Lite-L, N, S, M, L, N6 e S6), dois com YOLOv7 (YOLOv7 e YOLOv7-X) e quatro com YOLOv8 (variações N, S, M e L). O Quadro 2 mostra os modelos e variações utilizadas durante os experimentos, é importante verificar que o YOLOv6 possui nove experimentos considerando experimentos realizados com as variações Lite-L utilizando diferentes resoluções de imagens (320x320, 640x640 e 1280x1280).

As variações das quatro diferentes arquiteturas YOLO são definidas na seguinte lista:

- **MobileNetV2:** O *backbone* da rede utilizado é o MobileNetV2;
- **Lite-L:** Modelo otimizado para dispositivos móveis presente na arquitetura YOLOv6;
- **N:** relativo ao termo (nano) e representa um modelo geralmente de complexidade baixa;
- **S:** relativo ao termo (*small*), isto é, pequeno. Está relacionados a modelos mais complexos que os modelos nano;
- **M:** relativo ao termo (*medium*), ou seja, médio. Relacionado a modelos mais complexos que modelos *small*;
- **L:** relativo ao termo (*large*), significando grande. Relacionado a modelos mais complexos

[1] Disponível em: https://cocodataset.org/

Quadro 2 – Variações de arquitetura entre os modelos.

Modelo	FLOPS (G)	Parâmetros (M)	FPS	Latência (ms)
YOLOv3-MobileNetV2	-	3,50	-	-
YOLOv6-Lite-L	0,24	1,06	275,06	3,63
YOLOv6-N	11,4	4,63	1187	0,842
YOLOv6-S	45,3	18,50	484	2,07
YOLOv6-M	85,8	34,80	226	4,42
YOLOv6-L	150,7	59,54	116	8,62
YOLOv6-N6	48,9	10,34	281	3,56
YOLOv6-S6	198,0	41,32	108	9,26
YOLOv7	-	36,49	161	2,8
YOLOv7-X	-	70,79	114	4,3
YOLOv8-N	8,7	3,2	12,44	80,4
YOLOv8-S	28,6	11,12	7,79	128,4
YOLOv8-M	78,9	25,84	4,26	234,7
YOLOv8-L	165,2	43,60	2,66	375,2

Fonte: elaborado pelo autor.

que os modelos *medium*;

– **X:** relativo ao termo (*extra-large*), extra-grande. Está relacionado a modelos muito grandes, normalmente os maiores produzidos pela arquitetura de rede neural.

Cada um dos modelos foi treinado até convergir. A convergência ocorre durante o processo de treinamento dos modelos quando o erro de treinamento atinge um nível aceitável, de modo a indicar que o modelo aprendeu a representar de maneira satisfatória os padrões nos dados de treinamento.

4.3 Obtenção do conjunto de dados

As imagens utilizadas como base de dados são provenientes de centros de reparo de uma fabricante de dispositivos eletrônicos do setor privado. A captura de fotos das imagens foi realizada entre 01 de julho de 2022, e 26 de janeiro de 2023 contendo aproximadamente 82 mil imagens não rotuladas, que compõe um repositório de imagens brutas. Todas as imagens possuem ao menos um DIC de acordo com os técnicos que analisam as PCBs e realizaram a captura das imagens.

Foram selecionadas 6383 imagens para a rotulação dos dados através de um método de seleção aleatória.

A rotulação das imagens foi realizada com a ferramenta *Label Studio*[2]. Das imagens

[2] Disponível em: https://labelstud.io/

selecionadas para a rotulação, 4223 imagens foram descartadas por apresentarem algum problema de qualidade de imagem, a exemplo de iluminação, baixa qualidade de imagem e danos muito pequenos. Sendo assim, a taxa de descartes de imagens foi de 65,86%.

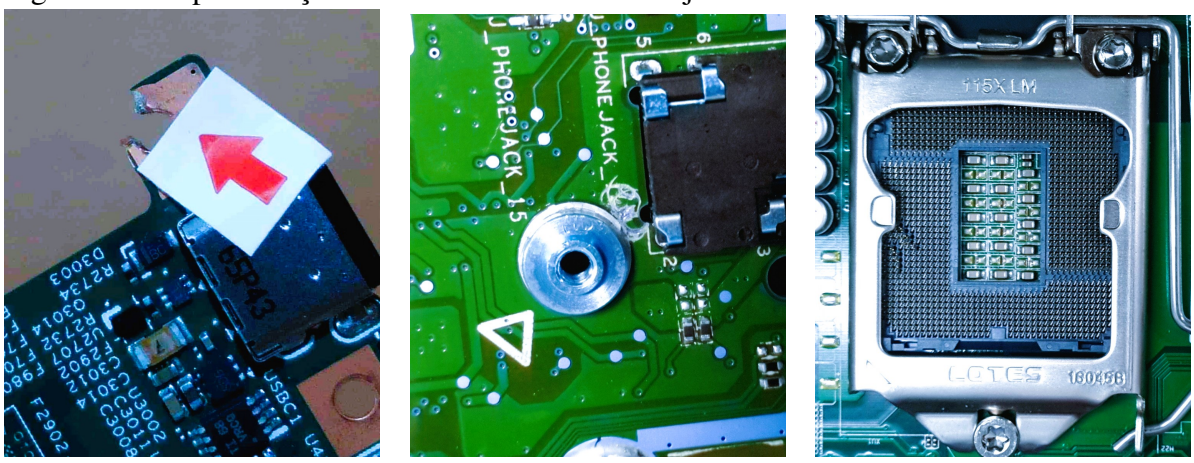
O conjunto de dados rotulado possui 2179 imagens, com 514 imagens para validação, 1468 imagens de treinamento e 197 imagens de teste, totalizando 9% para teste, 67,4% para treinamento e 23,6% para validação. A Figura 11 mostra a representação das classes de DICs presentes no conjunto de dados rotulado. A distribuição de imagens por classe de DIC nos conjuntos de treinamento, validação e teste são apresentados na Tabela 1.

Tabela 1 – Distribuição de imagens por classe para os conjuntos de dados de treinamento e validação.

Classe	Conjunto de validação	Conjunto de treino	Conjunto de teste
S	188	557	72
P	81	225	71
D	245	686	58
Total	**514**	**1468**	**197**

Fonte: elaborada pelo autor.

Figura 11 – Representação de classes de DICs no conjunto de dados.



(a) Classe D, *break* (b) Classe S, *scratch* (c) Classe P, *pin damage*

Fonte: elaborada pelo autor.

Toda a rotulação foi realizada por um especialista que marcou e classificou os DICs nas imagens. O processo de rotulação é realizado identificando o dano, seguido pela delimitação da região danificada de acordo com a classe de dano. O conjunto de dados possui um total de 2179 imagens com 2529 danos rotulados. A Tabela 2 mostra a distribuição das anotações por classe no conjunto de dados.

Tabela 2 – Distribuição de classes no conjunto de dados.

Classe	Quantidade
S	927
P	568
D	1034
Total	**2529**

Fonte: elaborada pelo autor.

4.4 Pré-Processamento

As anotações rotuladas com a ferramenta *Label Studio* são exportadas em um formato JSON[3] próprio do *Label Studio* e convertidas com um utilitário pertencente ao *Label Studio* para o formato COCO[4]. O formato de conjunto de dados COCO é composto por um arquivo JSON que fornece informações sobre o conjunto de dados e suas imagens, como o tamanho das imagens, caminho em que a imagem se encontra e as anotações contendo as coordenadas do posicionamento das caixas delimitadoras dos objetos em cada imagem, conforme mostrado no Código-Fonte 4.4.1.

Listing 4.4.1 – Anotação no formato COCO.

```
{
   "annotations": [
       {
           "id": 1,
           "image_id": 1,
           "category_id": 1,
           "bbox": [0.1, 0.2, 0.3, 0.4],
           "area": 0.12,
           "iscrowd": 0
       },
   ]
}
```

Fonte: Shah (2023)

Dos modelos treinados durante os experimentos, apenas o YOLOv6-MobileNetv2 utiliza o formato COCO no conjunto de dados. O restante das arquiteturas utilizam o formato de conjunto de dados YOLO. No formato YOLO, cada imagem rotulada possui um único arquivo de texto para suas anotações, de modo que se uma imagem não possui um arquivo de texto, então

[3] Disponível em: https://www.json.org/json-en.html
[4] Disponível em: https://cocodataset.org/

essa imagem não possui nenhum objeto a ser identificado. O formato do arquivo de texto das anotações YOLO é dado por id da classe, centro do x, centro do y, largura e altura, separados por espaços, conforme o Código-Fonte 4.4.2.

Listing 4.4.2 – Anotação no formato YOLO Darknet.

```
1 0 0.481719 0.634028 0.690625 0.713278
```

Fonte: adaptado de Rehman (2022)

Foi desenvolvido um código em Python que percorre as anotações no formato COCO e para cada imagem, captura as informações referentes a caixa delimitadora e gera um arquivo de texto no formato YOLO para a imagem, fazendo a conversão do conjunto de dados para o formato utilizado pelos modelos YOLOv6, YOLOv7 e YOLOv8.

4.5 *Fine-tuning*

O modelo é treinado com o conjunto de dados que rotulado a partir das imagens brutas, e é realizado o ajuste fino, do inglês (*fine-tuning*) do modelo. O *fine-tuning* é realizado para obter modelos aprimorados por meio de uma seleção de hiperparâmetros para treinamento do modelo. São considerados os seguintes parâmetros de treinamento para o ajuste fino do modelo:

- O número de lotes, do termo em inglês *batchs* que significa o tanto de imagens processadas simultaneamente no algoritmo de treinamento;
- A quantidade épocas de treinamento, onde uma época corresponde a processar uma quantidade de *batchs* que é equivalente a processar todo o conjunto de treinamento.
- Tamanho da imagem.

4.6 *Deploy* em dispositivos móveis

As etapas que serão necessárias para a conversão do modelo para o *deploy* em dispositivos móveis são listadas a seguir.

1. Nesta etapa é necessário serializar e otimizar o modelo do PyTorch para remover a dependência da linguagem de programação Python[5] através do módulo TorchScript[6].

[5] Disponível em: https://www.python.org/
[6] Disponível em: https://pytorch.org/docs/stable/jit.html

2. Nesta etapa o modelo é convertido para ser suportado pelo *framework* ncnn[7]. O ncnn é uma estrutura de computação de inferência de rede neural de alto desempenho otimizada para plataformas móveis, e não possui dependências de terceiros.
3. Por fim, uma aplicação para dispositivos Android[8] foi desenvovida com a linguagem Kotlin[9] utilizando o modelo treinado e convertido para o *framework* ncnn através do módulo de desenvolvimento OpenCV-mobile[10] para identificar danos causados por consumidores em PCBs por meio da câmera de dispositivos.

4.7 Procedimento Experimental

Uma máquina virtual na plataforma Google Colab[11] foi utilizada durante todos os experimentos realizados neste trabalho. As especificações da máquina de experimentação são descritas no Quadro 3. Os experimentos foram conduzidos com a versão do Python 3.10.12, PyTorch 2.0.1, e NVIDIA CUDA[12] 11.8.

Quadro 3 – Especificações do ambiente de experimentação.

OS	Linux 5.15.120+ #1 SMP x86_64 GNU-Linux
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz
Mem. RAM	51 GB
GPU	NVIDIA TESLA V100 (16 GB)

Fonte: elaborado pelo autor.

Nos experimentos, foi utilizado um conjunto diversificado de hiperparâmetros para ajuste fino em diferentes modelos YOLO. Para YOLOv3-MobileNetV2, foi aplicado um tamanho de *batch* de 16, dimensões de imagem de 320x320 e 50 épocas de treinamento. Para YOLOv6, foram exploradas várias complexidades de modelo, incluindo Lite-L, N, S, M, L, N6 e S6, juntamente com tamanhos de imagem de 320x320, 640x640 e 1280x1280, tamanhos de *batch* de 8, 16 e 32, e o número de épocas de treinamento variando de 100 a 400. Para YOLOv7, foram conduzidos experimentos usando as complexidades de modelo YOLOv7 e YOLOv7-X, um tamanho de *batch* de 16, um tamanho de imagem de 640x640 e um número fixo de 100 épocas de treinamento. Por fim, para YOLOv8, foram aplicados experimentos usando as variações de modelo N, S, M e L, com um tamanho de *batch* de 16 e um tamanho de imagem de

[7] Disponível em: https://github.com/Tencent/ncnn
[8] Disponível em: https://www.android.com/intl/pt-BR_br/
[9] Disponível em: https://kotlinlang.org/
[10] Disponível em: https://github.com/nihui/opencv-mobile
[11] Disponível em: https://colab.research.google.com/
[12] Disponível em: https://developer.nvidia.com/cuda-toolkit

640x640. Essas combinações de parâmetros foram selecionadas para investigar o desempenho e a versatilidade dos modelos YOLO em uma ampla gama de configurações.

O melhor modelo obtido nos experimentos foi selecionado para ser testado em dispositivos móveis reais e observar o comportamento do modelo durante inferências em tempo real.

5 RESULTADOS E DISCUSSÕES

A Tabela 3 mostra os resultados dos experimentos. Em uma análise geral, pode-se observar que o YOLOv6 obteve melhores resultados do que as outras arquiteturas avaliadas. O YOLOv6-N6 está destacado por ser o modelo que apresenta os melhores resultados em relação ao equilíbrio geral, considerando não apenas as métricas de mAP e AR, mas também em termos de custo de processamento atrelado aos FLOPs e parâmetros.

Tabela 3 – Resultados dos Experimentos.

Modelo	Resolução de Imagem	mAP@0.50	mAP@0.50:0.95	(AR) @[IoU=0.50:0.95]	FLOPs	Parâmetros
YOLOv3-MobilenetV2	320x320	0.055	0.014	0.065	0.32G	3.50M
YOLOv6 Lite-L	320x320	0.117	0.043	0.216	3.31G	1.06M
YOLOv6 Lite-L	640x640	0.214	0.076	0.291	3.31G	1.06M
YOLOv6-N	640x640	0.208	0.080	0.275	11.34G	4.63M
YOLOv6-S	640x640	0.294	0.099	0.264	45.17G	18.50M
YOLOv6-M	640x640	0.307	0.119	0.253	85.63G	34.80M
YOLOv6-L	640x640	0.331	0.129	0.285	150.50G	59.54M
YOLOv6 Lite-L	1280x1280	0.274	0.109	0.374	13.25G	1.06M
YOLOv6-N6	**1280x1280**	**0.331**	**0.124**	**0.347**	**49.57G**	**10.34M**
YOLOv6-S6	1280x1280	0.338	0.129	0.339	197.52G	41.32M
YOLOv7	640x640	0.03	0.011	0.044	103.2G	36.49M
YOLOv7-X	640x640	0.016	0.005	0.025	188G	70.79M
YOLOv8-N	640x640	0.211	0.076	0.255	8.1B	3.2M
YOLOv8-S	640x640	0.269	0.090	0.279	28.4G	11.12M
YOLOv8-M	640x640	0.237	0.088	0.274	78.7G	25.84M
YOLOv8-L	640x640	0.269	0.089	0.326	164.8G	43.60M

Fonte: elaborada pelo autor.

Ao analisar os resultados da AP e suas variações de limite de IoU, pode-se observar que os melhores resultados foram alcançados pelo YOLOv6-S6 com um AP @0.5 de 33,8% e um AP @0.5:0.95 de 12,9%. O segundo melhor resultado de AP é protagonizado pelo YOLOv6-L com AP @0.5 de 33,1% e AP @0.5:0.95 de 12,9%. O YOLOv6-N6 alcança resultados muito próximos aos obtidos com o YOLOv6-S6 e YOLOv6-L com um AP @0.5 de 33,1% e um AP @0.5:0.95 de 12,4%. Além de superar o YOLOv6-S6 na *Average Recall* (AR), outro ponto notável para o YOLOv6-N6 é seu custo computacional 3,98 vezes menor em FLOPs e 4 vezes menor em quantidade de parâmetros em comparação com o YOLOv6-S6, tornando-o o modelo com o melhor equilíbrio e considerado o modelo de melhor desempenho geral. Do ponto de vista da AR, o melhor resultado foi obtido pelo YOLOv6-Lite-L com uma resolução de imagem de 1280x1280, atingindo um AR @0.5:0.95 de 37,4%, superando YOLOv6-N6 e YOLOv6-S6, que alcançaram um AR @0.5:0.95 de 34,7% e 33,9%, respectivamente.

A família de modelos que obteve os melhores resultados foi a família YOLOv6, seguida pela família YOLOv8. A família que apresentou os piores resultados foi a família

YOLOv7, tendo um desempenho ainda pior do que o YOLOv3-MobileNetV2, que era o único representante da família de arquitetura YOLOv3.

As famílias de modelos YOLOv6 e YOLOv3 incluem modelos com arquiteturas de rede neural otimizadas para dispositivos móveis. Esses modelos são o YOLOv3-MobileNetV2 e o YOLOv6-Lite, projetados para operar de forma eficiente em hardware móvel. Em termos de FLOPs, o YOLOv3-MobileNetV2 é o modelo com o menor custo computacional. No entanto, seus resultados de AP e AR são prejudicados devido ao foco em obter elevado desempenho computacional e estão entre os três piores resultados de nossos experimentos.

Dentre os modelos otimizados para baixo custo computacional usando configurações padrão e com métricas de AP e AR melhores, está o YOLOv6-Lite-L, com uma resolução de imagem de 320x320 treinado por 400 épocas, alcançando apenas 3,31G de FLOPs e 1,06 milhão de parâmetros, com um AP@0.5 de 11,7% e um AR de 21,6%. Também há um experimento com o YOLOv6-Lite e uma resolução de imagem de 1280x1280 que obteve resultados melhores do que os resultados alcançados com arquiteturas otimizadas para dispositivos móveis, com um AP@0.5 de 27,4% e um AR de 37,4%. No entanto, o custo computacional em termos de FLOPs é de 13,25G, o que é maior do que o do YOLOv6-N, que não possui otimizações específicas para dispositivos móveis.

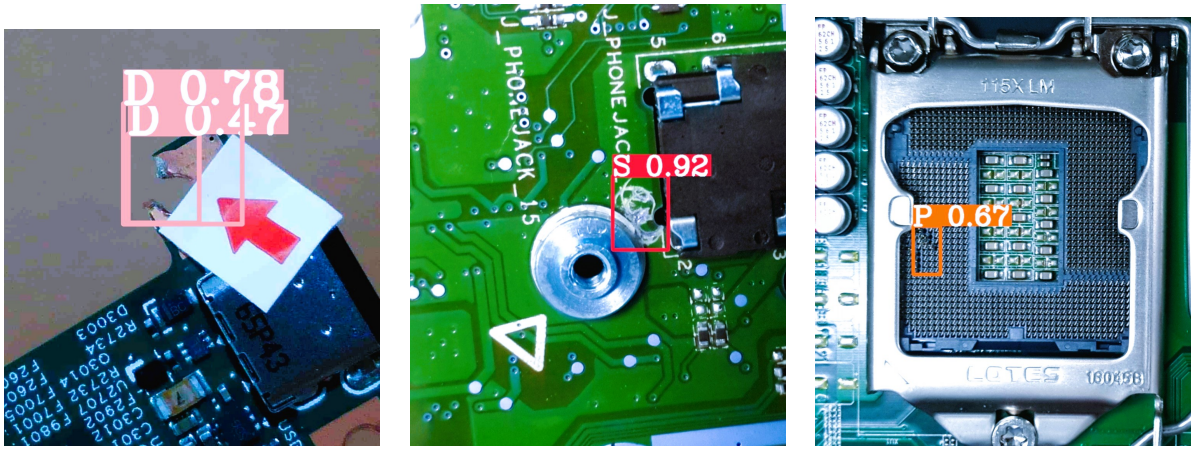
Uma particularidade entre as famílias YOLO treinadas durante os experimentos é que a família YOLOv6 pode calcular a AP e a AR durante o processo de validação para diferentes tamanhos de objetos (grandes, médios, pequenos). Isso permite determinar qual modelo obteve melhor desempenho para esses diferentes tamanhos de objetos. A Tabela 4 mostra o AP e o AR entre os modelos YOLOv6 para diferentes tamanhos de objetos.

Tabela 4 – Resultado para diferentes tamanhos de objetos (YOLOv6).

Modelo	AP^s	AP^m	AP^l	AR^s	AR^m	AR^l
YOLOv6-Lite-L (320x320)	0.000	0.038	0.053	0.000	0.203	0.238
YOLOv6-Lite-L (640x640)	0.000	0.061	0.106	0.000	0.219	0.392
YOLOv6-Lite-L (1280x1280)	0.000	0.094	0.135	0.036	**0.331**	**0.436**
YOLOv6-N	0.000	0.053	0.127	0.000	0.188	0.395
YOLOv6-S	0.005	0.095	0.136	0.011	0.247	0.330
YOLOv6-M	**0.009**	0.100	**0.169**	0.016	0.232	0.313
YOLOv6-L	0.000	0.116	**0.169**	0.002	0.268	0.344
YOLOv6-N6	0.004	**0.122**	0.150	0.024	0.307	0.416
YOLOv6-S6	0.008	0.111	0.160	**0.078**	0.304	0.398

Fonte: elaborada pelo autor.

Figura 12 – Inferência em amostras do conjunto de dados.



(a) Inferência em amostra da classe D

(b) Inferência em amostra da classe S

(c) Inferência em amostra da classe P

Fonte: elaborada pelo autor.

Os resultados obtidos para a AP de objetos de diferentes tamanhos nos permitem observar que todos os modelos têm dificuldades em detectar objetos pequenos, sendo o melhor deles o YOLOv6-M com um AP de menos de 9%. Para objetos de tamanho médio, o YOLOv6-N6 obteve os melhores resultados com 12,2%. Para objetos grandes, o YOLOv6-L e o YOLOv6-M tiveram o melhor desempenho, com um AP de 16,9%. O pior resultado foi do YOLOv6-Lite-L (320x320), obtendo o pior AP para cada um dos diferentes tamanhos.

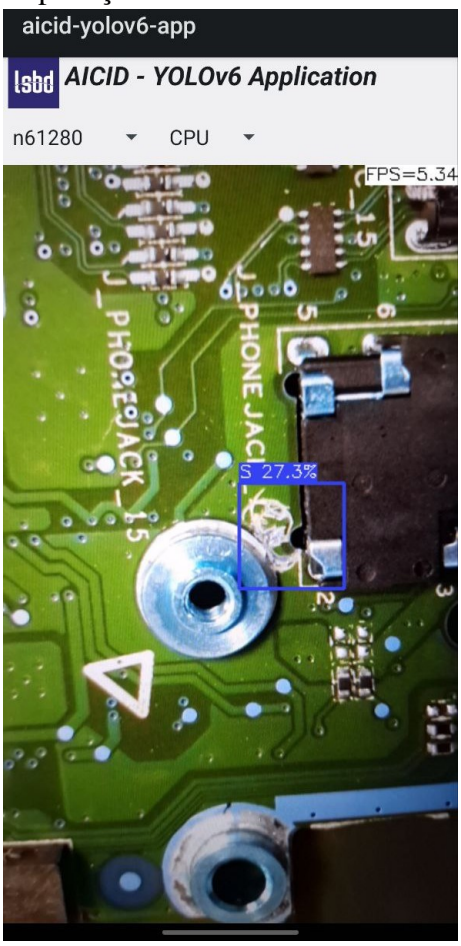
Avaliando a AR para objetos de diferentes tamanhos de detecção, podemos observar que, para objetos pequenos, o melhor modelo foi o YOLOv6-S6, alcançando 7,8%. O melhor modelo para objetos de tamanho médio foi o YOLOv6-Lite-L (1280x1280) com 33,1%. No caso de objetos grandes, mais uma vez, o modelo com os melhores resultados foi o YOLOv6-Lite-L (1280x1280), com 43,6%.

A Figura 12 mostra a inferência do modelo YOLOv6-N6 em uma amostra de imagens de cada classe do conjunto de dados. Podemos observar que o modelo se sai bem nessas imagens, atingindo um nível de confiança de 92% para a classe S, Figura 12b; 67% para a classe P, Figura 12c; e duas previsões na mesma imagem para a classe D com níveis de confiança de 47% e 78%, Figura 12a. Para a inferência nessas imagens, foi usado um limite de confiança de 0,4 e um limite de IoU de 0,45, que são valores padrão para a inferência de imagens na arquitetura YOLOv6.

5.1 Aplicação Móvel

Na parte superior da interface da aplicação, existem dois botões de seleção: um para escolher o modelo a ser usado e outro para alternar entre a CPU e a GPU do dispositivo. O restante da interface da aplicação é utilizado pela câmera traseira para capturar imagens e realizar inferências em tempo real. A aplicação também permite verificar o número de quadros por segundo (do inglês *frames per second* - FPS) nos quais o modelo realiza inferências nas imagens capturadas pela câmera, permitindo ao usuário avaliar em tempo real o desempenho de processamento de imagem para cada um dos diferentes modelos disponíveis.

Figura 13 – Demonstração da aplicação com o modelo YOLOv6-N6.



Fonte: elaborada pelo autor.

A aplicação não depende de recursos de terceiros, como infraestrutura de internet, APIs ou servidores na nuvem, e deve ser executada diretamente no dispositivo móvel onde a aplicação está instalada.

Conforme mencionado anteriormente, a melhor família de modelos YOLO obtidos através dos experimentos foi a família YOLOv6; portanto, esses modelos foram escolhidos para

serem validados na aplicação. Em geral, os modelos podem identificar os DICs em PCBs a partir de uma amostra de imagens que não foram apresentadas aos modelos durante o treinamento.

Os modelos foram testados em três dispositivos móveis com diferentes especificações de hardware. As especificações de hardware dos dispositivos testados podem ser vistas no Quadro 4.

Quadro 4 – Dispositivos móveis.

ID do Dispositivo	CPU	GPU	RAM
1	Helio P35 MediaTek MT6765	PowerVR GE8320	4 GB
2	Exynos 1330	Mali-G68 MP2	4 GB
3	Exynos 1380	Mali-G68 MP5	8 GB

Fonte: elaborado pelo autor.

O desempenho dos modelos em termos de processamento de imagem durante a inferência em cada um dos dispositivos varia dependendo do modelo atualmente em execução, com modelos de custo computacional mais baixo em FLOPs alcançando mais quadros por segundo do que dispositivos com custos computacionais mais altos.

Os modelos YOLOv6 (N, S, M e L) encontraram problemas de acesso à memória durante o processo de implantação em dispositivos móveis e, como resultado, foram excluídos dos testes em dispositivos móveis reais. Os modelos restantes foram testados, e tiveram a taxa média de quadros por segundo (FPS) registrada durante a execução desses modelos em diferentes dispositivos. A taxa média de FPS foi registrada exibindo a câmera do dispositivo por cinco minutos usando uma amostra de imagens que não foram usadas durante o treinamento do modelo. Tanto a GPU quanto a CPU dos dispositivos foram utilizadas para comparações de desempenho. A Tabela 5 mostra a taxa média de FPS com os modelos em execução na CPU e na GPU para cada um dos modelos avaliados em nossos três dispositivos de validação.

Em geral, os modelos tiveram um desempenho melhor ao utilizar os recursos da CPU dos dispositivos móveis durante os testes de desempenho. O desempenho da aplicação ao utilizar a GPU dos dispositivos foi consideravelmente pior para todos os *smartphones*, com problemas de desempenho ocorrendo em nossos testes nos dispositivos com hardware mais limitado, Dispositivo 1 e Dispositivo 2.

O Dispositivo 1, que é um dispositivo de entrada lançado em 2020, foi utilizado para avaliar o desempenho da aplicação em hardware mais antigo. Os modelos YOLOv6-Lite-L (1280x1280), YOLOv6-N6 e YOLOv6-S6 experimentaram ocasionais travamentos ao serem executados na GPU devido a uma baixa taxa de quadros. No uso da CPU apenas o YOLOv6-

Tabela 5 – Média de FPS.

ID do Dispositivo	Modelo	FPS (GPU)	FPS (CPU)
1	YOLOv6-Lite-L (320x320)	6.7	12
1	YOLOv6-Lite-L (640x640)	2.6	4.5
1	YOLOv6-Lite-L (1280x1280)	0.76	1.5
1	YOLOv6-N6	0.58	1.2
1	YOLOv6-S6	0.15	0.6
2	YOLOv6-Lite-L (320x320)	16	30
2	YOLOv6-Lite-L (640x640)	6.5	19.5
2	YOLOv6-Lite-L (1280x1280)	1.6	5
2	YOLOv6-N6	1.4	4
2	YOLOv6-S6	0.8	1.2
3	YOLOv6-Lite-L 320x320	20.3	30.8
3	YOLOv6-Lite-L 640x640	12.3	28.3
3	YOLOv6-Lite-L 1280x1280	2.7	6.6
3	**YOLOv6-N6**	**2.6**	**5.7**
3	YOLOv6-S6	1	2.1

Fonte: elaborada pelo autor.

S6 apresentou problemas de performance, demonstrando lentidão ao realizar a inferência nas imagens.

O Dispositivo 2, lançado em 2023, apresentou bom desempenho com os modelos testados em nossos dispositivos móveis, sem interrupções ou travamentos, mesmo ao utilizar a CPU. Durante os testes com a GPU, pequenos travamentos foram notados ao usar o modelo YOLOv6-S6 devido à baixa taxa de quadros alcançada com esse modelo.

O Dispositivo 3, com as melhores especificações de hardware, não apresentou problemas de desempenho ao executar os modelos na CPU ou GPU do dispositivo.

O modelo que obteve a melhor taxa de quadros por segundo (FPS) foi o YOLOv6-Lite-L (320x320), com uma média de 20,3 FPS na GPU e 30,8 FPS na CPU. Modelos com resoluções mais altas apresentaram médias de FPS mais baixas, sendo a resolução da imagem dos modelos treinados um fator significativo que afeta a complexidade do modelo. Isso fica evidente ao comparar os modelos YOLOv6-Lite-L treinados com diferentes resoluções de imagem.

O melhor modelo para detectar DICs em PCBs é o YOLOv6-N6, que alcançou uma média de 2,6 FPS ao usar a GPU e 5,7 FPS ao usar a CPU além de resultados de mAP e AR comparáveis ao YOLOv6-S6 e custo computacional menor, sendo o modelo ideal para executar em dispositivos de hardware limitado.

Pode ser notado durante a execução dos modelos nos diferentes dispositivos móveis que caso o modelo seja executado a uma taxa de quadros por segundo superior a 1 FPS, a

experiência de uso é satisfatória, sem que seja notado a presença de lentidão ou pequenos travamentos durante a inferência nas imagens captadas pelo dispositivo. Apesar disto, executar os modelos com quantidades de FPS maiores trás maior conforto, sendo notada maior fluides ao capturar as imagens.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresenta uma proposta de um modelo de detecção de objetos projetado para realizar a detecção de Danos Induzidos pelo Cliente (DIC) em Placas de Circuito Impresso (PCB) diretamente em dispositivos Android por meio de um aplicativo móvel. Uma série de experimentos também foram realizados com várias arquiteturas de modelos derivadas do YOLO (YOLOv3-MobileNetV2, YOLOv6, YOLOv7 e YOLOv8) para selecionar o melhor modelo dentro do contexto dos dados rotulados. A arquitetura de rede neural YOLOv6-N6 obteve o melhor resultado com mAP@0.5 de 33,1%, AR de 34,7% e 5,7 de média de FPS ao executar na CPU do melhor dispositivo móvel disponível durante os testes, possuindo resultados de AP e AR muito próximos do modelo mais complexo avaliado (YOLOv6-S6) e custo computacional 3,98 vezes menor.

O conjunto de dados no qual os experimentos foram conduzidos é derivado de centros de reparo, e os dados consistem em imagens de placas de circuito impresso com DICs em várias condições de iluminação, orientações de imagem e posições, o que representou um desafio durante a rotulagem dos dados e o treinamento do modelo.

Na metodologia proposta, este trabalho contribui com três pontos de inovação: foi realizado um estudo sobre detectores de objetos de baixo custo computacional; uma série de experimentos com arquiteturas de detectores de objetos YOLO foram realizados para identificar CIDs em placas de circuito impresso (PCBs) com custo computacional limitado; os modelos treinados foram implementados em dispositivos móveis reais, onde o desempenho na detecção de CIDs em PCBs foi verificado.

Em trabalhos futuros, pode-se destacar a inclusão de novas classes de DICs no conjunto de dados e a rotulação de mais dados para fornecer uma melhor qualidade e diversidade de dados para os modelos treinados. Também pretende-se tornar a aplicação multiplataforma, permitindo que ela seja desenvolvida e validada em dispositivos móveis que utilizam o sistema iOS da Apple.

REFERÊNCIAS

ADIBHATLA, V. A.; CHIH, H.-C.; HSU, C.-C.; CHENG, J.; ABBOD, M. F.; SHIEH, J.-S. Defect detection in printed circuit boards using you-only-look-once convolutional neural networks. **Electronics**, MDPI, v. 9, n. 9, p. 1547, 2020.

ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. In: IEEE. **2017 international conference on engineering and technology (ICET)**. [*S. l.*], 2017. p. 1–6.

ALVES, D.; FARIAS, V.; CHAVES, I.; CHAO, R.; MADEIRO, J. P.; GOMES, J. P.; MACHADO, J. Detecting customer induced damages in motherboards with deep neural networks. In: **2022 International Joint Conference on Neural Networks (IJCNN)**. [*S. l.*: *s. n.*], 2022. p. 1–8.

Andrej Karpathy. **Convolutional Neural Networks (CNNs / ConvNets)**. 2015. Disponível em: https://cs231n.github.io/convolutional-networks/. Acesso em: 14 abr. 2023.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. Yolov4: Optimal speed and accuracy of object detection. **arXiv preprint arXiv:2004.10934**, 2020.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, p. 5–32, 2001.

CHOLLET, F. **Deep learning with Python**. [*S. l.*]: Simon and Schuster, 2021.

GÉRON, A. **Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow**. [*S. l.*]: "O'Reilly Media, Inc.", 2022.

GONZALEZ, R. C.; WOODS, R. E. **Processamento Digital de Imagens**. [*S. l.*]: Pearson, 2009.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S. l.*]: MIT Press, 2016. http://www.deeplearningbook.org.

GOODFELLOW, I.; POUGET-ABADIE, J.; MIRZA, M.; XU, B.; WARDE-FARLEY, D.; OZAIR, S.; COURVILLE, A.; BENGIO, Y. Generative adversarial networks. **Communications of the ACM**, ACM New York, NY, USA, v. 63, n. 11, p. 139–144, 2020.

HAN, S.; MAO, H.; DALLY, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. **arXiv preprint arXiv:1510.00149**, 2015.

HE, F.; TANG, S.; MEHRKANOON, S.; HUANG, X.; YANG, J. A real-time pcb defect detector based on supervised and semi-supervised learning. In: **ESANN**. [*S. l.*: *s. n.*], 2020. p. 527–532.

HE, K.; GKIOXARI, G.; DOLLÁR, P.; GIRSHICK, R. Mask r-cnn. In: **Proceedings of the IEEE international conference on computer vision**. [*S. l.*: *s. n.*], 2017. p. 2961–2969.

HOWARD, A.; SANDLER, M.; CHU, G.; CHEN, L.-C.; CHEN, B.; TAN, M.; WANG, W.; ZHU, Y.; PANG, R.; VASUDEVAN, V. *et al.* Searching for mobilenetv3. In: **Proceedings of the IEEE/CVF international conference on computer vision**. [*S. l.*: *s. n.*], 2019. p. 1314–1324.

HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. **arXiv preprint arXiv:1704.04861**, 2017.

HUI, J. **mAP (mean Average Precision) for object detection**. 2018. Disponível em: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173. Acesso em: 09 jun. 2023.

JOCHER, G.; CHAURASIA, A.; QIU, J. **Ultralytics YOLOv8**. 2023. Disponível em: https://github.com/ultralytics/ultralytics. Acesso em: 15 out. 2023.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Ieee, v. 86, n. 11, p. 2278–2324, 1998.

LI, C.; LI, L.; GENG, Y.; JIANG, H.; CHENG, M.; ZHANG, B.; KE, Z.; XU, X.; CHU, X. Yolov6 v3. 0: A full-scale reloading. **arXiv preprint arXiv:2301.05586**, 2023.

LI, C.; LI, L.; JIANG, H.; WENG, K.; GENG, Y.; LI, L.; KE, Z.; LI, Q.; CHENG, M.; NIE, W. *et al.* Yolov6: A single-stage object detection framework for industrial applications. **arXiv preprint arXiv:2209.02976**, 2022.

LIU, Z.; LIN, Y.; CAO, Y.; HU, H.; WEI, Y.; ZHANG, Z.; LIN, S.; GUO, B. Swin transformer: Hierarchical vision transformer using shifted windows. In: **Proceedings of the IEEE/CVF international conference on computer vision**. [*S. l.*: *s. n.*], 2021. p. 10012–10022.

Muhammad Shoaib Ali. **Flattening CNN layers for Neural Network and basic concepts**. 2022. Disponível em: https://medium.com/@muhammadshoaibali/flattening-cnn-layers-for-neural-network-694a232eda6a. Acesso em: 14 abr. 2023.

O'SHEA, K.; NASH, R. An introduction to convolutional neural networks. **arXiv preprint arXiv:1511.08458**, 2015.

RASCHKA, S.; MIRJALILI, V. **Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2**. [*S. l.*]: Packt Publishing Ltd, 2019.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [*S. l.*: *s. n.*], 2016. p. 779–788.

REDMON, J.; FARHADI, A. Yolo9000: better, faster, stronger. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [*S. l.*: *s. n.*], 2017. p. 7263–7271.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. **arXiv preprint arXiv:1804.02767**, 2018.

REHMAN, A. **Converting a custom dataset from COCO format to YOLO**. 2022. Disponível em: https://medium.com/red-buffer/converting-a-custom-dataset-from-coco-format-to-yolo-format-6d98a4fd43fc. Acesso em: 16 jun. 2023.

RISH, I. *et al.* An empirical study of the naive bayes classifier. In: **IJCAI 2001 workshop on empirical methods in artificial intelligence**. [*S. l.*: *s. n.*], 2001. v. 3, n. 22, p. 41–46.

ROSENBERG, C.; HEBERT, M.; SCHNEIDERMAN, H. Semi-supervised self-training of object detection models. In: **WACV/MOTION**. [*S. l.*: *s. n.*], 2005. p. 29–36.

RUSSELL, S. J. **Artificial intelligence a modern approach**. [*S. l.*]: Pearson Education, Inc., 2010.

SANDLER, M.; HOWARD, A.; ZHU, M.; ZHMOGINOV, A.; CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [*S. l.*: *s. n.*], 2018. p. 4510–4520.

SHAH, D. **Mean Average Precision (mAP) Explained: Everything You Need to Know**. 2022. Disponível em: https://www.v7labs.com/blog/mean-average-precision. Acesso em: 08 jun. 2023.

SHAH, D. **COCO Dataset: All You Need to Know to Get Started**. 2023. Disponível em: https://www.v7labs.com/blog/coco-dataset-guide. Acesso em: 09 jun. 2023.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning**: An introduction. [*S. l.*]: MIT press, 2018.

SZELISKI, R. **Computer vision**: algorithms and applications. [*S. l.*]: Springer Nature, 2022.

TAN, M.; CHEN, B.; PANG, R.; VASUDEVAN, V.; SANDLER, M.; HOWARD, A.; LE, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In: **Proceedings of the IEEE/CVF conference on computer vision and pattern recognition**. [*S. l.*: *s. n.*], 2019. p. 2820–2828.

TAN, M.; PANG, R.; LE, Q. V. Efficientdet: Scalable and efficient object detection. In: **Proceedings of the IEEE/CVF conference on computer vision and pattern recognition**. [*S. l.*: *s. n.*], 2020. p. 10781–10790.

TARVAINEN, A.; VALPOLA, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. **Advances in neural information processing systems**, v. 30, 2017.

WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [*S. l.*: *s. n.*], 2023. p. 7464–7475.

XU, M.; ZHANG, Z.; HU, H.; WANG, J.; WANG, L.; WEI, F.; BAI, X.; LIU, Z. End-to-end semi-supervised object detection with soft teacher. In: **Proceedings of the IEEE/CVF International Conference on Computer Vision**. [*S. l.*: *s. n.*], 2021. p. 3060–3069.

YEGNANARAYANA, B. **Artificial neural networks**. [*S. l.*]: PHI Learning Pvt. Ltd., 2009.

ZHANG, X.; ZHOU, X.; LIN, M.; SUN, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [*S. l.*: *s. n.*], 2018. p. 6848–6856.

ZOPH, B.; VASUDEVAN, V.; SHLENS, J.; LE, Q. V. Learning transferable architectures for scalable image recognition. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [*S. l.*: *s. n.*], 2018. p. 8697–8710.