UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Metodologia e Plataforma Baseadas em Aprendizado de Máquina para Inspeção de Defeitos na Indústria Têxtil

Angelo Leite Medeiros de Góes

Orientador: Prof. Dr. Adrião Duarte Dória Neto

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e de Computação da UFRN (área de concentração: Engenharia de Computação) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Número de Ordem do PPgEEC: M683
Natal, RN, janeiro de 2025

Universidade Federal do Rio Grande do Norte - UFRN
Sistema de Bibliotecas - SISBI
Catalogação de Publicação na Fonte. UFRN - Biblioteca Central Zila Mamede

Goes, Angelo Leite Medeiros de.
Metodologia e plataforma baseadas em aprendizado de máquina para inspeção de defeitos na indústria têxtil / Angelo Leite Medeiros de Goes. - 2025.
152f.: il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Norte, Centro de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica e de Computação, Natal, 2025.
Orientação: Prof. Dr. Adrião Duarte Dória Neto.

1. Visão computacional - Dissertação. 2. Aprendizado profundo - Dissertação. 3. Detecção em grelha - Dissertação. 4. Refinamento em ripple - Dissertação. 5. YOLOv8 - Dissertação. 6. TILDA - Dissertação. I. Dória Neto, Adrião Duarte. II. Título.

RN/UF/BCZM CDU 677

Elaborado por Jackeline dos Santos Pinheiro da Silva Maia Cavalcanti - CRB-CRB-15/317

Metodologia e Plataforma Baseadas em Aprendizado de Máquina para Inspeção de Defeitos na Indústria Têxtil

Angelo Leite Medeiros de Góes

Dissertação de Mestrado aprovada em 27 de janeiro de 2025 pela banca examinadora composta pelos seguintes membros:

Prof. Dr. Adrião Duarte Dória Neto (orientador) . . . . . DCA/UFRN

Prof. Dr. Aluísio Igor Rego Fontes . . . . . IFRN

Prof. Dr. Heitor Medeiros Florêncio . . . . . UFRN

Prof. Dr. Itamir de Morais Barroca Filho . . . . . UFRN

Aos meus pais, Anilda Leite e Wagner Marcks, por todo o carinho, paciência e incentivo ao longo da minha jornada acadêmica. Sem vocês, nada disso seria possível.

Agradecimentos

Ao meu orientador, professor Adrião, minha sincera gratidão pela orientação e incentivo ao longo de tantos anos. Obrigado por acreditar na minha capacidade, mesmo nos momentos em que eu próprio duvidei.

Aos professores avaliadores da banca, pelas valiosas contribuições e críticas construtivas que enriqueceram o texto final desta dissertação.

Aos professores do Departamento de Engenharia de Computação e Automação (DCA), cuja experiência e ensinamentos foram fundamentais para minha formação pessoal e acadêmica.

Aos colegas da pós-graduação, pela ajuda e companhia durante as atividades do curso.

À minha família, pelo apoio incondicional ao longo desta jornada.

À CAPES, pelo apoio financeiro.

"Não se preocupe por ninguém te conhecer; busque ser alguém que valha a pena conhecer".

(Confúcio)

Resumo

A inspeção de qualidade (IQ) na indústria têxtil é um processo essencial, mas desafiador, em especial devido à diversidade de defeitos e à dependência de critérios subjetivos de inspetores humanos. Este trabalho propõe uma metodologia baseada em visão computacional e aprendizado profundo para otimizar o processo de IQ em tecidos, aliada ao desenvolvimento de uma plataforma associada para demonstrar sua viabilidade e aplicação em ambientes industriais. A plataforma integra os algoritmos propostos: a detecção em grelha, que segmenta imagens em *patches* para classificação local de defeitos, e o refinamento em *ripple*, que melhora a qualidade da detecção analisando células vizinhas.

Os resultados experimentais validam a eficácia da metodologia. Um *benchmark* de modelos de aprendizado de máquina foi realizado, comparando oito abordagens, incluindo variantes do YOLOv5, YOLOv8 e redes populares na literatura, utilizando o *dataset* TILDA 400, composto por cinco classes distintas. Os modelos da família YOLOv8 destacaram-se, especialmente o YOLOv8 *medium*, que atingiu 90,35% de acurácia com tempo de inferência de 0,5ms em uma GPU Tesla P100, superando a média de 70% de precisão dos inspetores humanos. O YOLOv8 *small* também apresentou um desempenho notável, com uma capacidade teórica de inspeção de até 46,875 m/min e acurácia de 86,96%, superando a velocidade manual máxima de 15 a 20 m/min.

A aplicação desta metodologia, juntamente com a plataforma desenvolvida, demonstra potencial para aprimorar a precisão, a velocidade e a organização na IQ, além de reduzir os custos operacionais e fomentar a automação e a eficiência no setor têxtil.

Palavras-chave: visão computacional, aprendizado profundo, detecção em grelha, refinamento em *ripple*, YOLOv8, TILDA.

Abstract

Quality inspection (QI) in the textile industry is an essential yet challenging process, particularly due to the diversity of defects and reliance on subjective criteria from human inspectors. This work proposes a methodology based on computer vision and deep learning to optimize the QI process in textiles, alongside the development of an associated platform to demonstrate its feasibility and applicability in industrial environments. The platform integrates the proposed algorithms: grid-based detection, which segments images into patches for local defect classification, and ripple refinement, which improves detection quality by analyzing neighboring cells.

The experimental results validate the effectiveness of the methodology. A benchmark of machine learning models was conducted, comparing eight approaches, including YOLOv5, YOLOv8 variants, and popular networks in the literature, using the TILDA 400 dataset, composed of five distinct defect classes. The YOLOv8 models stood out, especially YOLOv8 *medium*, which achieved 90.35% accuracy with an inference time of 0.5ms on a Tesla P100 GPU, surpassing the average precision of 70% from human inspectors. The YOLOv8 *small* also demonstrated remarkable performance, with a theoretical inspection capacity of up to 46.875 m/min and 86.96% accuracy, exceeding the manual maximum speed of 15 to 20 m/min.

The application of this methodology, combined with the developed platform, demonstrates potential to enhance accuracy, speed, and organization in QI, while also reducing operational costs and promoting automation and efficiency in the textile industry.

Keywords: computer vision, deep learning, grid-based detection, ripple refinement, YOLOv8, TILDA.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

AP: Average Precision

CCD: Charge Coupled Device

CE: Cross-Entropy

CMOS: Complementary Metal-Oxide-Semiconductor

CMYK: Cyan Magenta Yellow Black

CNN: Convolutional Neural Network

COCO: Common Objects in Context

CPS: Cyber Physical Systems

CV: Computer Vision

DNN: Deep Neural Networks

FC: Fully Connected

FPGA: Field-Programmable Gate Arrays

FPS: Frames Per Second

GPU: Graphics Processing Unit

HIS: Hybrid Inspection Systems

IoT: Internet of Things

IoU: Intersection Over Union

mAP: mean Average Precision

ML: Machine Learning

MLP: Multi Layer Perceptron

MSCNN: Multi-Scale Convolutional Neural Network

MSE: Mean Squared Error

NMS: Non-Maximum Suppression

PoC: Proof of Concept

PR: Precision Recall

PTIT: Patches Training and Image Testing

RA: Reference Architecture

RCNN: Region Based Convolutional Neural Networks

ReLU: Rectified Linear Unit

RGB: Red Green Blue

TILDA: Textile Texture Database

YOLO: You Only Look Once

Capítulo 1

Introdução

A Indústria 4.0, também chamada de Quarta Revolução Industrial, representa uma transformação paradigmática na manufatura global, impulsionada pela integração de tecnologias avançadas nos processos produtivos. Essa nova era industrial caracteriza-se pela implementação de dispositivos inteligentes que podem se comunicar de forma autônoma ao longo da cadeia de valor, promovendo maior eficiência, flexibilidade e inovação (Santos et al. 2018).

Nesse contexto, os Sistemas Ciber-Físicos (*Cyber-Physical Systems*, CPS) desempenham um papel central, ao permitir a auto-organização e o monitoramento contínuo dos processos industriais, criando uma cópia virtual do mundo real. A Internet das Coisas (*Internet of Things*, IoT), por sua vez, conecta máquinas, objetos e pessoas em tempo real, viabilizando um fluxo contínuo de informações. Além disso, a Computação em Nuvem (*Cloud Computing*) possibilita o armazenamento, a troca e a gestão eficiente desses dados, permitindo a combinação de processos produtivos e de negócios para a geração de valor (Santos et al. 2018).

Entre os setores que podem se beneficiar amplamente dessas inovações, destaca-se a indústria têxtil, um dos pilares da manufatura global. Em particular, a inspeção de qualidade (IQ) pode ser aprimorada significativamente com o uso dessas tecnologias, possibilitando um controle mais preciso e eficiente da produção, reduzindo desperdícios e aumentando a competitividade no mercado.

A IQ desempenha um papel crítico na manufatura, assegurando a integridade dos produtos e a confiança dos clientes. Historicamente, o equilíbrio entre os papéis humanos e das máquinas na IQ tem sido fundamental. Pesquisas iniciais conduzidas por Drury & Sinclair (1983), que utilizavam a variação de luz refletida em uma matriz de fotodiodos para detecção de defeitos, destacaram a importância desse equilíbrio, revelando que, embora as máquinas sejam superiores na localização de defeitos com rapidez e precisão, os humanos se destacam no processo de tomada de decisão, classificando defeitos com maior exatidão. Essa sinergia deu origem aos Sistemas de Inspeção Híbridos (*Hybrid Inspection Systems*, HIS), que combinam as vantagens de ambos para resultados mais eficazes.

Embora a Indústria 4.0 tenha promovido avanços significativos na automação e adaptabilidade dos processos produtivos, por meio da integração de CPS, IoT e *Cloud Computing*, a IQ na indústria têxtil ainda enfrenta desafios complexos. A variabilidade e a complexidade dos defeitos nos tecidos dificultam a detecção e classificação automatiza-

das, tornando a intervenção humana ainda essencial. No entanto, essa dependência pode resultar em inconsistências, vieses e ineficiências, comprometendo a padronização e a eficácia do controle de qualidade.

Nessa perspectiva, esta dissertação propõe uma metodologia fundamentada em visão computacional e aprendizado profundo para otimizar o processo de IQ na indústria têxtil. Essa abordagem busca mitigar as limitações atuais, reduzindo a dependência de operadores humanos e aumentando a precisão e a velocidade na detecção de defeitos. Para demonstrar a viabilidade dessa metodologia, será desenvolvida uma prova de conceito (*Proof of Concept*, PoC) de uma plataforma de IQ, integrando os conceitos propostos e demonstrando sua aplicabilidade em ambientes industriais.

Espera-se que essa metodologia contribua não apenas para a redução de custos operacionais, mas também para o aprimoramento contínuo da qualidade do produto final, consolidando o papel da automação avançada como uma ferramenta estratégica para a evolução do setor têxtil.

1.1 Contextualização

A indústria têxtil tem um papel estratégico na economia global e, no Brasil, sua relevância é ainda mais evidente. Com quase 200 anos de história, o setor brasileiro abriga a maior cadeia produtiva completa do Ocidente, cobrindo todas as etapas, desde a produção de fibras até a confecção e comercialização de produtos finais, segundo a Associação Brasileira da Indústria Têxtil e de Confecção (ABIT)[1]. Além disso, o país se destaca como referência mundial em segmentos como moda praia, *jeanswear*, moda *fitness* e *lingerie*.

O impacto econômico dessa indústria é significativo. De acordo com a ABIT, em 2023, o setor registrou um faturamento de R$ 203,9 bilhões, com investimentos de R$ 4,6 bilhões. No que diz respeito à empregabilidade, cerca de 8 milhões de pessoas trabalham direta ou indiretamente na indústria têxtil, considerando efeito renda, sendo 60% da mão de obra composta por mulheres. Com 25,3 mil unidades produtivas formais, o setor segue como um dos alicerces da economia nacional.

Na Subseção 1.1.1, serão abordados, de forma mais didática, o funcionamento e os principais processos que compõem este setor econômico. Já na Subseção 1.1.2, será dada ênfase ao processo de IQ, elemento central desta dissertação, através uma análise mais técnica da situação atual, acompanhada de uma discussão sobre a adoção de abordagens mais modernas ao processo.

1.1.1 Processos Produtivos de um Complexo Têxtil

O setor têxtil abrange uma ampla gama de indústrias, muitas vezes categorizadas como indústrias tradicionais, ligadas diretamente à produção de tecidos, roupas e acessórios. Essas indústrias possuem características marcantes, como o uso intensivo de mão de obra pouco qualificada, a transformação mínima da matéria-prima no processo produtivo, baixa inovação nos métodos de fabricação, poucas barreiras à entrada de novas

[1] `https://www.abit.org.br/cont/perfil-do-setor`

empresas e a concorrência de pequenas e microempresas com grandes indústrias (de Oliveira & de Lima 2017).

A Figura 1.1 ilustra a cadeia produtiva têxtil, que pode ser resumida nos seguintes elementos básicos sequenciais (Pereira 2009).

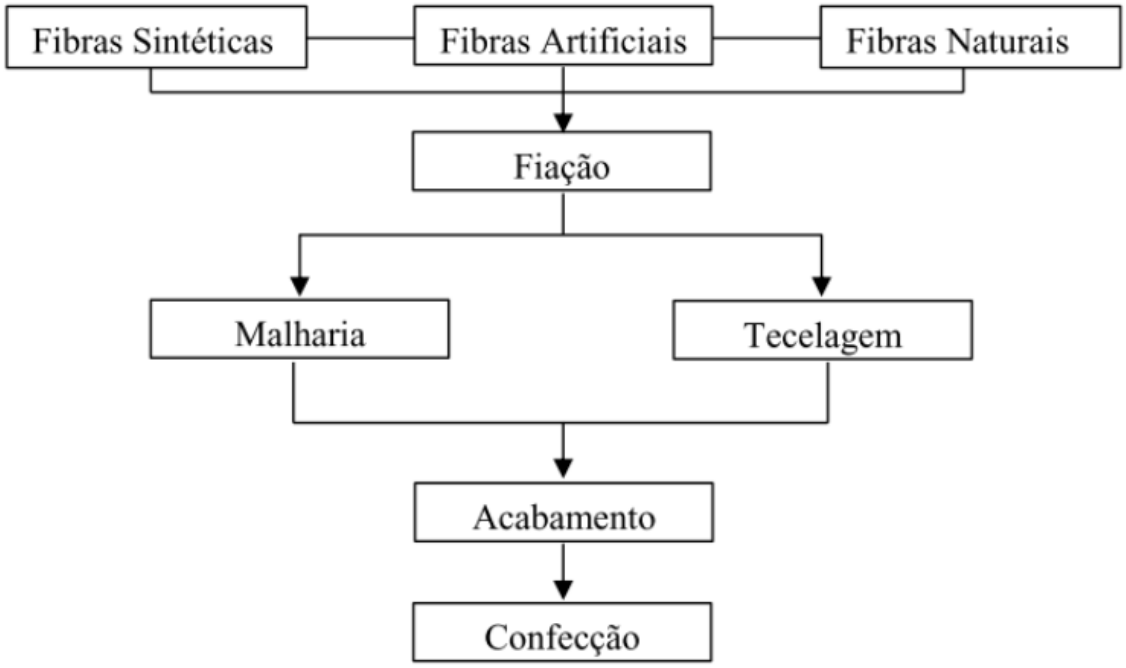

Figura 1.1: A Cadeia de Produção Têxtil (Pereira 2009)

Como é possível observar, a cadeia abrange os seguintes processos/segmentos principais:

- **Produção de fibras:** envolve fibras naturais e químicas. As fibras químicas são obtidas por processos químicos, sendo apresentadas como filamentos contínuos ou cortados;
- **Fiação:** etapa em que as fibras, naturais ou químicas, são transformadas em fios;
- **Tecelagem plana:** os fios são entrelaçados para formar tecidos planos, também conhecidos como tecidos convencionais;
- **Malharia:** os fios são transformados em tecidos de malha por meio de laçadas entrelaçadas, diferenciando-se dos tecidos planos pela forma de entrelaçamento;
- **Acabamento ou Beneficiamento:** inclui processos como alvejamento, tinturaria e estamparia, que conferem aos produtos características visuais, táteis e de cor, de acordo com a demanda.
- **Confecção:** o produto têxtil é repassado ou vendido para clientes nos diversos segmentos de confecção, como vestuário, artigos domésticos, roupas hospitalares, produtos industriais, entre outros (Pereira 2009).

Cada um desses segmentos pode oferecer um produto acabado ao mercado e, na prática, funcionar de maneira independente dos demais. Nem todas as etapas precisam ser

internalizadas pelas empresas, o que permite a coexistência de organizações especializadas com diferentes níveis de atualização tecnológica (Pereira 2009).

O resultado de cada etapa de produção pode alimentar a próxima, independentemente de fatores como escala ou tecnologia. Por exemplo, algumas indústrias atuam exclusivamente no subsetor de fiação e fornecem insumos para outras especializadas em malharia ou tecelagem plana. Outras, por sua vez, são totalmente verticalizadas, integrando todos os subsetores têxteis para atender a indústrias de confecção e vestuário (Pereira 2009).

O processo de IQ abordado nesta dissertação ocorre logo após as etapas de tecelagem e malharia, no momento em que o tecido está sendo formado. No entanto, caso o material passe por tratamentos adicionais ou modificações, a inspeção também pode ser realizada após as etapas de beneficiamento ou confecção.

Segundo Schneider et al. (2020), após a confecção do tecido base, o processo de transformação em produtos finais envolve diversas etapas. O processo começa com o **planejamento do produto**, em que são definidos aspectos como tamanho, cor e estampa em parceria com o cliente. Em seguida, o **corte** dos tecidos é feito conforme moldes baseados nas medidas fornecidas, enquanto a **estamparia** aplica as estampas utilizando métodos como *screen printing* ou *roller printing*. A **costura**, que pode ser realizada manualmente ou terceirizada, a depender do porte da empresa. O **acabamento** acrescenta detalhes como botões e barras, seguido pela revisão para identificação de falhas. Após isso, a **embalagem** prepara e protege os produtos, e, por fim, a **separação** no estoque organiza as embalagens para facilitar a retirada ou entrega dos itens.

Dessa forma, desde o recebimento da matéria-prima até a entrega ao consumidor final, o complexo têxtil abrange diversos processos interdependentes. Observa-se, contudo, que muitas dessas etapas ainda são realizadas manualmente, o que limita a eficiência da cadeia como um todo. Essa realidade evidencia um significativo potencial de otimização por meio da implementação de metodologias inovadoras de automação e digitalização de sistemas (de Oliveira & de Lima 2017).

1.1.2 Inspeção de Defeitos em Tecidos

Apesar de sua solidez, a indústria têxtil enfrenta um desafio persistente: defeitos em tecidos. Esses defeitos correspondem a aproximadamente 85% dos problemas observados no segmento de vestuário, resultando em perdas significativas de lucro, uma vez que os fabricantes conseguem recuperar apenas entre 45% e 65% de sua receita potencial proveniente de produtos defeituosos (Sengottuvelan et al. 2008).

A detecção e a classificação de defeitos em tecidos são intrinsecamente desafiadoras devido aos fundos complexos e às escalas irregulares dos defeitos. Os métodos tradicionais de inspeção manual, embora prevalentes, dependem fortemente da experiência dos inspetores e sofrem com altas taxas de erro, detectando apenas cerca de 70% dos defeitos (Sengottuvelan et al. 2008). Além disso, a natureza de alta velocidade da produção têxtil moderna, que requer uma inspeção de 15 a 20 metros de tecido por minuto (m/min), intensifica ainda mais as limitações da inspeção manual. A Figura 1.2 apresenta um cenário comum de inspeção manual na indústria têxtil.

Os avanços tecnológicos, especialmente na área de inteligência artificial e suas di-

Figura 1.2: Detecção manual de defeitos em tecidos (Zhao et al. 2020)

versas técnicas, como o aprendizado profundo, trazem soluções promissoras para esses desafios. Embora métodos tradicionais de visão computacional tenham buscado aprimorar a detecção de defeitos, muitas vezes eles falham em alcançar níveis satisfatórios de precisão e capacidade de generalização. O aprendizado profundo, no entanto, apresenta um potencial significativo para aprimorar a detecção e classificação de defeitos, oferecendo maior velocidade e precisão, mesmo em cenários de aplicação em tempo real.

Apesar dos avanços, a implementação de um sistema eficaz de detecção automatizada de defeitos continua desafiadora. Os defeitos em tecidos são variados e multiescalares, demandando modelos de detecção que sejam eficientes e adaptáveis a diferentes tipos de tecidos e defeitos. Além disso, é crítico encontrar um equilíbrio entre velocidade computacional e precisão, especialmente considerando as limitações de recursos computacionais em ambientes industriais.

1.2 Objetivos e Metodologia

O principal objetivo desta dissertação é propor uma metodologia fundamentada em visão computacional e modelos de aprendizado profundo para otimização do processo de IQ na indústria têxtil. Atrelada a esta proposta, será desenvolvida a PoC de uma plataforma de IQ, com o objetivo de demonstrar a viabilidade da aplicação dessa metodologia.

Em relação aos objetivos específicos, destacam-se os seguintes:

1. **Investigar Técnicas de Aprendizado Profundo Estado da Arte para Detecção em Imagens**: Investigar funcionamento e aplicação de redes neurais convolucionais (CNNs), como a YOLOv8, para a identificação e classificação de objetos em imagens, em especial, de defeitos em tecidos.

2. **Elaborar Algoritmos de Detecção e Refinamento**: Desenvolver e fundamentar algoritmos especializados para sistemas de IQ voltados à detecção de defeitos em tecidos, utilizando CNNs para classificação e inferência de *patches* para localização.
3. **Documentar Processo de Preparação de *Dataset* para Treinamento dos Modelos**: Detalhar o processo de coleta, rotulação e estruturação do *dataset* voltado para a detecção de defeitos têxteis, utilizado no treinamento dos modelos investigados.
4. **Análisar e Comparar Modelos de Detecção de Defeitos**: Comparar modelos e técnicas propostas com outras metodologias existentes por meio de *benchmarks*.
5. **Desenvolver uma Plataforma de IQ**: Documentar e desenvolver uma plataforma completa voltada para IQ, capaz de processar imagens de tecidos em tempo real, identificar e reportar defeitos, além de oferecer funcionalidades como treinamento de modelos, criação de *datasets* e rastreamento de experimentos.

Para alcançar os objetivos práticos propostos, este trabalho está organizado em etapas bem definidas. A partir do Capítulo 4 sobre a metodologia usada, serão descritas todas as ferramentas de *software* relevantes para a implementação lógica do projeto e da PoC da aplicação.

Na sequência, serão desenvolvidos algoritmos específicos para a detecção de defeitos em tecidos. Esses algoritmos utilizarão CNNs, com o *backbone* YOLOv8 sendo empregado para a extração de características e classificação de *patches* — pequenos segmentos das imagens de tecido — com o objetivo de localizar defeitos de maneira eficiente. Os defeitos serão classificados em cinco categorias principais: sem defeito (*good*), buraco (*hole*), objetos (*object*), mancha de óleo (*oil stain*) e erro de fio (*thread error*).

Para a etapa inicial de detecção, será implementado um algoritmo baseado em grade, configurado com um limiar alto de confiança para identificar regiões de interesse primárias. Posteriormente, um limiar mais baixo será aplicado para refinar as detecções em células adjacentes, utilizando um método proposto denominado refinamento em *ripple*.

Em sequência, será apresentado o processo de modelagem da Plataforma de Inspeção de Qualidade em Tecidos, tomando como base uma arquitetura de referência simplificada para aplicações IoT na indústria têxtil. Essa arquitetura não é formalizada com o rigor técnico de um modelo industrial, servindo apenas como um ponto de partida para a modelagem da arquitetura de *software* concreta. A plataforma tem como objetivo processar imagens para identificar defeitos em tecidos, sendo validada por meio de arquivos de vídeo locais, simulando um cenário real de inspeção em uma máquina revisadeira. As informações extraídas permitem detalhar a localização e categorização dos defeitos, possibilitando sua utilização para notificação da equipe de produção conforme a gravidade dos problemas identificados.

O sistema de processamento será dinâmico, capaz de lidar com fluxos de vídeo em tempo real e apresentar os resultados simultaneamente. A implementação integrará diversos componentes de *software*, incluindo:

- Um *backend* para gerenciamento geral das sessões de operação;
- Outro para gerenciamento, armazenamento e aumento dos dados de treinamento;
- Um terceiro dedicado ao treinamento e à interface de inferência dos modelos.

Esses componentes serão desenvolvidos utilizando o *framework* Flask, em Python. O gerenciamento dos modelos será centralizado na ferramenta MLFlow, enquanto o *frontend*, construído em Angular, será responsável por fornecer uma interface amigável para demonstração e operação do sistema.

Por fim, modelos serão treinados utilizando dados publicamente disponíveis e comparados com métodos consolidados na literatura. O desempenho será avaliado com base em métricas de acurácia, velocidade de inferência e eficiência operacional, expressa em metros de tecido inspecionados por minuto (m/min). Adicionalmente, serão apresentadas as principais interfaces da aplicação desenvolvida. Tanto a implementação quanto os dados utilizados foram disponibilizados nas plataformas GitHub[2] e Kaggle[3], visando promover a reprodutibilidade dos experimentos e fomentar futuras pesquisas na área.

1.3 Estruturação dos Capítulos

Este trabalho está organizado de forma a abordar os principais aspectos teóricos, metodológicos e experimentais relacionados à dissertação. A estruturação dos capítulos é descrita a seguir:

O Capítulo 2 fornece o embasamento teórico necessário para compreender a metodologia proposta, a modelagem da plataforma desenvolvida e os resultados obtidos nos experimentos. Este capítulo abrange tópicos fundamentais em visão computacional e inteligência artificial, com ênfase em aprendizado profundo, CNNs e modelos de detecção, incluindo o modelo YOLO. Além disso, são discutidos tópicos essenciais da engenharia de *software*, como arquitetura de referência, engenharia de requisitos e diagramas UML, oferecendo o contexto necessário para a fundamentação e desenvolvimento da solução proposta.

No Capítulo 3, são revisados os trabalhos relacionados à dissertação, tanto no domínio de inteligência artificial, quanto no domínio de engenharia de *software*. A seção inicia com a descrição do procedimento de seleção dos estudos relevantes e prossegue com a análise dessas pesquisas prévias, ressaltando seus pontos fortes e as limitações dos métodos existentes.

O Capítulo 4 descreve a metodologia adotada para a obtenção dos resultados, abordando a utilização das ferramentas empregadas, a formalização dos algoritmos propostos e o detalhamento do projeto da arquitetura de referência. Também é apresentada a modelagem da PoC do *software* concreto, desde a definição dos requisitos até a criação dos diagramas.

No Capítulo 5, são detalhados os experimentos realizados, incluindo a criação e preparação do *dataset* e os resultados obtidos, seguidos da análise e interpretação desses resultados. O capítulo também apresenta as principais telas da plataforma implementada, ilustrando seu funcionamento e destacando as funcionalidades-chave.

Por fim, o Capítulo 6 encerra o trabalho revisitando as principais contribuições da dissertação, discutindo as limitações encontradas e sugerindo possíveis direções para pes-

[2] https://github.com/angelolmg/textile-defect-detection-app

[3] https://www.kaggle.com/datasets/angelolmg/tilda-400-64x64-patches

quisas futuras. Este capítulo oferece uma reflexão sobre os resultados alcançados e suas implicações para a indústria têxtil, encerrando com uma menção às produções acadêmicas realizadas ao longo do programa.

Adicionalmente, as informações adicionais apresentadas no Apêndice A complementam a discussão, fornecendo detalhes sobre a especificação da arquitetura de referência, especificações de caso de uso da aplicação, artefatos de treino e validação, e outras informações relevantes ao projeto.

Capítulo 2

Fundamentação Teórica

A seguir, será apresentada a fundamentação teórica do trabalho. Inicia-se com uma introdução à visão computacional, abordando ideias básicas e aplicações. Em seguida, será discutida a aprendizagem profunda, ou *deep learning*, incluindo sua definição geral, métodos, treinamento e conceitos fundamentais. Posteriormente, serão abordadas as CNNs, detalhando suas camadas e algumas arquiteturas clássicas. Por fim, será analisado o modelo e o algoritmo YOLO, abordando aspectos relevantes ao trabalho, com uma explicação sobre seu funcionamento, a estrutura da rede neural e seus principais diferenciais.

2.1 Visão Computacional e IA

Visão computacional (*Computer Vision*, CV) é um campo da computação que busca replicar a complexidade da visão humana, permitindo que computadores capturem, interpretem e processem informações visuais de forma semelhante aos humanos[1]. Em termos de engenharia, isso envolve o estudo de como a visão e a interpretação dos estímulos visuais funcionam e como é possível usar máquinas para replicar e automatizar esse processo natural.

O processo de visão computacional começa com a aquisição de dados visuais por meio de dispositivos como câmeras, seguidos por etapas de processamento e interpretação para extrair informações úteis. Um dos desafios centrais é a conversão desses dados visuais em representações numéricas que possam ser processadas por algoritmos computacionais. Para isso, uma representação comum é a matriz de *pixels*, onde cada *pixel* corresponde a um ponto da imagem com valores de intensidade de luz variando de 0 a 255, como mostrado na Figura 2.1 (a).

Nos últimos anos, os avanços em inteligência artificial, particularmente com CNNs, têm impulsionado o desenvolvimento da visão computacional, superando até o desempenho humano em várias tarefas. Estes avanços têm sido possíveis devido ao aumento do poder computacional e à crescente disponibilidade de dados, os quais são fundamentais

[1] https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e

[3] http://docs.loopbio.com/motif/image-quality/digital-video-data/

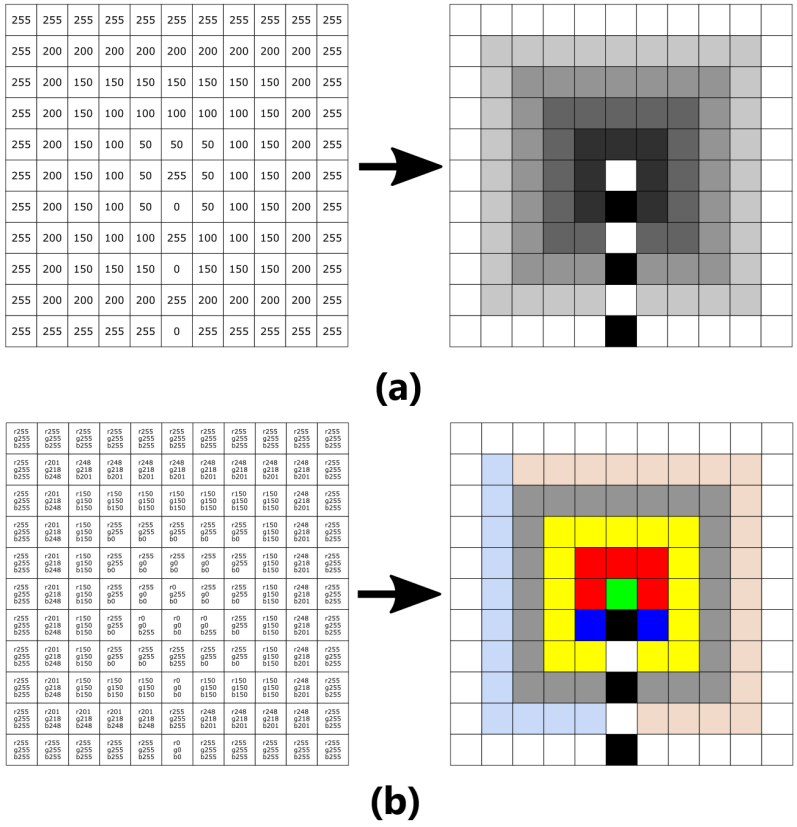

Figura 2.1: (a) Matriz de imagem em preto e branco e (b) Matriz de imagem colorida (RGB) [3]

para o treinamento dos modelos de aprendizado profundo. Estima-se que a quantidade de dados gerados globalmente alcance 180 zettabytes até o final de 2025[4].

O crescimento da CV também se deve à combinação de diversas áreas de conhecimento, como geometria, física e estatística, que são aplicadas na extração de características visuais de imagens. Ferramentas específicas, como a biblioteca *OpenCV*, desempenham papel importante no processamento de imagens e na implementação de algoritmos que transformam dados visuais em informações úteis.

As aplicações de visão computacional são amplas e incluem desde a identificação de objetos e controle de qualidade na indústria, até navegação autônoma, detecção de eventos, modelagem tridimensional e interação humano-computador. Em ambientes industriais, a CV é frequentemente usada para inspeção automatizada de defeitos e controle de processos em tempo real.

Sistemas de visão computacional geralmente incluem um dispositivo de aquisição de imagens, como câmeras digitais ou sensores CCD (*Charge-Coupled Device*), um processador para interpretar os dados visuais, uma fonte de alimentação e um mecanismo de comunicação (e.g., *Ethernet, Wi-Fi*). Em versões mais sofisticadas, esses sistemas podem incorporar fontes de iluminação especializadas para otimizar a captura de imagens e interfaces de monitoramento, como estações de trabalho e monitores. Além disso, tecnologias como câmeras termográficas, ressonância magnética, *scanners* LIDAR, e *scanners* hiperespectrais odem ser incorporados para capturar informações fora do espectro da luz visível. Esses sistemas ampliam as capacidades da CV, permitindo aplicações em áreas como segurança, saúde e automação.

2.1.1 Aprendizagem Profunda

Aprendizagem profunda, ou *deep learning*, é um subconjunto de *machine learning* que se concentra em métodos de aprendizado de máquina por meio da extração de padrões em dados de entrada, conhecidos como dados de treinamento, rotulados ou não. Esses métodos são utilizados para fazer previsões ou tomar decisões sobre novos dados nunca antes vistos. Nesse contexto, os algoritmos não são explicitamente programados, mas "se especializam" em uma determinada tarefa por meio do processamento de grandes volumes de dados de entrada e da minimização do erro entre a resposta obtida e a esperada.

Deep learning é a tecnologia responsável por diversas inovações contemporâneas, como carros autônomos e controle por voz, e permeia várias áreas, incluindo bioinformática, *design* de medicamentos, análise de imagens médicas, análise climática e jogos de tabuleiro. Em alguns casos, essas redes neurais obtêm resultados comparáveis ou até superiores aos de especialistas humanos.

O termo "profunda" (*deep*) refere-se ao uso de múltiplas camadas na rede, permitindo a extração de características de níveis progressivamente mais complexos e a realização de tarefas de classificação diretamente com imagens, sons, texto ou qualquer conjunto de dados representáveis visualmente. Por exemplo, a Figura 2.2 ilustra como os mapas de *features* em uma CNN tornam-se progressivamente mais ricos semanticamente, embora

[4] https://www.statista.com/statistics/871513/worldwide-data-created/

mais intricados, à medida que sua dimensionalidade diminui. Aprimorando, neste caso da figura, a capacidade do modelo em representar e identificar expressões faciais.

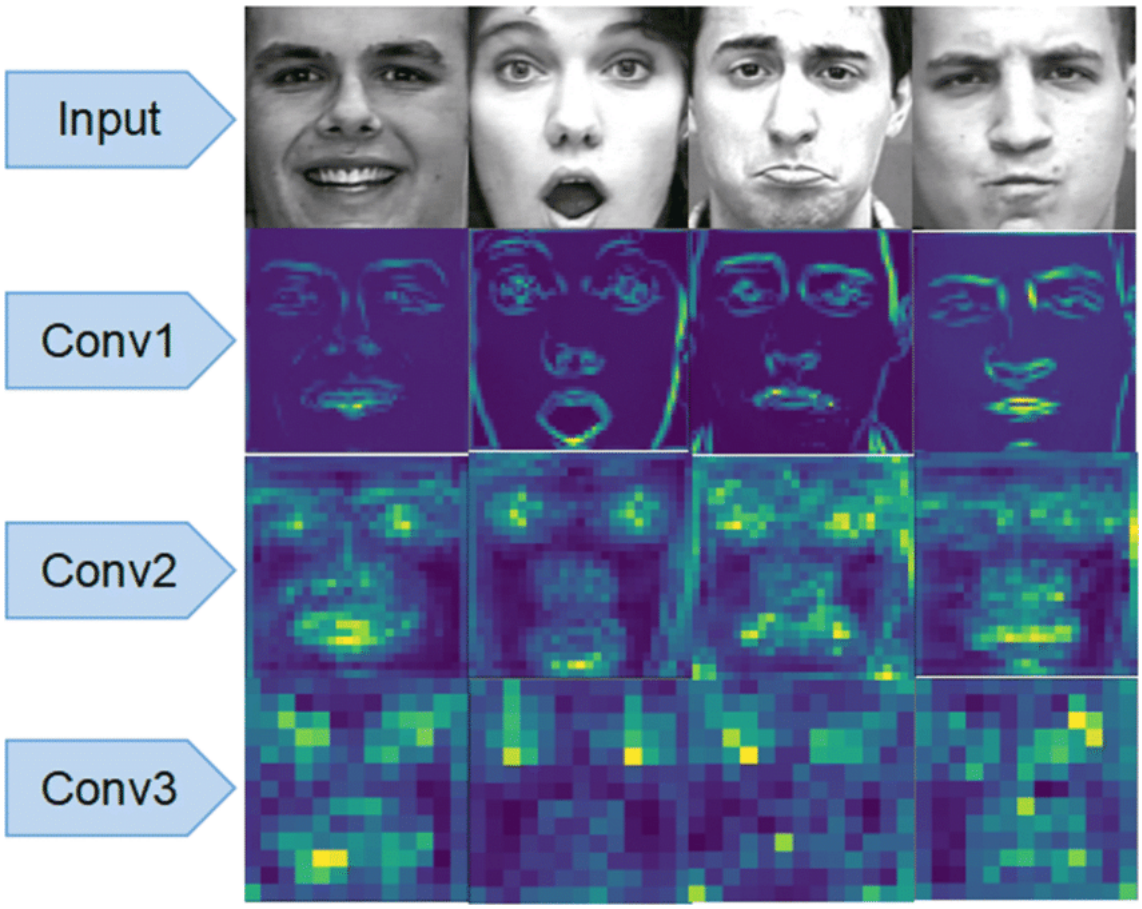

Figura 2.2: Visualização de diferentes níveis de *features* convolucionais extraídas por um CNN (Sun & Lv 2019)

Os métodos de *deep learning* utilizam arquiteturas de redes neurais. Redes tradicionais, como *perceptrons* de múltiplas camadas (*Multi Layer Perceptron*, MLP), geralmente possuem de 1 a 3 camadas, enquanto redes profundas (*Deep Neural Networks*, DNN) podem chegar a milhares de camadas e bilhões de parâmetros (Wang, Ma, Dong, Huang, Zhang & Wei 2022). Cada neurônio (ou nó) na camada oculta e de saída possui uma função de ativação, que padroniza a saída dentro de um determinado intervalo e determina se aquele neurônio está ativado ou não. A ativação de um neurônio determina sua importância para a previsão final e dá um aspecto de não-linearidade à operação. A Figura 2.3 representa a lógica de um único nó, onde as entradas x_i são ponderadas e somadas pelos pesos (*weights*) ω_i, adicionadas a um viés (*bias*) ω_0 e o resultado z é passado por uma função de ativação para obter-se o resultado final daquele nó específico.

O cálculo se repete para todos os nós ao longo de uma rede, propagando o resultado para frente até a camada de saída, onde se encontra o resultado final. Esse processo é conhecido como *feedforward*, sendo a primeira etapa do treinamento de uma DNN. Os

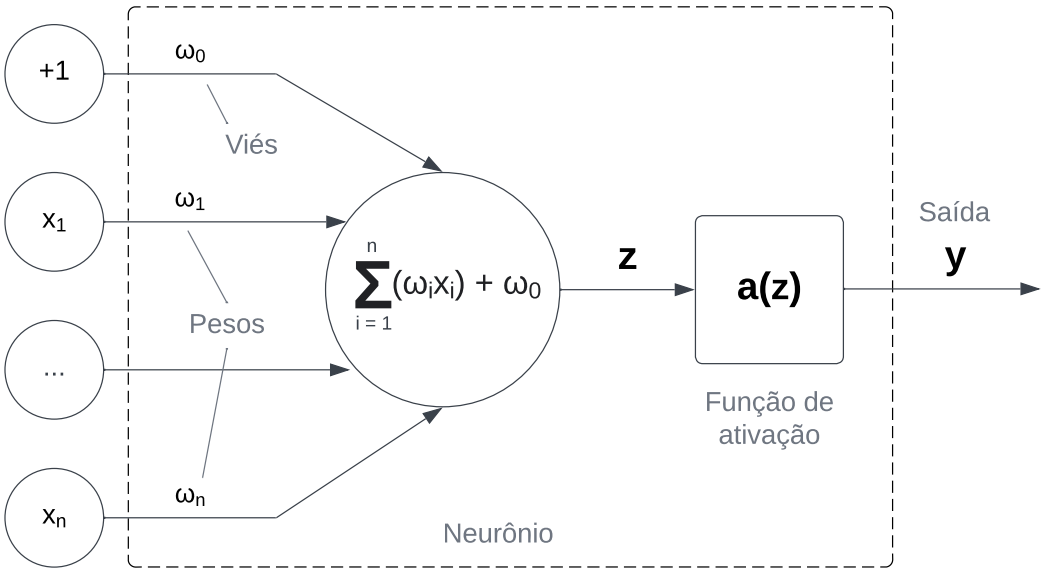

Figura 2.3: Modelo de neurônio de McCulloch-Pitts

pesos indicam a força da conexão entre um nó e a saída, enquanto o viés é um nó com valor sempre igual a +1, usado para "deslocar" a função de ativação, aumentando o grau de liberdade para ajustar melhor a saída aos dados de entrada.

O treinamento supervisionado e o aprendizado ocorrem utilizando o erro entre a saída e o resultado esperado para ajustar os pesos e vieses ao longo de múltiplas iterações (um *batch*) ou até épocas (completando todos os *batches*, ou seja, a base de dados completa). Esse processo é conhecido como *backpropagation*. A Figura 2.4 ilustra um *perceptron* de múltiplas camadas, onde os nós rosas representam o vetor de entrada $x_i = a_i^{(1)}$, $i \in \{1,2,3,4\}$, indicando o vetor de ativações na camada 1 (*input*). Nos nós verdes (a), os valores são computados pela soma ponderada dos valores de entrada (z) passados por uma função de ativação (f):

$$z^{(2)} = \begin{bmatrix} W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + W_{14}^{(1)}x_4 \\ W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + W_{24}^{(1)}x_4 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \tag{2.1}$$

$$\begin{aligned} z^{(2)} &= W^{(1)}x + b^{(1)} \quad & a^{(2)} &= f(z^{(2)}) \\ z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \quad & a^{(3)} &= f(z^{(3)}) \\ s &= f(W^{(3)}a^{(3)} + b^{(3)}) \end{aligned} \tag{2.2}$$

Sendo $W^{(i)}$ a matriz de pesos na camada i e $b^{(i)}$ o vetor de pesos na camada i, $a^{(i)}$ é computado usando f, uma função de ativação. Diferentes funções de ativação podem ser usadas a depender da aplicação, no entanto as mais comuns são a *sigmoid*/logística, *tanh*, ReLU e *Leaky* ReLU, como ilustrado na Figura 2.5. A característica mais importante

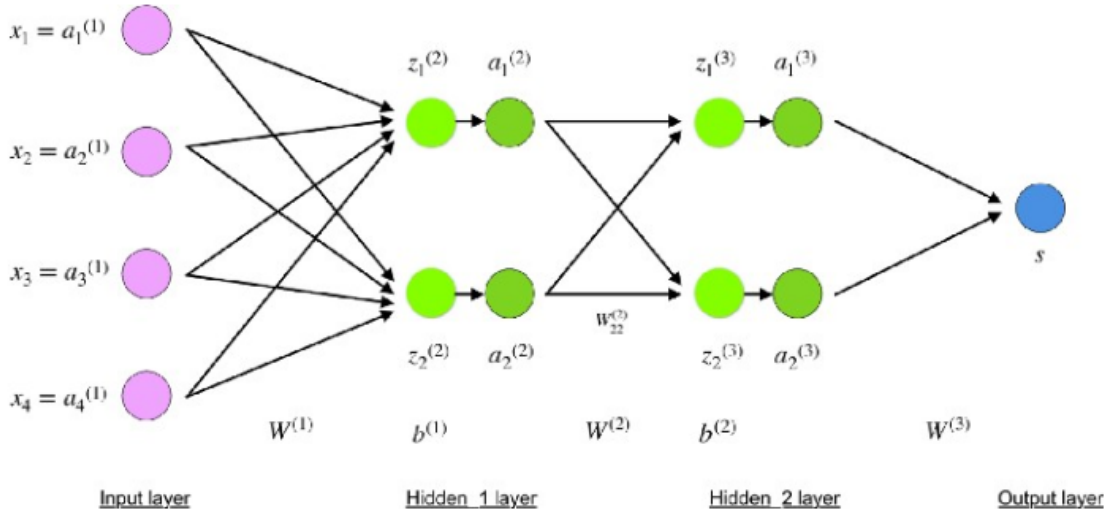

Figura 2.4: Estrutura genérica de um perceptron de múltiplas camadas

dessas funções é sua natureza não-linear, permitindo à rede aprender padrões complexos nos dados.

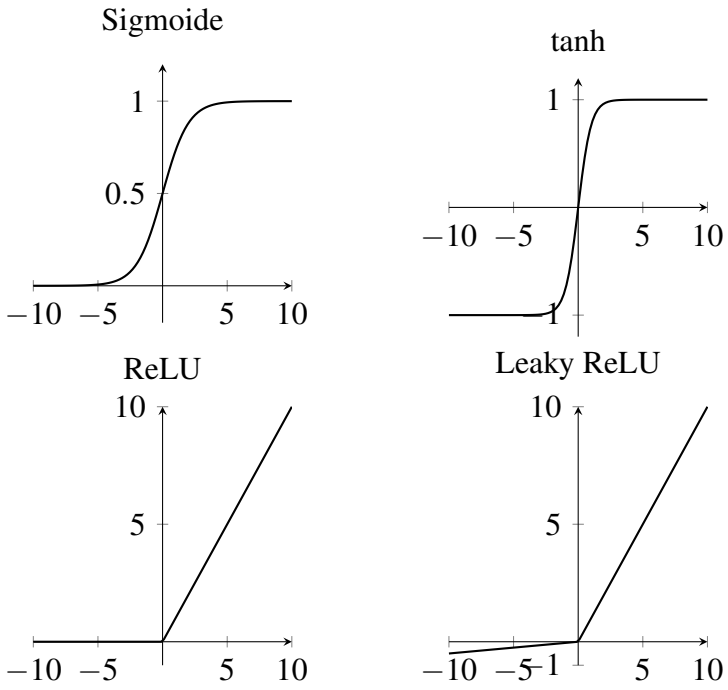

Figura 2.5: Curvas Sigmoide, Tanh, ReLU e *Leaky* ReLU

Para tarefas de classificação, onde cada nó de saída representa o nível de confiança em uma classe específica, utiliza-se a função de ativação *softmax*, que converte um vetor de K números de entrada em um vetor de K probabilidades, onde a soma dos valores é 1.

A função *softmax* é definida por:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.3}$$

Após o processo de *feedforward*, é preciso avaliar a resposta dada em relação à esperada, utilizando uma função de custo. Uma opção de função custo é o erro quadrático médio (*Mean Squared Error*, MSE), definido por:

$$MSE(y_i, \hat{y}_i) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.4}$$

Outra opção mais complexa é a entropia cruzada (*Cross Entropy*, CE), dada por:

$$CE(y_i, \hat{y}_i) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i), \tag{2.5}$$

onde y_i é um valor binário que indica se a classe i é a verdadeira classe (1 para a classe correta, 0 para as demais), e $\hat{y}_i$ é a probabilidade prevista para a classe i.

Ao calcular o custo, o modelo ajusta os parâmetros para aproximar o *output* dos valores corretos. O objetivo do algoritmo de *backpropagation* é minimizar a função de custo ajustando os valores de pesos e vieses, utilizando o cálculo do gradiente do custo, que indica o quanto alterar cada parâmetro para minimizá-la, processo conhecido como "descida gradiente". A Equação 2.6 representa a derivada da curva de perda relativa a ω_{ij}, usada na redução do erro durante o treinamento.

$$\Delta\omega_{ij} = -\eta\frac{\partial E}{\partial \omega_{ij}} = -\eta o_i \delta_i \tag{2.6}$$

Sendo $\Delta\omega_{ij}$ a variação necessária para atualizar o parâmetro ω_{ij}, E a função de erro que quantifica a discrepância entre a saída prevista e o valor esperado, e η a taxa de aprendizado que controla a magnitude dos ajustes durante o treinamento. O objetivo é se aproximar do mínimo da função de erro em pequenos passos para evitar *overshooting*, para tal η geralmente é definido com um valor pequeno, como 0,01.

A equação expressa que a alteração no peso $\Delta\omega_{ij}$ é igual a $-\eta\frac{\partial E}{\partial \omega_{ij}}$, onde $\frac{\partial E}{\partial \omega_{ij}}$ é a derivada da função de erro em relação ao peso, indicando a direção e a intensidade da mudança necessária para minimizar o erro.

Podemos observar que esta variação é também expressa como $-\eta o_i \delta_i$, em que o_i representa a saída do neurônio i e δ_i é o erro retropropagado. Essa última forma elabora a ideia de que a atualização dos pesos é influenciada tanto pela saída do neurônio quanto pelo erro que ele gera.

Além da taxa de aprendizado, existem outros hiperparâmetros que devem ser definidos, entre eles o *batch size*, que determina o tamanho dos lotes de treino. O *batch size* influencia a descida do gradiente, transformando o processo em uma "descida de gradiente estocástica". Essa abordagem tende a convergir mais rapidamente em conjuntos de dados maiores, embora os ajustes possam ser menos precisos.

2.1.2 Redes Neurais Convolucionais

Redes neurais convolucionais, ou *convolutional neural networks* (CNNs ou ConvNets), são redes especializadas na análise de imagens e qualquer informação que possa ser representada como uma imagem: sons, uso de palavras em uma sentença, entre outros.

Ao contrário dos *perceptrons* de múltiplas camadas totalmente conectados, as CNNs são, em essência, regularizadas. Elas utilizam operações convolucionais e técnicas de redução (*downsampling*) para identificar padrões hierárquicos que vão se tornando progressivamente mais complexos e de tamanhos menores. Esses padrões são, por fim, achatados (*flattened*), e os nós resultantes são usados como entrada para um *perceptron* comum com função de ativação *softmax* ou *argmax* para a classificação, conforme ilustrado na Figura 2.6.

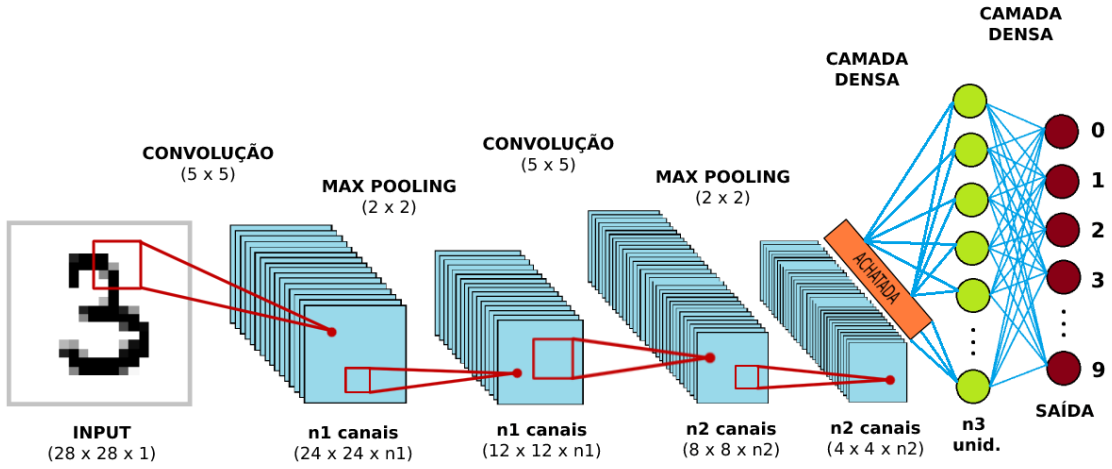

Figura 2.6: Estrutura Genérica de uma Rede Neural Convolucional

Nas CNNs, a camada oculta desempenha o papel de camada de extração de características (*feature learning*). Esse processo é fundamental na análise de imagens, pois, por exemplo, uma imagem RGB com dimensões $n \times m$ pixels resultaria em $n \times m \times 3$ nós na camada de entrada. Consequentemente, na primeira camada oculta com L nós, haveriam $L \times (n \times m \times 3 + 1)$ parâmetros, o que não se mostra escalável, mesmo para imagens de baixa resolução.

Camadas das Redes Neurais Convolucionais

Para compreender melhor a camada oculta, é necessário entender a função das operações principais:

Camada de Convolução e Ativação

A camada convolucional é a principal responsável pela extração de características relevantes da imagem. Ela utiliza múltiplos filtros (ou *kernels*), que geralmente são matrizes quadradas de tamanho variado, *e.g.* 3x3. A convolução aplicada a cada filtro gera um

mapa de características (*feature map*) de tamanho reduzido. O resultado é um volume de N mapas de características, que são passados por uma função de ativação (como a ReLU) antes de serem enviados para a camada de *pooling*.

Os filtros são geralmente inicializados com valores aleatórios, e seus pesos são ajustados durante o treinamento por meio de *backpropagation*. Entre os hiperparâmetros importantes nessa camada, destacam-se o tamanho do filtro (*kernel size*, por exemplo, 2x2 ou 3x3), o *stride* (que indica o número de *pixels* que a janela do filtro se desloca em cada iteração), o *padding* (quantidade de *pixels* adicionados ao redor da imagem) e o número de filtros utilizados. A Figura 2.7 ilustra o resultado de uma convolução aplicada a uma imagem de tamanho $n = 3$ com *padding* $p = 1$ e um filtro de tamanho $f = 2$, movendo-se com um *stride* $s = 1$. O tamanho O da imagem de saída (*feature map*) pode ser calculado pela Fórmula 2.7.

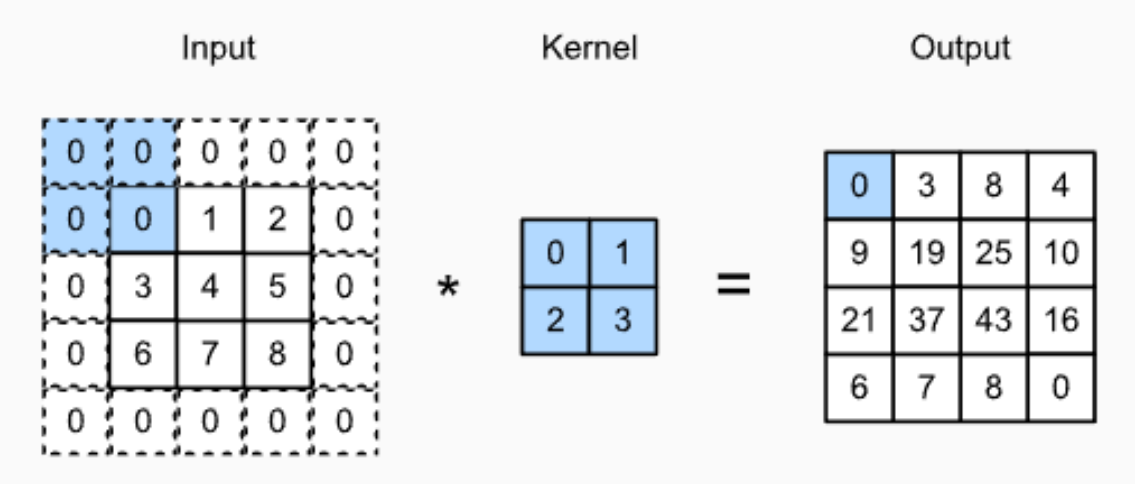

Figura 2.7: Convolução de uma Imagem e um Filtro ($n = 3$, $p = 1$, $s = 1$, $f = 2$) [5]

$$O = \frac{n + 2 * p - f}{s} + 1 \tag{2.7}$$

Camada de *Pooling*

A camada de *pooling* é responsável por reduzir significativamente a complexidade das imagens de entrada através do processo de *downsampling*. As operações de *pooling* mais comuns são o *max pooling* e o *mean pooling*, que selecionam, respectivamente, o maior valor ou a média dos valores dentro de uma sub-região retangular. Os principais hiperparâmetros nesta camada incluem o tipo de *pooling*, o tamanho da janela de *pooling* (*pooling size*) e o *stride*. A Figura 2.8 mostra um exemplo de aplicação de *max pooling* com um *pooling size* de 2 e *stride* de 2, resultando em uma redução de 75% na quantidade total de dados.

[5] https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html
[6] https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network

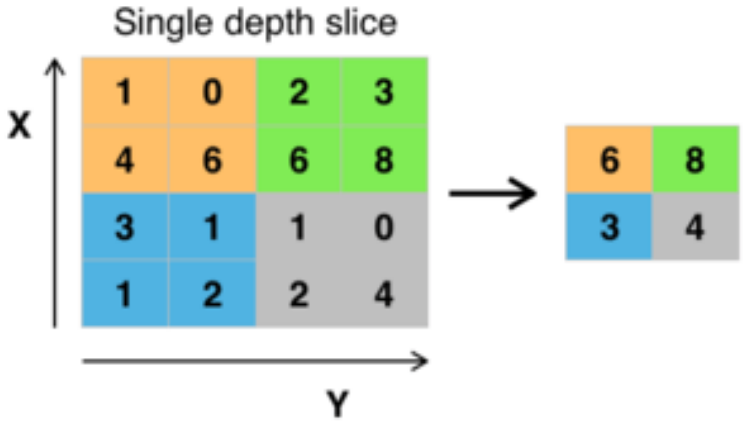

Figura 2.8: Exemplo de Aplicação de uma Camada *Max Pooling* (*size* = 2, *stride* = 2) [6]

Embora haja uma redução de dados, as características mais importantes são preservadas, o que é útil em imagens com regiões de alta similaridade, como a pelagem de um animal ou um fundo uniforme, além de tender a tornar a detecção invariante a rotações. Além disso, o *pooling* ajuda a diminuir o ruído ao descartar ativações menos significativas.

Camada Densa (*Fully Connected, FC*)

A camada densa, também conhecida como camada totalmente conectada, é onde todos os nós estão conectados a todos os nós das camadas adjacentes, tal qual um MLP tradicional. Essa camada recebe o resultado da extração de características, realiza o achatamento (*flattening*), e processa a classificação ou regressão final, se necessário.

Existem várias arquiteturas clássicas de CNNs disponíveis para uso e estudo, como a LeNet e a AlexNet (Krizhevsky et al. 2012), cujas estruturas são diagramadas na Figura 2.9.

Para compreender melhor o funcionamento das CNNs, é interessante observar as diferenças de complexidade entre as redes. A LeNet, proposta por LeCun et al. em 1998, recebe como entrada imagens de 28x28x1, possui 7 camadas e gera um vetor de probabilidades de 10 posições. Entre os hiperparâmetros escolhidos estão filtros de 5x5, *stride* de 2, *padding* de 0, *average pooling* e função de ativação do tipo sigmóide.

Por outro lado, a AlexNet, proposta por Alex Krizhevsky et al. em 2012, foi projetada para imagens maiores e para um contexto muito mais variado, podendo ter até 1000 classes distintas como saída. Com entradas de tamanho 224x224x3, 11 camadas e um vetor de 1000 probabilidades como saída, seus principais hiperparâmetros incluem filtros de 5x5 e 3x3, *strides* de 4 e 2, *padding* de 1 e 2, *max pooling* e ativação ReLU. A AlexNet foi treinada utilizando o *dataset* ImageNet, originalmente composto por 15 milhões de imagens rotuladas e de alta resolução, abrangendo 22 mil classes diferentes.

Até os dias de hoje, essas arquiteturas continuam a apresentar um vasto potencial de

[7] https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network

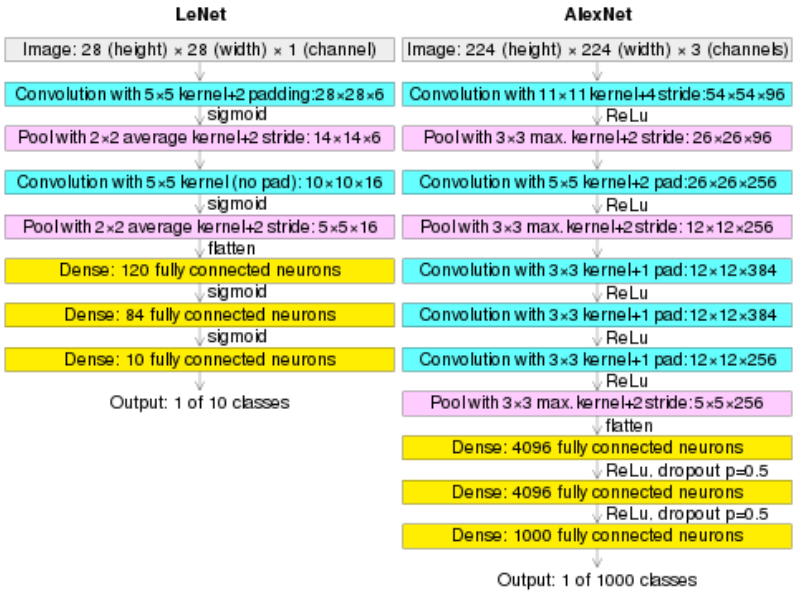

Figura 2.9: Comparação entre as redes LeNet e AlexNet [7]

exploração, sendo sua eficácia fortemente dependente de experimentações e da aplicação do método de tentativa e erro para alcançar resultados satisfatórios em contextos cada vez mais específicos.

Métricas de Avaliação

Para avaliar objetivamente a qualidade das inferências realizadas por algoritmos de classificação e detecção, é possível utilizar diversas métricas distintas. A seguir, são apresentadas as principais consideradas neste trabalho, acompanhadas de suas definições.

Métricas de Acerto

As principais métricas utilizadas para avaliar tarefas de **classificação**, em termos de número de acertos, incluem acurácia, precisão, *recall* e *F1-Score*. Todas essas métricas são baseadas na "matriz de confusão", que fornece uma visualização do desempenho do algoritmo. Para cada predição, a resposta é classificada como verdadeiro positivo (VP), falso positivo (FP), verdadeiro negativo (VN) e falso negativo (FN). É importante notar que uma matriz de confusão também pode descrever a performance para inferências com mais de duas classes.

A principal métrica, e também a mais usada para avaliar tarefas de classificação, é a proporção de positivos e negativos verdadeiros em relação ao número total de predições, conhecida como acurácia, definida por:

$$\text{Acurácia} = \frac{VP+VN}{VP+VN+FP+FN} \tag{2.8}$$

No entanto, no âmbito do aprendizado profundo, nem sempre a acurácia é a melhor

métrica, especialmente em casos com *datasets* desbalanceados (Vakili et al. 2020).

Outras métricas importantes são precisão e *recall*. Precisão indica o quão acurado é o algoritmo em relação a uma classe específica, ou seja, a proporção de verdadeiros positivos (VP) dentre todas as predições classificadas como positivas (VP + FP), conforme descrito por:

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{2.9}$$

O *Recall*, por outro lado, indica a proporção de verdadeiros positivos (VP) em relação ao número de vezes que a classe era realmente positiva (VP + FN), ou seja, quantas vezes foi possível prever corretamente uma classe específica:

$$\text{Recall} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{2.10}$$

Por fim, há o *F1-Score*, também conhecido como *F-Score* ou *F-Measure*, que é a média harmônica entre precisão e *recall*:

$$\text{F1-Score} = 2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}} \tag{2.11}$$

A métrica *mean Average Precision* (mAP) é amplamente utilizada na avaliação de desempenho de algoritmos de detecção de objetos. A mAP fornece uma medida da precisão média para todas as classes em questão. É calculada com base nas curvas de Precisão-*Recall* (*Percision-Recall*, PR) para cada classe.

Para calcular a precisão média (AP) de uma classe específica, a área sob a curva Precisão-*Recall* é computada. Isso geralmente é feito em diferentes pontos de *recall*, resultando em uma média das precisões correspondentes.

Em outras palavras, o AP resume tal gráfico como a média ponderada das precisões alcançadas em cada limiar, com o aumento na *recall* do limiar anterior usado como o peso:

$$\text{AP} = \sum_{n} (R_n - R_{n-1}) P_n \tag{2.12}$$

onde P_n e R_n são a precisão e o *recall* no n-ésimo limiar. Um par (R_k, P_k) é referido como um ponto de operação.

O mAP é então definido como a média das precisões médias (APs) de todas as classes no conjunto de dados, sendo calculado da seguinte forma:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i \tag{2.13}$$

onde N é o número total de classes e AP_i é a *Average Precision* para a classe i.

A métrica mAP é uma das mais robustas para avaliação de algoritmos de detecção de objetos, pois leva em consideração tanto a precisão quanto a capacidade deste de recuperar todas as instâncias de uma classe. Um valor de mAP mais alto indica uma combinação eficaz de alta precisão e alto *recall* para todas as classes, sendo, portanto, considerado um

bom indicador de desempenho geral em problemas de detecção.

Métricas de Velocidade

A métrica **FPS** (*Frames Per Second*) representa o número de imagens que um sistema ou algoritmo é capaz de processar por segundo, sendo amplamente utilizada para avaliar a eficiência de tempo desse sistema. A fórmula para calcular o FPS é:

$$\text{FPS} = \frac{\text{Número de Imagens Detectadas}}{\text{Tempo Decorrido para Detecção dessas Imagens}} \tag{2.14}$$

Por outro lado, a velocidade de inferência é o tempo necessário para que o algoritmo processe uma única imagem e realize uma predição. Ela é geralmente expressa em milissegundos por imagem (ms/imagem). A fórmula para calcular a velocidade de inferência é:

$$\text{Velocidade de Inferência (ms/imagem)} = \frac{\text{Tempo Total de Inferência (ms)}}{\text{Número de Imagens Processadas}} \tag{2.15}$$

Se o FPS do sistema é conhecido, a velocidade de inferência pode ser calculada a partir dele utilizando a seguinte fórmula:

$$\text{Velocidade de Inferência (ms/imagem)} = \frac{1000}{\text{FPS}} \tag{2.16}$$

onde o fator 1000 é utilizado para converter segundos para milissegundos. Portanto, um FPS maior indica uma velocidade de inferência menor, o que significa que o sistema é mais rápido em processar cada imagem.

2.1.3 Modelos de Detecção

Em visão computacional, o termo "modelo" de detecção refere-se a arquiteturas específicas projetadas para localizar e classificar objetos em imagens. Nesse contexto, um modelo representa uma concretização do conceito abstrato de "algoritmo", particularmente quando relacionado a algoritmos derivados de arquiteturas de redes neurais específicas.

Modelos como LeNet e AlexNet, introduzidos na Subseção 2.1.2, podem ser vistos como algoritmos determinísticos, no sentido de que seguem sempre a mesma sequência de passos. No entanto, referimo-nos a eles como "modelos" para enfatizar sua estrutura, dependência de dados e capacidade de aprendizado, aspectos que vão além da ideia tradicional de algoritmo como uma simples sequência de comandos.

A eficácia de tais modelos está intrinsecamente ligada à qualidade dos dados de treinamento, enquanto a escolha do modelo mais apropriado depende da complexidade dos dados e da tarefa a ser realizada. Nesta seção, será explorado o modelo *You Only Look Once* (YOLO), utilizado neste trabalho, abordando sua estrutura, funcionamento e evolução ao longo da última década.

You Only Look Once (YOLO)

You Only Look Once (YOLO) é uma CNN que possibilita a detecção de objetos em tempo real. Suas versões mais recentes são reconhecidas como "estado da arte", especialmente pela sua velocidade e precisão. A primeira versão foi concebida por Redmon et al. (2015) e superou os resultados de generalização obtidos com outras técnicas de detecção, como as CNNs baseadas em região (*Region Based Convolutional Neural Networks*, RCNN, e *Fast* RCNN) e o Modelo com Partes Deformáveis (*Deformable Parts Model*, DPM).

Diferentemente de técnicas como RCNN, que exigem a extração de múltiplas regiões de interesse seguida por uma etapa de inferência, o YOLO, como o próprio nome indica, processa a imagem inteira em uma única passagem pela rede, otimizando o processo de detecção. Isso resulta em um diferencial significativo em termos de velocidade: na sua primeira versão já alcançava 45 e 155 FPS em uma GPU Titan X nas versões completa e reduzida, respectivamente, alcançando até o dobro da mAP de outros métodos vigentes.

O YOLOv1 era proeminente na tarefa de detecção de objetos, que combina a classificação com a localização através de caixas delimitadoras (ou *bounding boxes*). Essa abordagem difere de tarefas como a classificação de imagens, a segmentação de instâncias (que envolve localização precisa) e a segmentação semântica, conforme ilustrado na Figura 2.10. A detecção implica a possibilidade de presença de múltiplos objetos de diferentes classes em uma cena, visando classificar e localizar todos eles por meio de *bounding boxes*.

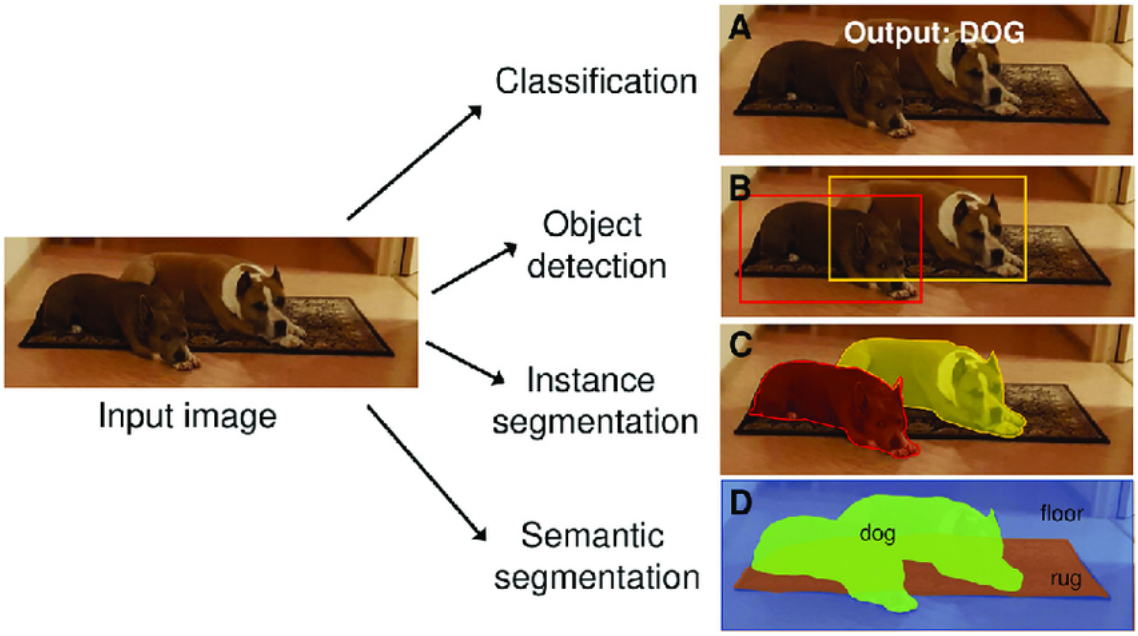

Figura 2.10: Diferença entre classificação, detecção e segmentação [8]

[8] https://akjournals.com/view/journals/1647/14/2/article-p73.xml

Modelos YOLO, no geral, também podem ser utilizados para classificação de imagens. Em tal configuração, a rede é usada com uma cabeça de classificação pré-definida, aproveitando seu *backbone* eficiente para extrair características das imagens.

A técnica usada para a detecção no YOLO divide a imagem em uma grade (*grid*) de dimensão $S \times S$. Para cada célula da grade, são previstas B *bounding boxes*. Cada uma dessas caixas prevê as coordenadas do centro (b_x, b_y), a largura (b_w), a altura (b_h), a probabilidade de conter um objeto (p_c), e as probabilidades para cada uma das C classes $(C_1, C_2, \ldots, C_c)$. A saída da rede é um tensor de tamanho $S \times S \times B \times (5 + C)$, como mostrado na Figura 2.11.

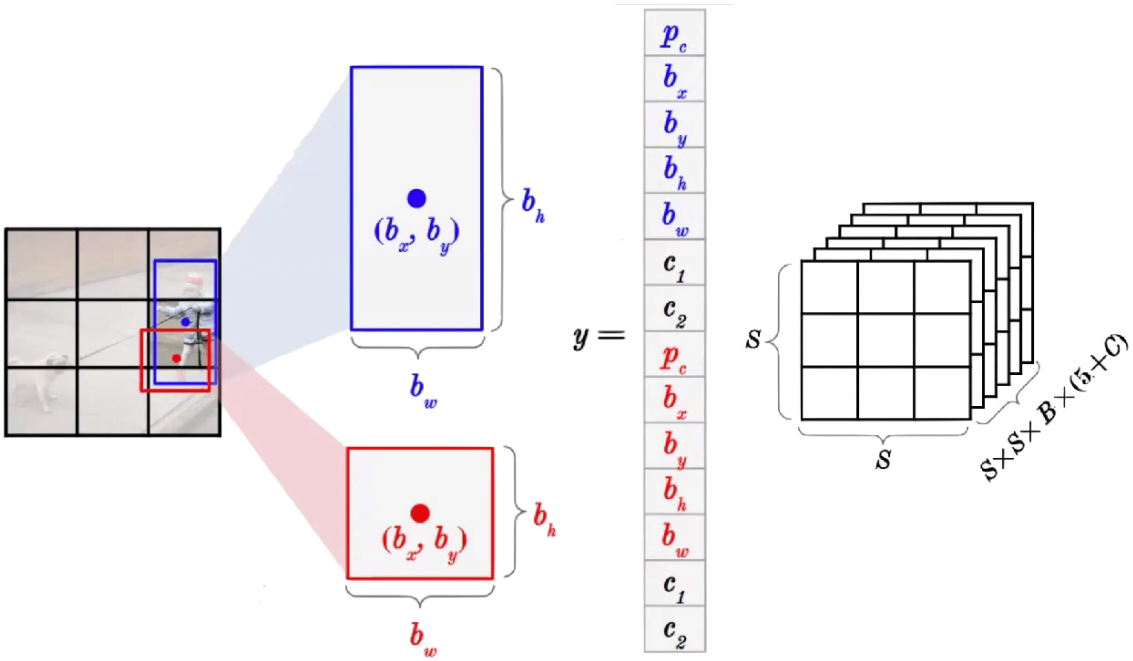

Figura 2.11: Tensor de saída da rede YOLO. S representa a dimensão da grelha de detecção, B o número de *bounding boxes* e C o número de classes [10]

Como ilustrado na Figura 2.12, após dividir a imagem em células, a célula responsável por detectar um objeto será aquela cujo centro do objeto estiver contido dentro de seus limites. Cada célula gera B detecções, correspondendo a B caixas delimitadoras que incluem coordenadas dos centróides, limites das caixas e valores de confiança (probabilidades).

No exemplo em que $S = 3$ e $B = 2$, o total é de 18 caixas. Em versões mais recentes do modelo, valores maiores de S (*e.g.* > 10) podem ser usados em diferentes camadas, além de dezenas de *bounding boxes* por célula. Assim, é possível que centenas ou até milhares de caixas sejam geradas em um único processo *feedforward*.

Dado esse cenário, é necessário filtrar as caixas desnecessárias e manter apenas as que realmente estão localizando um objeto. Esse processo é conhecido como "supressão não-máxima" (*non-max suppression*, NMS) e envolve as seguintes etapas: primeiro, filtrar

[10] https://www.youtube.com/watch?v=971gUEsMEVA

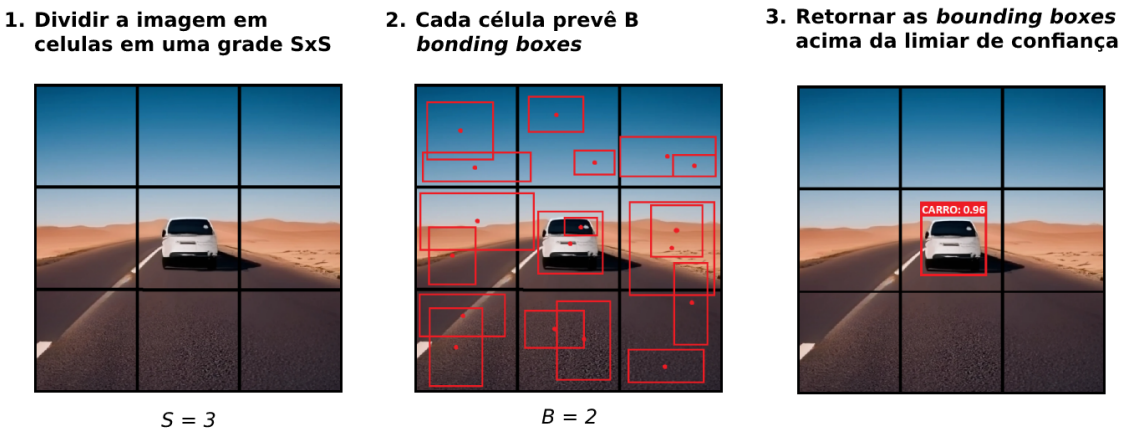

Figura 2.12: *Pipeline* de detecção da rede YOLO

as caixas que tenham uma confiança abaixo de um limite pré-determinado (*e.g.* 0,3), eliminando assim a maioria delas; depois, quando múltiplas caixas com alta confiança e o mesmo rótulo se sobrepõem em uma mesma região, calcula-se a razão de interseção sobre a união (*intersection over union*, IoU) entre elas. Se essa razão ultrapassar um limite pré-definido, as caixas com menor confiança são descartadas. O IoU também funciona como um hiperparâmetro do algoritmo. Sua fórmula é expressa por:

$$\text{IoU} = \frac{\text{Área da Interseção}}{\text{Área da União}} \tag{2.17}$$

Onde a área de interseção é a sobreposição entre a caixa delimitadora predita pelo detector e a caixa delimitadora real (*ground truth*), e a área da união é a soma das áreas das duas caixas delimitadoras, subtraindo a área de interseção.

Se a IoU entre a caixa sugerida pelo detector e a caixa real ultrapassar um limite qualitativo pré-definido, a detecção é considerada um "acerto" ou um "erro". Durante os testes de performance, o IoU é utilizado para calcular a precisão média, ou mAP, que mede a precisão para um determinado limiar considerado como verdadeiro positivo. Diferentes limiares podem levar a diferentes avaliações, sendo que um maior IoU indica uma detecção mais precisa, como mostrado na Figura 2.13. Esse limiar de sobreposição pode ser, por exemplo, 50% (mAP@0.5) ou uma média dos mAPs dentro de um intervalo. Por exemplo, utilizando um passo de 5%, o mAP@[0.5:0.95] representa a média de todas as mAPs (e.g., mAP@0.5 + mAP@0.55 + ... + mAP@0.90 + mAP@0.95) dividida pelo número de médias somadas, que neste caso é 10.

De maneira geral, quando utiliza-se a métrica "mAP", refere-se ao mAP@0.5, que representa a média das precisões médias com uma interseção sobre união de 50%.

Sobre as versões do YOLO, estas são projetadas para oferecer desempenho em tempo real e alta precisão, mesmo com um número reduzido de parâmetros. A arquitetura da rede

[11] https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Figura 2.13: Diferentes níveis de intercessão e possíveis avaliações [11]

YOLOv1 original consiste em 24 camadas convolucionais, 4 camadas de *max pooling* e 2 camadas densas, com imagem de entrada com resolução de 448x448 *pixels* e 3 canais de cores (RGB), conforme ilustrado na Figura 2.14.

Embora tenham surgido vários detectores de estágio único, como o *Single Shot Detector* (SSD), o *Deconvolutional Single Shot Detector* (D-SSD) e o RetinaNet, a família de arquiteturas YOLO se destaca especialmente por sua compatibilidade com os requisitos industriais de eficiência e capacidade de implantação em dispositivos de borda com recursos limitados. A evolução das variantes do YOLO continua a atender a essas exigências, culminando na versão mais recente disponível no momento da escrita deste trabalho, o YOLOv8. Esta versão, juntamente com as iterações anteriores, será discutida na seção a seguir.

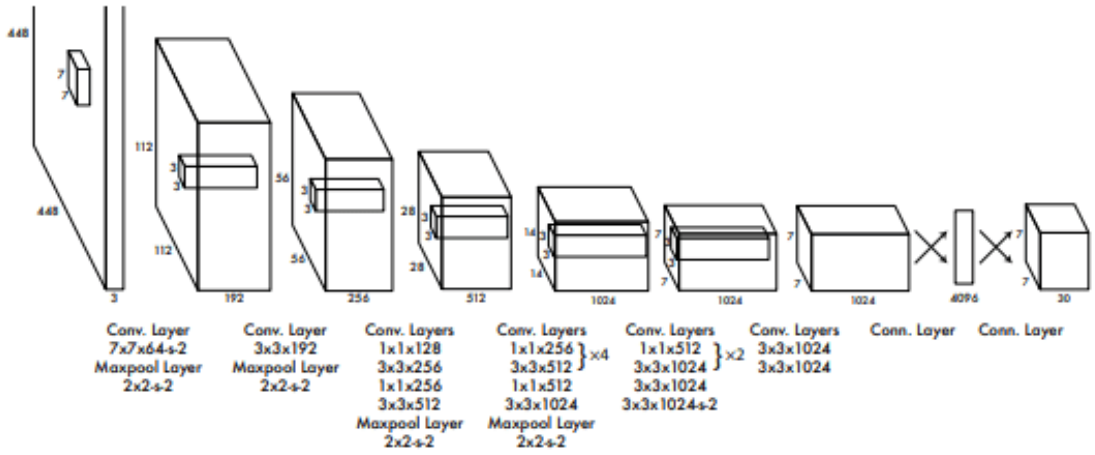

Figura 2.14: Arquitetura original YOLO (Redmon et al. 2015)

Linha Evolutiva

A arquitetura YOLO passou por uma série de evoluções desde seu lançamento, com cada versão trazendo melhorias para a área de visão computacional em tempo real. Após o marco inicial em 2015, o YOLOv2 (Redmon & Farhadi 2016) introduziu melhorias notáveis, como a implementação de *anchor boxes*, que aumentaram a precisão ao detectar objetos de diferentes tamanhos. Além disso, o modelo passou a adotar o *backbone Darknet-19*, composto por 19 camadas de convolução, resultando em uma melhoria da etapa de extração de características. Já em 2018, o YOLOv3 (Redmon & Farhadi 2018) elevou ainda mais a capacidade de detecção multiescala, utilizando *upsampling* e concatenação de *features* em diferentes níveis, permitindo a identificação de objetos em tamanhos variados dentro de uma mesma imagem. Essa versão trouxe o *Darknet-53*, com 53 camadas de convolução, melhorando as métricas de desempenho sem comprometer a característica de detecção em tempo real.

No lançamento do YOLOv4 (Bochkovskiy et al. 2020), a comunidade de desenvolvimento focou na otimização de hiperparâmetros e na introdução de uma função de perda baseada no Índice IoU, o que resultou em melhorias na precisão da detecção, especialmente para objetos pequenos. O YOLOv4 continuou a ser eficiente, unindo alta precisão com um desempenho de 50 FPS em uma GPU Tesla P100. No mesmo ano, a Ultralytics lançou o YOLOv5 (Jocher 2020), que marcou uma grande mudança ao migrar do *framework Darknet*[12] para o *PyTorch*. Essa mudança facilitou o desenvolvimento e a personalização do modelo. O YOLOv5 também introduziu variantes de diferentes tamanhos (*e.g.* YOLOv5 *nano*, YOLOv5 *large*, *etc*), permitindo o *tradeoff* entre modelos mais rápidos ou mais precisos, a depender da aplicação. Além disso, aprimorou a exportação para plataformas como *ONNX, CoreML* e *TensorRT*, facilitando a integração com dispositivos móveis e sistemas de borda, mesmo em situações onde recursos computacionais são limitados.

Em 2021, o YOLOR (Wang et al. 2021), uma variação do YOLOv4, introduziu o aprendizado multitarefa, combinando detecção, classificação e estimativa de pose em uma única rede. Essa abordagem aumentou o nível de integração e possibilidades no processamento de dados visuais complexos. Ainda em 2021, a *Megvii Technology* desenvolveu o YOLOX (Ge et al. 2021), que simplificou o processo de detecção de *bounding boxes* ao reintroduzir um método sem âncoras, trazendo uma nova melhoria na detecção de objetos de diferentes tamanhos.

Em 2022, o lançamento do YOLOv6 (Li et al. 2022) trouxe avanços na detecção de objetos pequenos, aproveitando aprimoramentos em *frameworks* como *OpenCV* e *TensorFlow*, mantendo a velocidade de processamento. Na sequência, o YOLOv7 (Wang, Bochkovskiy & Liao 2022) aprimorou ainda mais a detecção de objetos em diferentes escalas com a introdução do bloco *Extended Efficient Layer Aggregation Network* (E-ELAN) e das redes piramidais de características.

Por fim, em 2023, a Ultralytics lançou o YOLOv8 (Jocher et al. 2023), trazendo uma arquitetura otimizada e aprimoramentos nas tarefas de detecção, segmentação e classificação de objetos. Essa versão também introduziu novas funcionalidades, como a estimação de pose e a detecção orientada, ampliando seu escopo de aplicações. Comparado

[12] `https://github.com/pjreddie/darknet`

ao YOLOv5, último modelo da empresa, o YOLOv8 introduz um modelo sem âncoras, que prevê diretamente o centro dos objetos, eliminando a necessidade de *anchor boxes* pré-definidas. Essa mudança simplifica o processo de detecção e melhora a eficiência do algoritmo de NMS. Outras melhorias incluem a substituição da convolução 6x6 do módulo inicial por uma convolução 3x3, além da reformulação da estrutura `C3` para `C2f`, onde todas as saídas do *bottleneck* são concatenadas, em contraste com o `C3`, que utilizava apenas a saída final. Essas modificações resultam em uma redução no número de parâmetros e no tamanho dos tensores de saída. A estrutura completa do YOLOv8 é ilustrada na Figura 2.15.

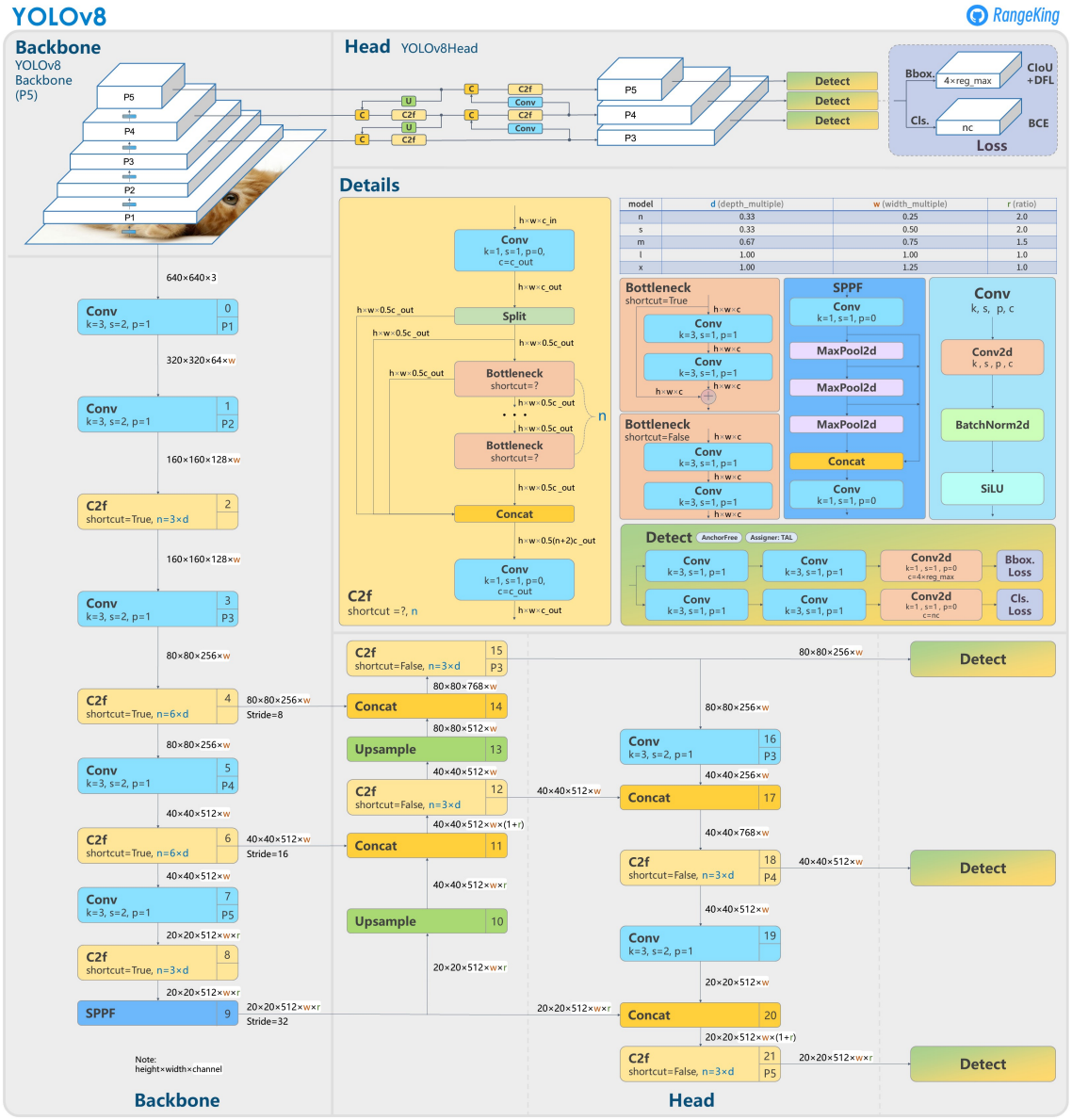

Figura 2.15: Arquitetura YOLOv8 [13]

Adicionalmente, o YOLOv8 utiliza uma técnica de aumento de dados chamada mosaico, que combina quatro imagens para criar variações durante o treinamento. Dito isto, essa técnica é desativada nas últimas épocas de treinamento para evitar a degradação do desempenho.

O YOLOv8 demonstra resultados notáveis em *benchmarks* de precisão, como o COCO e o Roboflow 100. No *benchmark* COCO, o modelo YOLOv8m alcançou uma mAP de 50,2%[14], uma das melhores entre os modelos com latências de inferência comparáveis. No Roboflow 100, que avalia o desempenho dos modelos em domínios específicos, o YOLOv8 superou o YOLOv5, mostrando menos *outliers* e um mAP superior em comparação com modelos anteriores.

A evolução do YOLO ao longo dos anos não apenas aprimorou significativamente sua capacidade na detecção de objetos, como também expandiu sua aplicação em uma ampla gama de indústrias, incluindo veículos autônomos, agricultura de precisão e controle de qualidade em processos de manufatura. Isso consolidou o YOLO como uma das ferramentas mais versáteis e eficientes em visão computacional. Embora novas versões continuem a ser desenvolvidas, até o momento da elaboração desta dissertação, as arquiteturas mais recentes e seus pesos pré-treinados ainda não foram amplamente disponibilizados. A Figura 2.16 ilustra a linha do tempo evolutiva da arquitetura YOLO, desde a sua concepção em 2015 até 2023.

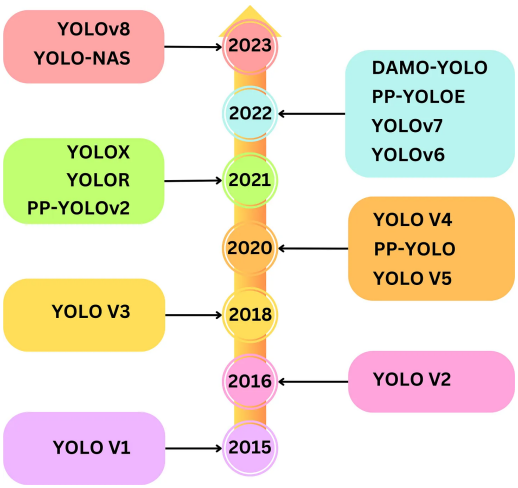

Figura 2.16: Linha do tempo da evolução da arquitetura YOLO, da versão 1 (2015) à versão 8 (2023) [16]

[13] https://github.com/ultralytics/ultralytics/issues/189
[14] https://docs.ultralytics.com/pt/datasets/detect/coco/
[16] https://newsletter.theaiedge.io/p/deep-dive-how-yolo-works-part-1-from

2.2 Engenharia de *Software*

Software, como é comumente conhecido, não se resume apenas a programas de computador, mas inclui também documentação, dados de configuração e processos necessários para sua operação adequada. A engenharia de *software* é uma disciplina que abrange todos os aspectos da criação de sistemas de *software*, indo da especificação inicial à manutenção após a implementação (Sommerville 2011).

O engenheiro de *software* visa entregar resultados de qualidade dentro dos prazos e orçamentos disponibilizados, frequentemente fazendo compromissos para balancear as diversas exigências e limitações do projeto. Enquanto desenvolvedores de *software* podem dedicar mais tempo ao processo de escrita de programas, engenheiros, por sua vez, seguem uma abordagem estruturada para garantir a produção de *software* de alta qualidade. Essa abordagem envolve a aplicação de teorias, métodos e ferramentas de forma seletiva, buscando soluções adequadas para problemas específicos, enquanto leva em consideração as restrições organizacionais e financeiras impostas.

A relevância de adotar uma abordagem estruturada está ligada a dois pontos principais:

- A crescente dependência de sistemas de *software* avançados exige que estes sejam desenvolvidos de forma confiável, segura, econômica e ágil (Sommerville 2011). Sistemas modernos precisam ser escaláveis, flexíveis e, acima de tudo, garantir a segurança necessária para proteger dados e assegurar a integridade dos processos, especialmente em um contexto de evolução tecnológica acelerada e demandas dinâmicas do mercado.
- A utilização de métodos de engenharia de *software* é, a longo prazo, mais econômica do que abordagens *ad hoc*. Isso ocorre porque os maiores custos associados ao *software* geralmente estão relacionados às alterações necessárias após sua entrada em operação (Sommerville 2011), sendo mais vantajoso aplicar práticas estruturais para reduzir custos com manutenção e ajustes durante seu ciclo de vida.

Uma abordagem estruturada na engenharia de *software* é chamada de **processo de *software*** (Sommerville 2011), que consiste em uma série de atividades fundamentais que levam à criação de um produto de *software*. Essas atividades incluem a especificação, onde clientes e engenheiros definem o *software* a ser desenvolvido e as limitações de seu funcionamento; o desenvolvimento, onde o *software* é projetado e codificado; a validação, que garante que o *software* atenda aos requisitos do cliente por meio de testes rigorosos; e a evolução, onde o *software* é ajustado para refletir as mudanças nas necessidades do cliente e no ambiente de operação (Sommerville 2011). Essa sequência de atividades assegura que o produto final esteja alinhado com as expectativas do cliente, seja capaz de se adaptar às mudanças e ofereça a qualidade desejada ao longo de seu ciclo de vida.

Nas subseções seguintes, serão apresentados conceitos fundamentais do processo de *software*, abordando desde a importância do projeto ou seleção de uma arquitetura de referência de um determinado domínio até a fase de especificação do produto, onde serão detalhadas abordagens comuns para definição de requisitos e algumas técnicas de modelagem visual de sistemas.

2.2.1 Arquitetura de Referência

> *Sistemas de software são construídos para satisfazer os objetivos de negócios das organizações. A arquitetura é uma ponte entre esses objetivos de negócios (frequentemente abstratos) e o sistema final (concreto) resultante. Embora o caminho dos objetivos abstratos para os sistemas concretos possa ser complexo, a boa notícia é que as arquiteturas de software podem ser projetadas, analisadas, documentadas e implementadas utilizando técnicas conhecidas que apoiarão o alcance desses objetivos de negócios e missão. A complexidade pode ser domada, tornada tratável.*
>
> *(Bass et al. 2012)*

O conceito de arquitetura de referência de *software* refere-se a uma estrutura genérica desenvolvida para uma determinada classe de sistemas, servindo como base para o *design* de arquiteturas concretas dentro desse domínio (Angelov et al. 2012). Devido à sua natureza abrangente, essa arquitetura não é detalhadamente definida para um único contexto, o que torna a definição de objetivos e o processo de modelagem tarefas desafiadoras e frequentemente permeadas por incertezas (Angelov et al. 2012).

Essas arquiteturas surgiram como abstrações de arquiteturas concretas de *software* em domínios específicos e têm papel fundamental no desenvolvimento de novas soluções, atuando tanto como fonte de inspiração quanto como ferramenta de padronização (**?**). Com o aumento da complexidade dos sistemas de *software* e a crescente demanda por processos de *design* mais eficientes e altos níveis de interoperabilidade, a relevância das arquiteturas de referência no cenário atual tornou-se ainda mais evidente.

No desenvolvimento de *software* – inclusive em abordagens ágeis – a definição da arquitetura global do sistema é uma das primeiras etapas do processo de *design* (Sommerville 2011). Essa etapa inicial é importante, pois, embora a refatoração de componentes individuais seja relativamente simples, a alteração da arquitetura como um todo pode ser extremamente custosa. A arquitetura estabelece o elo crítico entre a engenharia de requisitos e o *design*, definindo os principais componentes estruturais do sistema e suas inter-relações.

Por serem de natureza abrangente, elas são projetadas para atender às funcionalidades e qualidades desejadas por todas as partes interessadas (*stakeholders*) em seus contextos específicos. Sobre estas partes, não há um grupo claro de partes interessadas em uma arquitetura de referência (Angelov et al. 2008).

As partes interessadas podem ser todas as empresas do domínio, todas as empresas que desenvolvem *software* para o domínio, *etc*. No entanto, não é possível envolver todas essas partes interessadas na definição de uma arquitetura de referência (por razões logísticas, políticas, *etc*) (Angelov et al. 2008). A arquitetura é uma apresentação de alto nível do sistema que pode ser usada como foco para discussões por essas diferentes partes (Sommerville 2011).

Para cumprir tal objetivo, a arquitetura precisa omitir certas informações sobre elementos que não são úteis para o raciocínio de um sistema genérico. Segundo Bass et al. (2012), essa abstração é essencial para domar a complexidade de um sistema - *"simples-*

mente não podemos, e não queremos, lidar com toda a complexidade o tempo todo" (Bass et al. 2012).

De maneira geral, uma estrutura é considerada arquitetural se suporta o raciocínio sobre o sistema e suas propriedades. Esse raciocínio deve abordar um atributo do sistema importante para alguma parte interessada. Entre esses atributos, vinculados aos requerimentos não funcionais, podem estar a disponibilidade do sistema diante de falhas, a dificuldade de realizar mudanças específicas no sistema, a capacidade de resposta do sistema às solicitações dos usuários, entre outros (Bass et al. 2012). Sendo assim, as decisões de *design* arquitetônico têm um efeito profundo sobre se o sistema pode ou não atender a esses tipos de requisitos críticos, como o desempenho, a robustez, a distribuibilidade, a confiabilidade, a manutenibilidade, *etc* (Bosch 1999, Sommerville 2011).

No que difere de uma arquitetura concreta, uma arquitetura de referência precisa abordar mais requisitos funcionais e atributos de qualidade, dada a sua aplicação em diversos contextos. Como Bosch (1999) discute, os componentes individuais da arquitetura implementam os requisitos funcionais do sistema. Os requisitos não funcionais dependem da arquitetura do sistema - a forma como esses componentes estão organizados e se comunicam. Em contrapartida, arquiteturas de *software* concretas são concebidas para contextos específicos, refletindo objetivos de negócio bem definidos das partes interessadas (Angelov et al. 2012).

Devido a essas diferenças entre arquiteturas concretas e arquiteturas de referência, estas últimas são consideradas por alguns autores como muito distantes das arquiteturas concretas: *"Arquiteturas de referência não são arquiteturas; são conceitos úteis que capturam elementos de uma arquitetura."* (Clements 2003, Angelov et al. 2008) (de Morais Barroca Filho 2019).

Dito isso, todo sistema concreto possui uma arquitetura subjacente, mesmo que essa arquitetura não seja necessariamente conhecida por alguém. É possível que todas as pessoas que projetaram tenham ido a óbito, que a documentação tenha desaparecido (ou nunca tenha sido produzida), que o código-fonte tenha sido perdido (ou nunca entregue) e que tudo o que reste seja o código binário em execução (Bass et al. 2012). Isso destaca a diferença entre a arquitetura de um sistema e a representação dessa arquitetura. Como uma arquitetura pode existir independentemente de sua descrição ou especificação, isso ressalta a importância da documentação arquitetural (Bass et al. 2012).

Sobre tipos de documentação, segundo Sommerville (2011), diagramas de blocos são uma forma apropriada de descrever a arquitetura do sistema durante o processo de *design*, pois são uma boa maneira de apoiar a comunicação entre as pessoas envolvidas no processo. Em muitos projetos, esses diagramas frequentemente constituem a única documentação arquitetural existente (Sommerville 2011).

Para exemplificar a documentação de uma arquitetura de referência, podemos considerar a arquitetura desenvolvida por de Morais Barroca Filho (2019) para o domínio de aplicações de saúde baseadas em IoT apresentada na Figura 2.17.

A ilustração demonstra uma arquitetura organizada em diferentes camadas, o fluxo de dados e informações entre elas e como diferentes componentes são agrupados para atender aos requisitos funcionais e não funcionais do domínio mais amplo. Evidencia-se, ainda, como certos componentes podem ser selecionados para instanciar diferentes tipos

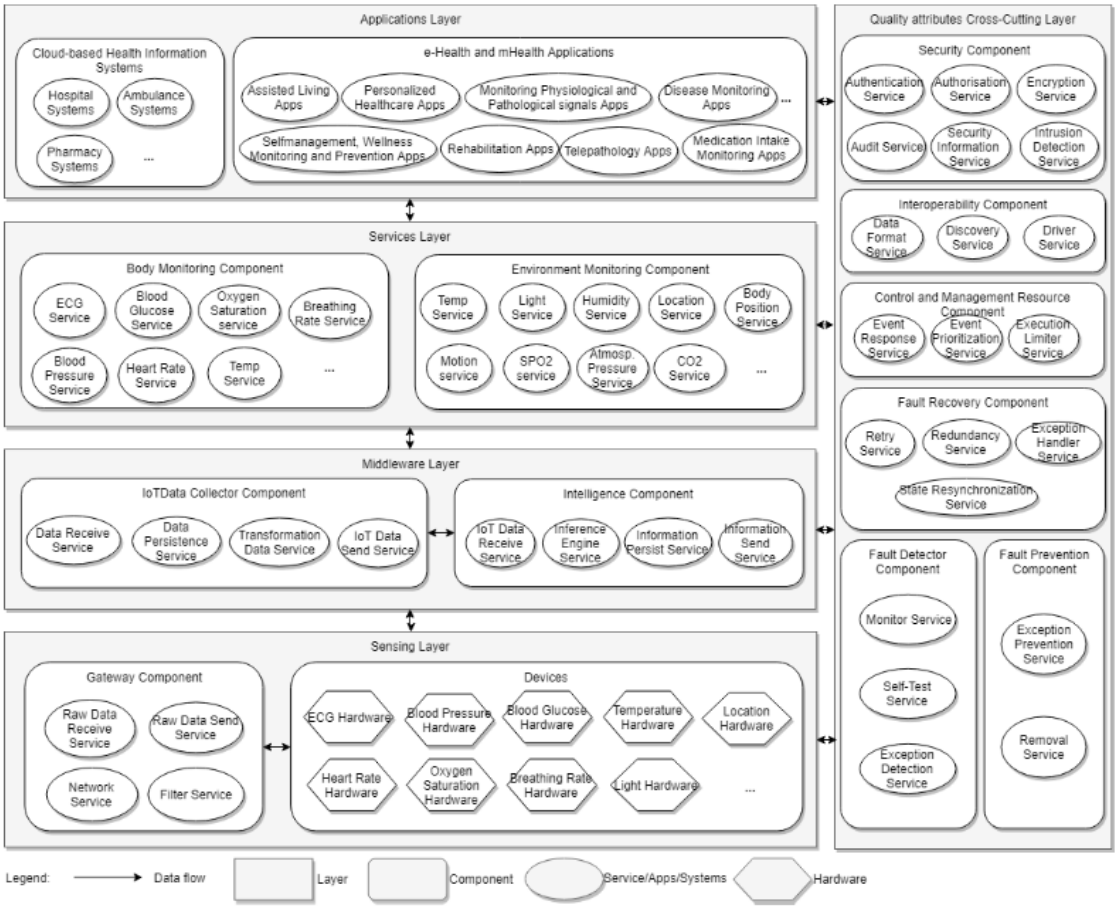

Figura 2.17: Arquitetura de Referência para Aplicações de Saúde Baseadas em IoT por de Morais Barroca Filho (2019)

de aplicações, cada uma adaptada a requisitos específicos dentro desse contexto.

2.2.2 Engenharia de Requisitos

De acordo com Sommerville (2011), antes da implementação de um sistema em larga escala, ocorre uma etapa fundamental conhecida como engenharia de requisitos (ER).

Os requisitos de um sistema são descrições das funcionalidades e restrições que ele deve atender. Em outras palavras, tratam-se dos serviços que o sistema deve oferecer e das limitações sob as quais ele deve operar. Esses requisitos refletem as necessidades dos clientes, que demandam um sistema para cumprir objetivos específicos, como controlar dispositivos, processar pedidos ou gerar relatórios (Sommerville 2011). O processo de identificar, analisar, documentar e verificar essas funcionalidades e restrições é a ER (Sommerville 2011).

Ainda segundo Sommerville (2011), os requisitos podem ser classificados em duas categorias principais:

1. **Requisitos funcionais**: Definem os serviços que o sistema deve fornecer, especificando como ele deve reagir a determinados *inputs* e comportar-se em cenários específicos.
2. **Requisitos não funcionais**: Estabelecem as restrições associadas aos serviços do sistema, como limitações de desempenho, restrições de desenvolvimento ou requisitos derivados de normas e padrões. Esses requisitos frequentemente se aplicam ao sistema como um todo e envolvem propriedades emergentes, como confiabilidade, manutenibilidade e segurança.

Na fase inicial do ciclo de desenvolvimento, ocorre a coleta de informações sobre os processos operacionais e as regras de negócio. Essa etapa é crucial para subsidiar a futura prototipação do *software* a ser desenvolvido (Souza & Campo 2023), com o objetivo de alinhar expectativas entre desenvolvedores e clientes.

Durante a elicitação de requisitos, algumas técnicas podem ser empregadas:

- **Observação**: Envolve visitas ao local onde a solução será aplicada, permitindo uma compreensão mais detalhada do problema e de seus processos adjacentes.
- **Entrevistas**: Consistem na formulação de perguntas, muitas vezes baseadas em observações prévias, para identificar melhorias necessárias e funcionalidades desejadas (Souza & Campo 2023).

Essas técnicas, entre outras, exploram diferentes perspectivas dos *stakeholders*, refinando continuamente e progressivamente as informações coletadas (Schneider et al. 2020).

Para garantir que os requisitos sejam bem compreendidos por todas as partes interessadas (*e.g.* usuários, consumidores, fornecedores, desenvolvedores, *etc*), eles geralmente são expressos em linguagem natural. Essa abordagem prioriza a simplicidade, clareza e coerência, fazendo uso de tabelas, formulários ou diagramas intuitivos, e evitando jargões técnicos ou notações formais (van Lamsweerde 2014, Sommerville 2011).

O resultado dessa etapa é o documento de requisitos, que pode integrar o contrato de desenvolvimento do sistema. Contudo, os requisitos não são estáticos: mudanças podem ocorrer ao longo do processo. À medida que o sistema é desenvolvido, os requisitos se tornam mais detalhados e específicos, refletindo uma compreensão mais profunda do problema e das necessidades envolvidas (Sommerville 2011, Schneider et al. 2020).

2.2.3 Modelagem de *Software*

Após o levantamento dos requisitos iniciais do sistema, o próximo passo é a modelagem. A modelagem de um sistema consiste no processo de desenvolvimento de representações abstratas que refletem diferentes perspectivas ou visualizações do sistema (Sommerville 2011). Esses modelos podem ser analisados para melhorar a compreensão do sistema, verificar o atendimento aos requisitos e propor alterações antes mesmo da implementação do código (Unhelkar 2018).

A utilidade dos modelos se estende por todas as etapas da engenharia de *software*. Durante a engenharia de requisitos, eles auxiliam na derivação e comunicação desses requisitos. No desenvolvimento do sistema, fornecem descrições que guiam os engenheiros durante a implementação. E mesmo após a implementação, os modelos documentam a estrutura e operação do sistema, tornando-se ferramentas essenciais para manutenção e evolução deste (Sommerville 2011).

Modelos podem ser criados tanto para novos sistemas, explicando os requisitos propostos às partes interessadas, quanto para sistemas existentes, documentando suas funcionalidades. No segundo caso, esses modelos podem servir como base para discutir pontos fortes e fracos, orientar a criação de novos requisitos ou até mesmo planejar um sistema reestruturado. Nesse contexto, eles atuam como facilitadores da comunicação entre as partes interessadas, promovendo processos criativos mais ágeis e coerentes (Schneider et al. 2020).

Uma característica importante da modelagem é sua natureza visual, frequentemente realizada por meio da Linguagem de Modelagem Unificada (*Unified Modeling Language*, UML). A UML é uma padronização de simbologias para representar sistemas baseados em *software*, funcionando de forma análoga a uma planta baixa na construção civil (Group 2017). Não é uma linguagem de programação, mas um conjunto de notações gráficas que descrevem aspectos estruturais e comportamentais de sistemas.

Surgida em 1997, a UML unificou padrões predecessores dos anos 1980, como o método Booch, a *Object Modeling Technique* (OMT) e a *Object-Oriented Software Engineering* (OOSE) (Gomaa 2011), e se tornou o padrão de modelagem no paradigma orientado a objetos.

Um paradigma de programação pode ser definido como um modelo, padrão ou estilo de desenvolvimento de *software*, suportado por linguagens que compartilham características comuns (Gonçalves & Cortés 2015, Guedes 2018, p. 12). Entre os paradigmas mais populares estão o paradigma estruturado e o paradigma orientado a objetos, cada um com técnicas de modelagem específicas.

No paradigma estruturado, por exemplo, destacam-se outros métodos tradicionais como o Diagrama de Fluxo de Dados (DFD), o Modelo Entidade-Relacionamento (MER)

e o Diagrama Entidade-Relacionamento (DER) (Gonçalves & Cortés 2015, Guedes 2018, p. 12).

A UML possibilita a criação de diversos diagramas que representam tanto a estrutura estática quanto o comportamento dinâmico do sistema. A Figura 2.18 esquematiza os diferentes tipos de diagramas da UML (Group 2017, p. 685):

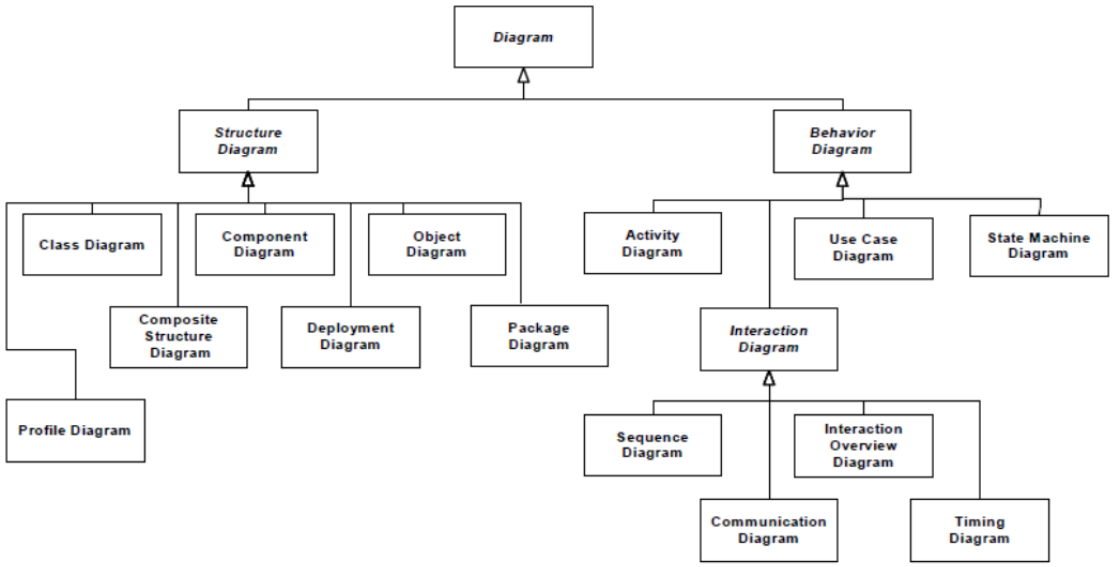

Figura 2.18: Classificação de Diagramas UML

Esses diagramas podem ser classificados em duas visões principais (Guedes 2018):

- **Visão estática (ou estrutural):** Foca na estrutura do sistema, incluindo objetos, operações e relacionamentos. Exemplos: diagramas de classes e de estrutura composta.
- **Visão dinâmica (ou comportamental):** Foca no comportamento do sistema, mostrando colaborações entre objetos e mudanças de estado. Exemplos: diagramas de sequência, de atividades e de estados.

A seguir, são descritas as principais classes de diagramas UML que possuem relevância para esta dissertação:

Diagrama de Classes

O diagrama de classes é uma ferramenta estrutural que descreve a composição interna das entidades envolvidas no sistema, incluindo classes, atributos, métodos e seus relacionamentos. É considerado o mais importante da UML por alguns autores, pois serve como referência para outros diagramas (Gonçalves & Cortés 2015, Guedes 2018). A Figura 2.19 apresenta um exemplo de diagrama de classes (Gonçalves & Cortés 2015):

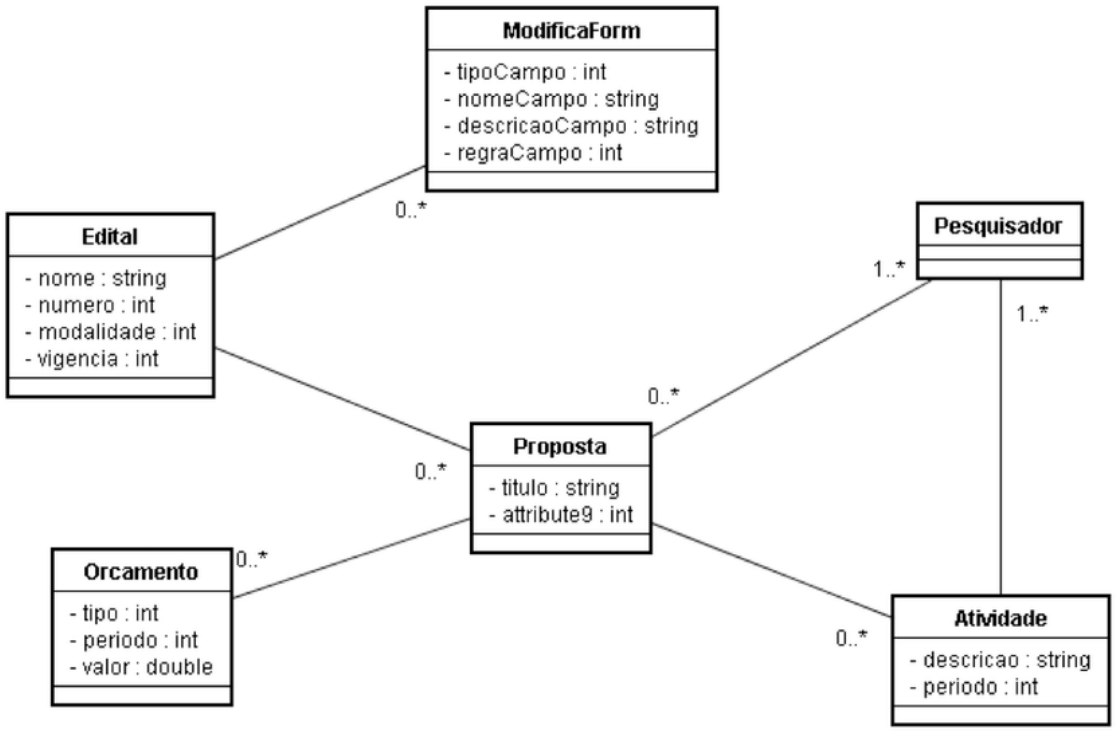

Figura 2.19: Exemplo de Diagrama de Classes

Diagrama de Caso de Uso

Conforme explicado por Booch & Rumbaugh (2006), o diagrama de caso de uso descreve as interações entre diferentes atores e o sistema, documentando como ele responderá às ações dos usuários. Geralmente, é acompanhado de documentação que detalha essas interações. A Figura 2.20 ilustra um exemplo desse diagrama (Gonçalves & Cortés 2015):

Diagrama de Componentes

Esse diagrama representa os componentes de um sistema de forma abstrata, conectados pelo fluxo de dados entre eles. Ele detalha os tipos de dados que fluem e as interfaces de comunicação entre os componentes (de Morais Barroca Filho 2019). "Componente" refere-se a um módulo de classes que representa subsistemas independentes com capacidade de interação. A Figura 2.21 exemplifica um diagrama de componentes (Gonçalves & Cortés 2015):

Essa visualização é essencial para esclarecer aos engenheiros responsáveis como os dados devem ser processados em cada componente, oferecendo maior clareza técnica para o desenvolvimento (de Morais Barroca Filho 2019).

Em projetos de *software*, são criados tantos diagramas quanto necessários para ilustrar a estrutura e os comportamentos do sistema de forma clara e objetiva (Schneider et al. 2020). Esses diagramas tornam-se parte da documentação oficial do *software*, essencial

Figura 2.20: Exemplo de Diagrama de Caso de Uso

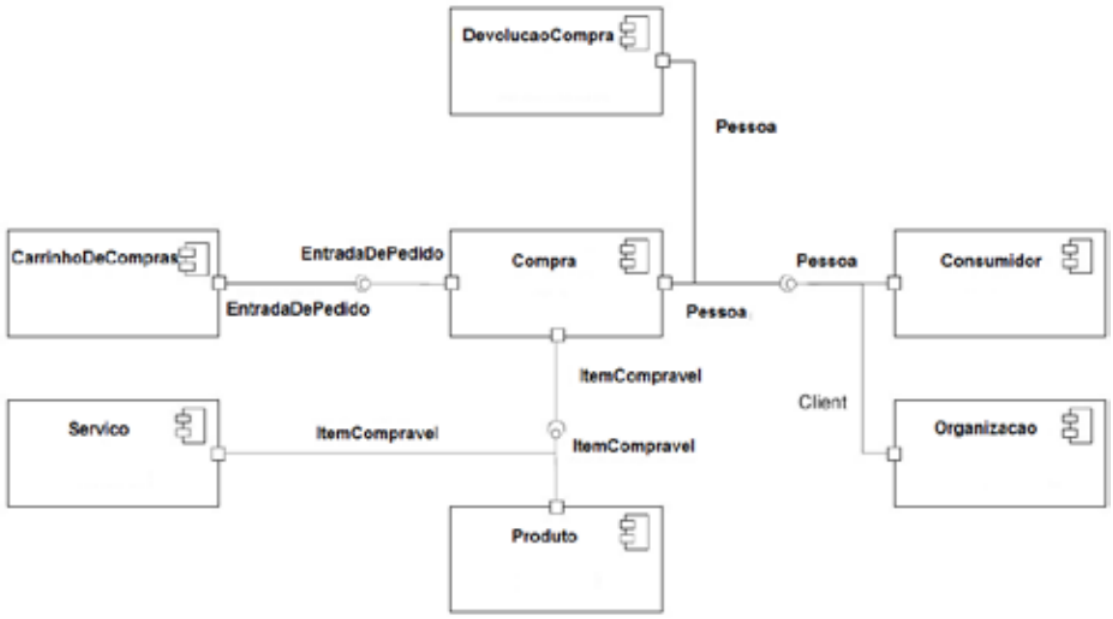

Figura 2.21: Exemplo de Diagrama de Componentes

para sua construção, evolução e manutenção (Sommerville 2011, Guedes 2018).

Capítulo 3

Trabalhos Relacionados

Nesta seção, serão apresentados e discutidos os estudos relacionados ao presente trabalho, organizados em três subseções. A Seção 3.1 aborda a metodologia utilizada para a seleção dos estudos relevantes. Em seguida, a Seção 3.2 explora pesquisas sobre modelos e técnicas específicas de aprendizado de máquina aplicadas à detecção de defeitos, com ênfase maior na aplicação na indústria têxtil. Por fim, a Seção 3.3 analisa trabalhos que apresentam plataformas ou sistemas de informação voltados à automação de processos no setor têxtil, destacando as expectativas e demandas associadas a esse tipo de solução no contexto fabril.

3.1 Procedimento de Aquisição de Estudos

Nesta seção, detalha-se o procedimento adotado para a aquisição de trabalhos relacionados e relevantes a esta dissertação. As buscas foram realizadas utilizando o *Google Acadêmico*[1] como ferramenta principal, empregando termos que abrangem tanto técnicas de aprendizado de máquina aplicadas à detecção de defeitos quanto termos relacionados à prevalência de sistemas de *software* no contexto da indústria têxtil.

Dos termos pesquisados, relevantes ao tópico de métodos de detecção de defeitos na indústria têxtil, foram utilizados: "Modelos de Detecção de Defeitos na Indústria Têxtil", "*Framework* para Detecção e Classificação de Defeitos em Tecidos", "Métodos de Detecção de Defeitos Texturais", "Inspeção de Qualidade de Tecidos", "Processamento de Imagens na Indústria Têxtil" e "Técnicas de Aprendizado Profundo para Detecção de Defeitos". Quanto aos termos relacionados a plataformas de *software* para esse setor, destacam-se: "Plataformas *Software* de Inspeção de Qualidade na Indústria Têxtil" e "Engenharia de *Software* na Indústria Têxtil". As buscas foram realizadas para os termos tanto em português quanto em inglês.

Para cada termo pesquisado, foram selecionados os 20 primeiros artigos retornados, ordenados por relevância pela ferramenta de pesquisa. Após a aquisição dos estudos, alguns filtros foram aplicados para determinar os artigos a serem considerados. O primeiro critério de filtragem foi a data de publicação, limitando a análise a artigos publicados nos últimos cinco anos, ou seja, até no máximo 2019, considerando o período de desenvolvimento deste estudo. Como segundo critério, foram mantidos apenas os artigos que

[1] `https://scholar.google.com.br`

apresentaram maior alinhamento com um ou mais objetivos específicos desta dissertação.

Esse processo sistemático de busca e filtragem permitiu identificar estudos atuais e pertinentes, que ajudaram a fundamentar as discussões, sustentar as proposições e fortalecer os resultados apresentados ao longo deste trabalho.

3.2 Metodologias de Detecção de Defeitos em Tecidos

Czimmermann et al. (2020) averiguaram diversas técnicas automatizadas de detecção de defeitos visuais em materiais variados, como metais, cerâmicas e tecidos. Eles categorizam os defeitos em tipos visíveis e palpáveis e descrevem métodos de processamento visual artificial para interpretação matemática e lógica de amostras. O levantamento destaca métodos de detecção de defeitos texturais, incluindo abordagens estatísticas e estruturais, e avalia sua eficiência computacional e precisão. O estudo sublinha as limitações de técnicas comuns, como filtros de Gabor, que são altamente dependentes dos padrões de material e textura, e discute os desafios na segmentação e janela de objetos sobrepostos, especialmente com características de cor. Redes neurais e aprendizado profundo são reconhecidos por suas robustas capacidades de classificação, embora exijam grandes volumes de dados de treinamento e recursos computacionais significativos. Os autores enfatizam a necessidade da indústria por métodos de detecção de defeitos generalizados que possam lidar com defeitos diversos em diferentes materiais, sugerindo que o aprendizado profundo oferece uma promessa para enfrentar esses desafios complexos sem custos computacionais exorbitantes.

Sobre estudos que abordam a aplicação de diferentes técnicas, algoritmos e modelos de aprendizado específicos voltados ao desempenho prático de soluções industriais, diversos artigos foram referenciados. De modo geral, as metodologias de detecção abrangem várias categorias, como espectral, estrutural, baseada em modelo, estatística, aprendizado de máquina, e híbrida. No contexto da metodologia aplicada, destacam-se o uso de aprendizado profundo e técnicas de decomposição, como o *patching*, que consiste em dividir a imagem original em múltiplas imagens menores.

Jing et al. (2019) propõem em seu estudo um método que utiliza *patches* locais de imagem para treinar redes neurais convolucionais profundas (*Deep Convolutional Neural Networks*, DCNNs), enquanto utiliza imagens inteiras para os testes. Essa abordagem, denominada treinamento por *patches* e teste por imagem (*Patch Training Image Testing*, PTIT), aborda o desafio do desequilíbrio entre *patches* com defeito e sem defeito, decompondo a imagem original do tecido em inúmeras imagens menores, que são categorizadas manualmente. O modelo de rede, inspirado no LeNet-5, foi aprimorado com mais mapas de características e apenas duas camadas totalmente conectadas para reduzir o número de parâmetros treináveis, extraindo eficientemente características desses *patches*. Os autores reportaram uma acurácia média de 97,31% em três *datasets*. A estrutura do DCNN é apresentada na Figura 3.1.

Sandhya et al. (2021) propuseram uma estratégia em duas etapas para a detecção de defeitos em tecidos, começando com a divisão da imagem do tecido em *patches* menores e a aplicação de técnicas de pré-processamento, como escalonamento, normalização e equalização de histograma, para melhorar a qualidade dos dados. Um primeiro modelo,

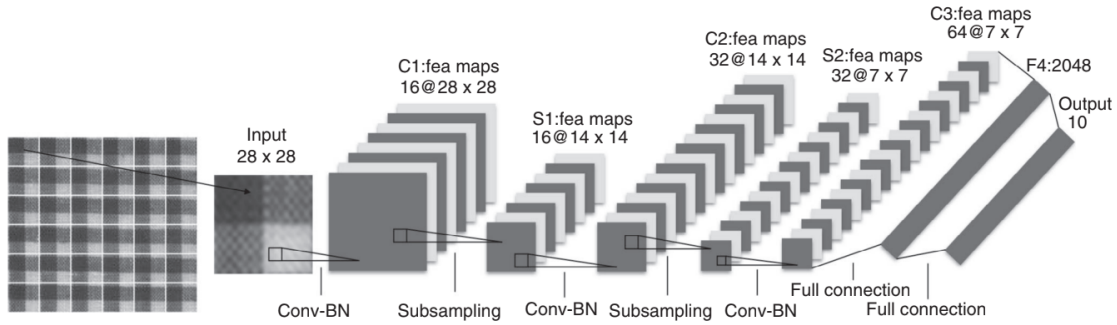

Figura 3.1: Arquitetura do DCNN por Jing et al. (2019)

baseado na rede pré-treinada AlexNet, realiza a classificação binária para identificar a presença de defeitos. Em caso positivo, um segundo modelo é ativado para categorizar o defeito em uma das cinco classes: problemas de cor, cortes, furos, falhas nos fios e contaminação por metais. A abordagem alcançou uma acurácia reportada de 92,6% no *MVTec Anomaly Detection Dataset*[2] para as cinco classes de defeitos.

No trabalho de Yu et al. (2022), foi introduzido o FA-YOLO (*Feature Augmentation* YOLO), uma modificação do modelo YOLOv4. Usando o conjunto de dados público Aliyun TIANCHI, que inclui 4371 imagens defeituosas e 4371 imagens normais com 15 tipos de defeitos, eles abordaram desafios como fundos complexos e escalas de defeitos desiguais enfrentados pelo modelo original. O FA-YOLO incorpora técnicas de *oversampling* e *copy-paste* para lidar com o desbalanceamento de classes e preservar pequenos defeitos, além de uma cabeça de detecção adicional e um *neck* modificado para uma melhor fusão de características. Utiliza ampliação de características residuais, um módulo de atenção SimAM, fusão adaptativa de características espaciais e a estrutura parcial *cross-stage* para otimizar o desempenho. Resultados experimentais mostraram que o FA-YOLO alcançou um mAP de 72,53%, superando o YOLOv4 padrão em 7,31% e reduzindo o número de parâmetros em 28,5%. Esses ganhos demonstram uma melhora na eficiência e precisão do modelo, operando a 40 FPS em uma NVIDIA GeForce RTX 2070 8GB.

Dlamini et al. (2022) apresentam uma estrutura de detecção de defeitos em tecidos projetada para detecção em tempo real *on-site*, como mostrado na Figura 3.2. Sua abordagem integra processamento de imagem convencional para aprimoramento inicial dos dados, decomposição de imagem PTIT usando DMF (*Distance Matching Function*), augmentação e escalonamento, seguido pelo treinamento de uma arquitetura YOLOv4 com pesos pré-treinados. Os autores relatam um mAP à 50% IoU de 71,6% e 34 FPS em uma GPU NVIDIA Tesla P100.

Lin et al. (2022) abordaram desafios na inspeção de qualidade de tecidos, incluindo a pequena escala dos defeitos, os aspectos de imagem desbalanceados e as baixas velocidades de detecção, considerando as limitações de implantação em dispositivos embarcados. Eles propuseram um mecanismo de autoatenção *multi-head* com janela deslizante adaptado para pequenos defeitos e também integraram o módulo *Swin Transformer* na

[2] `https://www.mvtec.com/company/research/datasets/mvtec-ad`

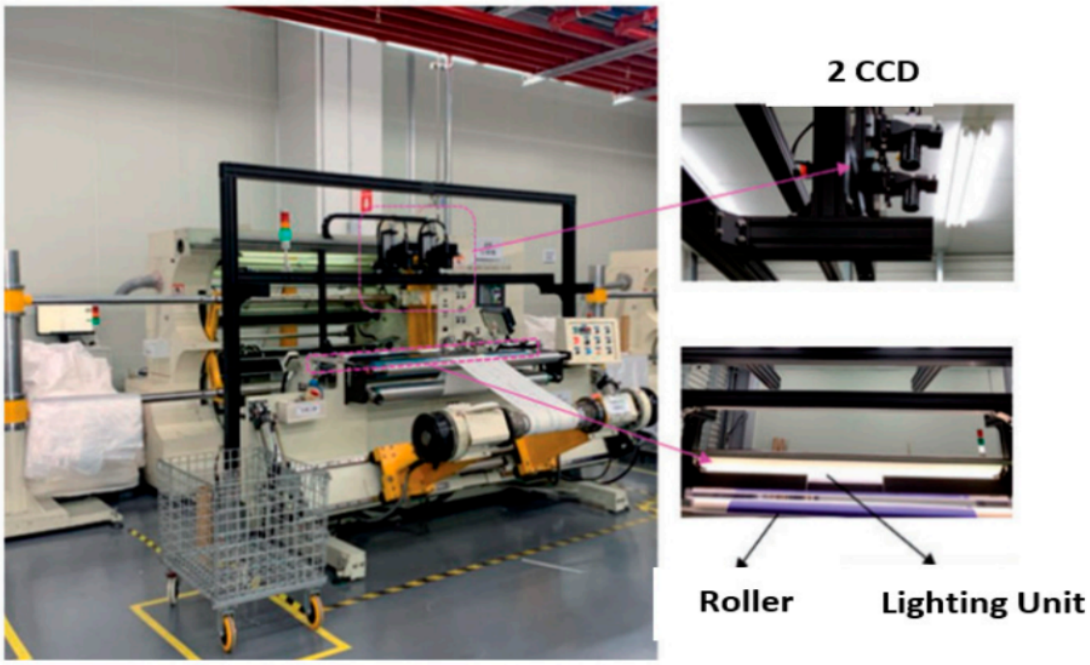

Figura 3.2: Integração de máquina de inspeção por Dlamini et al. (2022)

arquitetura YOLOv5 para extrair características hierárquicas. Além disso, implementam a função de perda focal generalizada para melhorar o aprendizado de instâncias positivas e reduzir a detecção de falhas. Sua solução alcançou 76,5% de mAP a 58,8 FPS na GPU NVIDIA GeForce RTX 3060TI. Esses resultados foram obtidos em uma base de dados proprietária de tecidos e atendem aos requisitos de detecção em tempo real para dispositivos embarcados.

Jun et al. (2021) propuseram uma estratégia de duas etapas para a detecção de defeitos em tecidos utilizando uma rede neural convolucional profunda. Inicialmente, a imagem do tecido é dividida em *patches* quadrados de tamanho fixo, que são aprimorados por meio de equalização de histograma. O modelo Inception-V1 é então utilizado para prever defeitos nessas áreas localizadas, seguido pelo modelo LeNet-5, que atua como um mecanismo de votação para reconhecer o tipo de defeito. As fases principais da estratégia são a previsão de defeitos locais e o reconhecimento global de defeitos, ilustradas na Figura 3.3. Experimentos realizados no *dataset* TILDA[3] demonstraram uma acurácia reportada de 96%. No entanto, a técnica demonstra desafios na identificação precisa de pequenos defeitos (menos de 10 *pixels*) e na distinção entre defeitos e dobras do tecido. O estudo enfatiza a necessidade de um sistema de inspeção de defeitos *online* para melhorar a precisão da detecção em tempo real e reduzir custos, composto por um sistema de *hardware* para a coleta e transmissão de imagens em tempo real, e um sistema de *software* para a identificação de defeitos e *feedback* aos operadores da produção.

Zhao et al. (2020) propuseram um método de detecção de defeitos em tecidos em

[3] https://lmb.informatik.uni-freiburg.de/resources/datasets/tilda.en.html

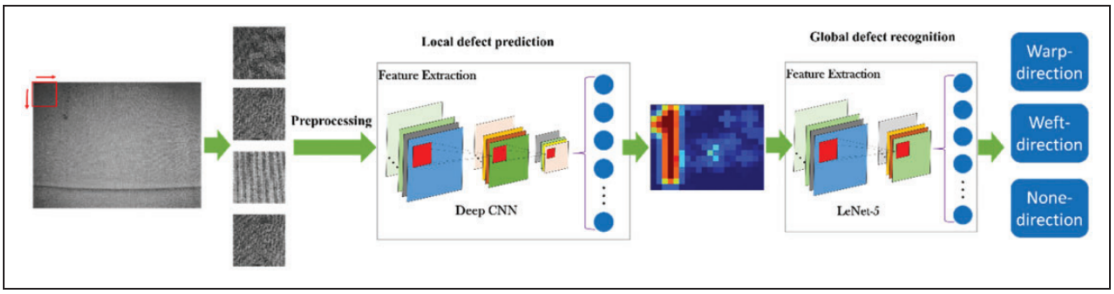

Figura 3.3: Método de detecção proposto por Jun et al. (2021)

tempo real baseado em uma rede neural convolucional de múltiplas escalas (MSCNN). A abordagem inicia com a extração de características das imagens de tecido utilizando pequenos *kernels* de convolução 3x3 para aprimorar a detecção de características locais. Os autores introduziram um método mais rápido de localização empregando *clustering* K-means para obter caixas delimitadoras de defeitos com informações de tamanho pré-conhecidas, o que melhorou a velocidade de detecção. A MSCNN proposta foi testada em defeitos típicos de tecidos de linho e tecidos padronizados, alcançando uma acurácia reportada de 93.7% para defeitos em pequena escala, 29 FPS em uma GPU GTX1060 e uma velocidade de detecção de em rolagem de 30 metros por minuto, desempenho este que atende aos requisitos industriais atuais.

Ashraf et al. (2022) propuseram uma técnica de processamento de imagens utilizando a arquitetura GoogleNet para classificar tecidos defeituosos e não defeituosos. O sistema foi treinado com vários tipos de defeitos de tecido e avaliado utilizando o conjunto de dados TILDA, alcançando uma acurácia reportada de 94,46%. Além disso, a rede proposta parece ter demonstrado bom desempenho em diferentes tipos de tecidos padronizados, distinguindo amostras defeituosas das não defeituosas de maneira consistente.

A Tabela 3.1 apresenta um resumo das metodologias práticas e das métricas reportadas nesta seção, ordenadas pelo seu nível (*Rank*) de similaridade com o presente estudo. Destacam-se os quatro primeiros, cujas metodologias são voltadas para a classificação de defeitos, em contraste com os quatro últimos, focados em detecção.

Entre os quatro primeiros, sobressaem os trabalhos de Jing et al. (2019) e Sandhya et al. (2021), que utilizam a técnica PTIT para o treinamento e a localização de defeitos por *patches* – abordagem também explorada neste estudo.

Vale ressaltar que, embora alguns desses estudos obtenham pontuações elevadas, uma comparação justa entre o desempenho de metodologias específicas só é possível se os dados utilizados forem os mesmos. Portanto, uma maior pontuação não implica, necessariamente, que uma metodologia seja superior à outra; é imprescindível realizar experimentos em bases de teste específicas que reflitam adequadamente o problema a ser solucionado.

Em capítulos subsequentes, serão exploradas as principais melhorias nas técnicas empregadas por essas pesquisas anteriores, bem como uma comparação isonômica de diversas métricas em bases de dados específicas.

Tabela 3.1: Resumo dos Estudos de Detecção e Classificação de Defeitos em Tecidos, Ordenados por Similaridade com o Presente Estudo

Rank	Estudo	Tipo de Problema	Metodologia	Pontuação	FPS
1	Jing et al. (2019)	Classificação (acurácia)	PTIT e Lenet-5 modificado	97.31%	11 (CPU), 40 (GPU)
2	Sandhya et al. (2021)	Classificação (acurácia)	Pré-processamento, PTIT e AlexNet	92.6%	-
3	Ashraf et al. (2022)	Classificação (acurácia)	Pré-processamento, GoogleNet, classificação binária	94.46%	-
4	Jun et al. (2021)	Classificação (acurácia)	Inception-VI, Lenet-5	96%	-
5	Yu et al. (2022)	Detecção (mAP)	FA-YOLO	72.53%	40 (GPU)
6	Lin et al. (2022)	Detecção (mAP)	YOLOv5, *Swin Transformer*	76.5%	58.8 (GPU)
7	Dlamini et al. (2022)	Detecção (mAP)	YOLOv4	71.6%	34 (GPU)
8	Zhao et al. (2020)	Detecção (mAP)	MSCNN	93.7%	29 (GPU)

3.3 Sistemas de Informação na Indústria Têxtil

Serão apresentados, a seguir, alguns estudos relacionados que se alinham ao objetivo específico deste trabalho de criação de uma plataforma voltada para a indústria têxtil. Esses estudos exploram, mais especificamente, conceitos de engenharia de requisitos e desenvolvimento de *software*, abrangendo desde a prototipação até a implementação em código, com foco na digitalização e otimização de processos pré-existentes. Entre os processos abordados estão a geração de relatórios, o controle geral de operações e a detecção de defeitos.

Arikan (2019) propôs um sistema para máquinas de inspeção de tecidos que substitui relatórios em papel por versões digitais, acessíveis em telas de computador e capazes de serem comparadas com relatórios anteriores. O sistema é composto por uma máquina de inspeção equipada com um *display* digital controlado por PLC e um *software* baseado na

web, projetado para realizar análises detalhadas, ilustrado na Figura 3.4. O objetivo é otimizar o tempo e reduzir a demanda por trabalho manual nas empresas têxteis, eliminando processos tradicionais de documentação e análise de defeitos.

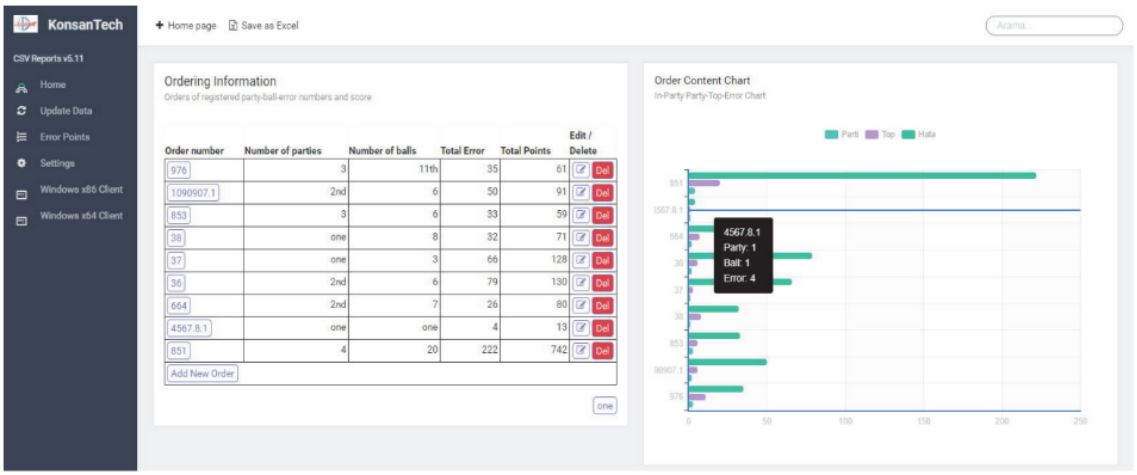

Figura 3.4: Tela do *software web* de análise e geração de relatórios por Arikan (2019)

Schneider et al. (2020) apresentaram um *software* voltado ao controle de processos na indústria de fabricação de uniformes, com foco na otimização da produção e na redução de perdas. O estudo baseou-se em uma empresa têxtil que enfrentava desafios relacionados ao controle de processos, resultando em custos adicionais de mão de obra, desperdício de material e atrasos nas entregas. A proposta buscou automatizar o controle dos processos de fabricação, visando aumentar a eficiência, minimizar perdas e melhorar a qualidade do produto final. A modelagem do *software* foi realizada utilizando a linguagem UML, e protótipos de baixa fidelidade foram desenvolvidos para validação da solução junto ao cliente.

Gonzalez et al. (2022) desenvolveram uma plataforma voltada à detecção de defeitos para controle de qualidade na indústria têxtil. A solução combina visão computacional, aprendizado profundo, geolocalização e tecnologias de comunicação para automatizar a inspeção de qualidade, superando as limitações da inspeção visual humana. O sistema é composto por módulos dedicados à aquisição de imagem, detecção e classificação de defeitos, além de um intermediário de mensagens e uma interface de visualização. Como relatado pelos autores, a plataforma é capaz de identificar defeitos com área mínima de 1 mm² a uma velocidade de 25 m/min, utilizando câmeras lineares de alta resolução e um algoritmo baseado em rede neural artificial pré-treinado. As telas de interface de *login* e detecção de defeitos desenvolvidas são ilustradas na Figura 3.5.

Souza & Campo (2023) apresentaram o desenvolvimento de um protótipo de interface para um sistema de gestão de estoque voltado para uma indústria de confecção com foco em vendas online. A proposta buscou solucionar problemas relacionados à gestão de estoque de produtos semiacabados em uma microempresa que comercializa seus produtos pela internet. O sistema foi projetado em arquitetura *web*, utilizando tecnologias como JavaScript, React e MySQL. O trabalho incluiu o levantamento de requisitos, a modelagem

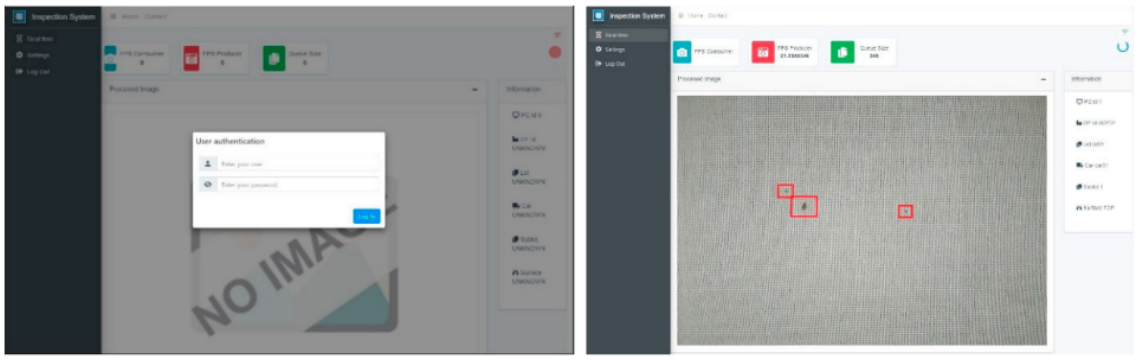

Figura 3.5: Componente de Interface Visual por Gonzalez et al. (2022). Lado esquerdo: interface de *login*, lado direito: detecção de defeitos em tempo real

do sistema com o uso de UML e a prototipagem da interface. Segundo os autores, a solução contribui para otimizar os processos de produção, reduzir desperdícios e aumentar a eficiência operacional da empresa na qual foi implementada.

A Tabela 3.2 apresenta uma comparação entre estudos focados na criação de plataformas para a digitalização e otimização de processos na indústria têxtil. Foram analisados os seguintes critérios: modelagem do *software*, presença de módulos de geração de relatórios digitais, aplicação de técnicas de inspeção automatizada para IQ e gestão de estoque.

Tabela 3.2: Comparação entre Estudos Relacionados à Digitalização e Otimização na Indústria Têxtil

Estudo	**Modelagem**	**Relatórios**	**Inspeção**	**Estoque**
Arikan (2019)	✗	✓	✗	✗
Schneider et al. (2020)	✓	✗	✗	✗
Gonzalez et al. (2022)	✓	✗	✓	✗
Souza & Campo (2023)	✓	✗	✗	✓
Nosso Estudo	✓	✓	✓	✗

Observa-se que, com exceção de Arikan (2019), todos os estudos incluem algum nível de modelagem do *software*. A geração de relatórios digitais é abordada apenas por Arikan (2019) e na proposta deste trabalho. Quanto à inspeção automatizada de qualidade, que emprega técnicas como visão computacional e aprendizado profundo, ela está presente exclusivamente no estudo de Gonzalez et al. (2022) e na abordagem proposta neste estudo. Por fim, a gestão de estoque foi tratada apenas por Souza & Campo (2023), diferenciando-se dos demais.

Dessa forma, o nosso estudo se destaca ao integrar a modelagem do *software*, a automação na IQ e a geração de relatórios digitais, oferecendo uma solução integrada e mais completa para a otimização desses processos.

Capítulo 4

Metodologia

4.1 Ferramentas Utilizadas

Esta seção descreve as principais ferramentas e configurações utilizadas para implementar e executar o experimento, garantindo a integridade e a reprodutibilidade dos resultados. Embora esta lista não seja exaustiva, ela destaca as ferramentas e bibliotecas mais relevantes. Para uma visão mais completa, acesse o repositório do projeto no GitHub[1].

Plataformas e Frameworks

- **Angular v18**: Utilizado como *framework frontend* para o desenvolvimento tanto das aplicações operacionais quanto das aplicações de ciência de dados. *Angular* foi escolhido por sua capacidade e versatilidade na criação de interfaces dinâmicas e responsivas, facilitando a interação do usuário com o sistema. Este *framework* é particularmente adequado para aplicações de nível empresarial devido à sua consistência e abordagem "*batteries included*", onde muitas tecnologias necessárias estão integradas e prontas para uso. Isso permite uma implementação mais uniforme e uma curva de aprendizado previsível, o que é benéfico em projetos de longo prazo com equipes que podem mudar ao longo do tempo. A documentação ampla e a consistência na forma de implementar soluções em *Angular* garantem que novos membros da equipe possam se integrar mais rapidamente em comparação a outras opções, como *React*.
  No entanto, essa escolha também apresenta alguns desafios. A princípio, a curva de aprendizado pode ser mais acentuada, resultando em um tempo de desenvolvimento inicial maior. Além disso, a abordagem opinativa do *Angular*, que tende a definir "a maneira correta" de implementar certas funcionalidades, como a gestão de estado ou a estruturação de componentes, pode limitar a flexibilidade e tornar o desenvolvimento menos ágil em cenários particulares.
- **MLflow 2.15.0**: Plataforma de código aberto utilizada como aplicação e *backend* para rastreamento de experimentos de modelos. O *MLflow* foi projetado para simplificar o ciclo de vida do aprendizado de máquina, oferecendo um conjunto abrangente de ferramentas e *frameworks* para gerenciar e rastrear o processo de desen-

[1] `https://github.com/angelolmg/textile-defect-detection-app`

volvimento de ML de ponta a ponta. Ele permite capturar cada "execução" de um experimento de treinamento, registrando o desempenho do modelo e os parâmetros utilizados, além de possibilitar visualizações comparativas entre diferentes execuções. Além disso, permite o gerenciamento de artefatos, como os modelos salvos, que podem ser utilizados em ambientes de implementação.
Operando como um sistema de controle de versão para modelos, o *MLflow* possibilita o rastreamento e a gestão eficiente dos parâmetros e características que influenciam o desempenho dos modelos, com mínimo esforço adicional. Isso permite que cientistas de dados e engenheiros de ML direcionem seus esforços para o desenvolvimento e a implantação dos modelos, garantindo ao mesmo tempo o controle e a reprodutibilidade em todas as fases do processo.

- **Flask e SQLite**: *Flask*, em conjunto com *SQLite*, foram utilizado como *backend* e banco de dados na criação de todos os serviços que compõem o sistema de detecção de defeitos. Esses *frameworks* foram escolhidos por sua flexibilidade e abordagem minimalista, o que acelerou o desenvolvimento dos serviços implementados, que, no contexto da PoC da plataforma, possuem baixa complexidade. Especificamente, foram empregados na implementação do componente de dados, do componente de sessão, e do serviço de treinamento de modelos dentro do componente de processos, a ser introduzido na Seção 4.3. Além disso, o *Flask* foi utilizado para implementar a lógica do componente de inteligência, incluindo o consumo de *feeds* de vídeo armazenados em disco, além do serviço de tratamento de exceções. No entanto, para aplicações de nível empresarial, pode ser necessário utilizar um *framework* mais opinado, como o *Django*, para serviços que exigem funcionalidades mais robustas de autenticação e autorização. O *Django* oferece uma série de recursos integrados para essas necessidades e conta com uma comunidade de suporte ativa, o que pode facilitar a implementação e manutenção de serviços complexos.
- **Ultralytics YOLO**[2]: *Framework* desenvolvido pela Ultralytics que oferece suporte a modelos da família YOLO, como as versões v5 e v8, amplamente utilizados em tarefas de classificação e detecção. Projetado para ser intuitivo e versátil, o *framework* também suporta funcionalidades como segmentação de imagens e detecção de poses, adaptando-se às necessidades específicas de diferentes aplicações. Sua integração com a plataforma completa implementada e seu desempenho superior serviram para assegurar a eficácia do sistema de detecção de defeitos desenvolvido.
- **OpenCV**: Biblioteca de código aberto para processamento de imagens e visão computacional, escolhida por seus algoritmos eficientes e pela variedade de ferramentas que oferece para trabalhar com imagens e vídeos. Sua flexibilidade e amplo suporte a diferentes plataformas fazem dela uma escolha sólida e confiável para implementar sistemas de CV modernos.

Comparativo entre Modelos Utilizados

Para os experimentos da metodologia aplicada, foram selecionados alguns modelos para comparação, incluindo aqueles propostos por Jing et al. (2019) e Sandhya et al.

[2] `https://github.com/ultralytics/ultralytics`

(2021), além das versões menores (*nano, small e medium*) das famílias YOLOv5 e YOLOv8 da Ultralytics, conforme mencionado no tópico anterior. A escolha desses modelos se deve ao fato de que, no momento do desenvolvimento deste trabalho, representam o estado da arte em CNNs voltadas para tarefas de visão computacional. Além disso, possuem pesos pré-treinados disponibilizados pelos desenvolvedores, o que facilita o processo de *transfer learning*. Os modelos menores foram priorizados, pois, para aplicações de detecção em tempo real, oferecem um bom equilíbrio entre velocidade e acurácia.

A Tabela 4.1 apresenta uma comparação entre algumas características e métricas (acurácia top 1, acurácia top 5, tempo para uma inferência em uma GPU NVIDIA A100 e nº de parâmetros) relevantes dos modelos utilizados, pré-refinamento. Especificamente, os modelos de classificação YOLO foram treinados no conjunto de dados *ImageNet* ao longo de 90 épocas, utilizando o otimizador SGD, com taxa de aprendizado inicial (*lr0*) de 0.001 e *weight decay* de 5×10^{-5}, em imagens de tamanho 224, mantendo as configurações padrão.

Tabela 4.1: Comparação entre Modelos Utilizados

Modelo	**acc top1 (%)**	**acc top5 (%)**	**inf A100 (ms)**	**params (M)**
Jing et al.	-	-	-	6.46
Sandhya et al.	-	-	-	54.56
YOLOv5n	64.6	85.4	0.5	**2.5**
YOLOv5s	71.5	90.2	0.6	5.4
YOLOv5m	75.9	92.9	0.9	12.9
YOLOv8n	69.0	88.3	**0.31**	2.7
YOLOv8s	73.8	91.7	0.35	6.4
YOLOv8m	**76.8**	**93.5**	0.62	17

Os modelos propostos por Jing et al. (2019) e Sandhya et al. (2021) não foram pré-treinados com o auxílio da base *ImageNet*, portanto, não há dados de desempenho baseados nesse conjunto. Dito isso, métricas relevantes sobre todos os modelos são apresentadas após os experimentos na Tabela 5.2.

Parâmetros de Configuração dos Experimentos

A Tabela 4.2 apresenta os principais parâmetros de configuração adotados no ambiente de desenvolvimento e execução do experimento. Para a realização dos testes, utilizou-se o ambiente padrão do Kaggle[3], com a configuração adicional de aceleração acessada em "Configurações" » "Acelerador" » "NVIDIA Tesla P100".

[3] `https://www.kaggle.com/`

Tabela 4.2: Parâmetros de Configuração do Experimento

Dispositivo	**Configuração**
Sistema Operacional	Debian 4.9.0-5 x86_64
Processador	Intel(R) Xeon(R) CPU @ 2.00GHz
Acelerador GPU	CUDA 12.4
GPU	NVIDIA Tesla P100 16GB
Linguagem	Python 3.10

4.2 Algoritmos Desenvolvidos

Dois algoritmos principais foram desenvolvidos para o sistema de detecção: a **detecção em grelha** e o **refinamento em *ripple***. Ambos desempenham papéis complementares e serão detalhados a seguir.

Detecção em Grelha

A técnica de detecção em grelha divide a imagem de entrada em retalhos quadrados uniformes, e cada retalho é classificado em uma das seguintes categorias: *['good', 'hole', 'objects', 'oil spot', 'thread error']*. A classe padrão é *'good'*, sendo alterada apenas se o classificador retornar outra categoria com uma confiança superior ou igual a um limiar definido, τ. Caso isso ocorra, o retalho é rotulado com a nova classe identificada. Essa segmentação local permite a identificação precisa de defeitos, que são posteriormente utilizados para alertas e visualizações.

Na Figura 4.1, os resultados do algoritmo são apresentados com um limiar de confiança de $\tau = 0.99$, demonstrando a eficácia da abordagem, especialmente em casos de defeitos evidentes.

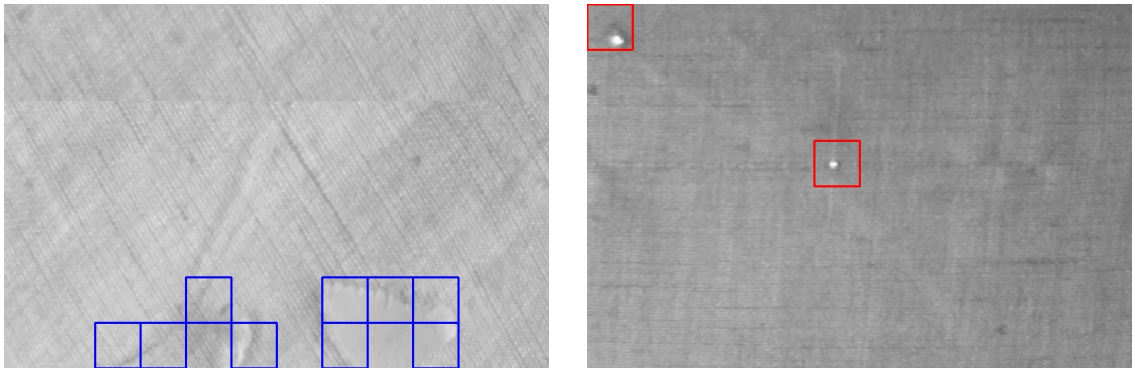

Figura 4.1: Exemplos de saídas do algoritmo de detecção em grelha ($\tau = 0.99$) evidenciando defeitos do tipo objeto (esquerda) e furo (direita)

Além disso, gráficos de dispersão podem ser utilizados para uma visualização rápida

da localização dos defeitos, evidenciando não apenas a posição relativa na imagem, mas também em relação a todas as capturas da sessão. Esses gráficos também permitem a classificação dos defeitos, como ilustrado na Figura 4.2.

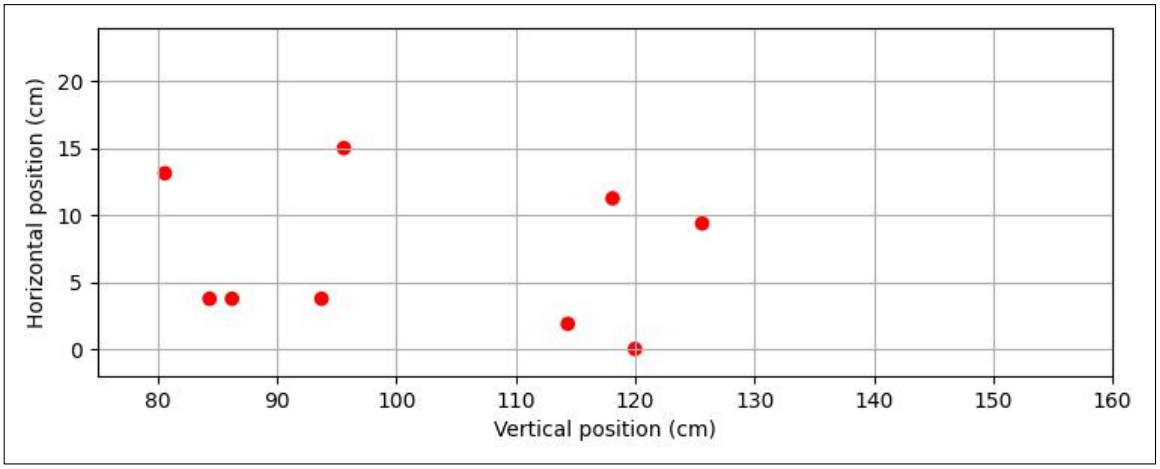

Figura 4.2: Grafico de dispersão de defeitos

O Algoritmo 1 detalha o funcionamento deste processo.

Refinamento em *Ripple*

A técnica de refinamento em *ripple* complementa o algoritmo previamente descrito ao adicionar uma segunda rodada de detecções para os vizinhos do retalho primário. Esta etapa utiliza um segundo limiar de confiança τ_2, onde $\tau_2 < \tau_1$, buscando associação da vizinhança ao rótulo primário. O processo completo, incluindo o refinamento, é ilustrado na Figura 4.3.

O refinamento pós-detecção, que utiliza o contexto espacial local, baseia-se na suposição de que, se uma célula apresenta defeito, é provável que suas células vizinhas também estejam comprometidas. Assim, o algoritmo busca defeitos nas células vizinhas com uma taxa de confiança menor, repetindo esse processo de *rippling* várias vezes, cada vez com limiares de confiança progressivamente menores, para melhorar a detecção. No caso descrito, o *rippling* é aplicado uma única vez.

O Algoritmo 2 detalha o processo ilustrado. Destaca-se que a principal diferença em relação ao Algoritmo 1 é a adição de um terceiro laço na segunda posição externa, que é responsável pelo refinamento da primeira etapa.

A Figura 4.4 demonstra alguns resultados de detecções após a aplicação do segundo passo de refinamento, sendo $\tau_1 = 0.99$ e $\tau_2 = 0.5$.

4.3 Modelagem da Prova de Conceito

Nesta seção, propõe-se um esboço de arquitetura de referência para aplicações baseadas em IoT na indústria têxtil. A motivação para essa proposta é fornecer um direci-

Algoritmo 1: Algoritmo para detecção de defeitos em grelha com CNN

Entrada: I: Imagem de entrada com tamanho arbitrário; M, N: dimensões da imagem redimensionada; n: tamanho dos retalhos quadrados; C: rede neural convolucional (CNN) treinada para classificação de defeitos; τ: limiar de confiança

Saída: $I_{rotulada}$: Imagem com defeitos destacados por bordas coloridas em retalhos classificados como defeituosos

$I_{redimensionada} \leftarrow$ `Redimensionar`(I, M, N);
$indices \leftarrow []$;
$rotulos \leftarrow []$;
para $i = 1$ **até** M/n **faça**
    para $j = 1$ **até** N/n **faça**
        $retalho \leftarrow I_{redimensionada}[i \cdot n : (i+1) \cdot n, j \cdot n : (j+1) \cdot n]$;
        $indices \leftarrow indices \cup (i, j)$;
        $(r,$ `conf`$) \leftarrow C(retalho)$;
        se `conf` $\geq \tau$ **então**
            $rotulos \leftarrow rotulos \cup r$;
        fim
        senão
            $rotulos \leftarrow rotulos \cup \,'good'$;
        fim
    fim
fim
para $k = 1$ **até** `Comprimento`$(rotulos)$ **faça**
    se $rotulos[k] \neq \,'good'$ **então**
        $(x, y) \leftarrow indices[k]$;
        `DesenharBorda`$(I_{redimensionada}, x, y,$ `cor`$(rotulos[k]))$;
    fim
fim
retorna $I_{rotulada}$

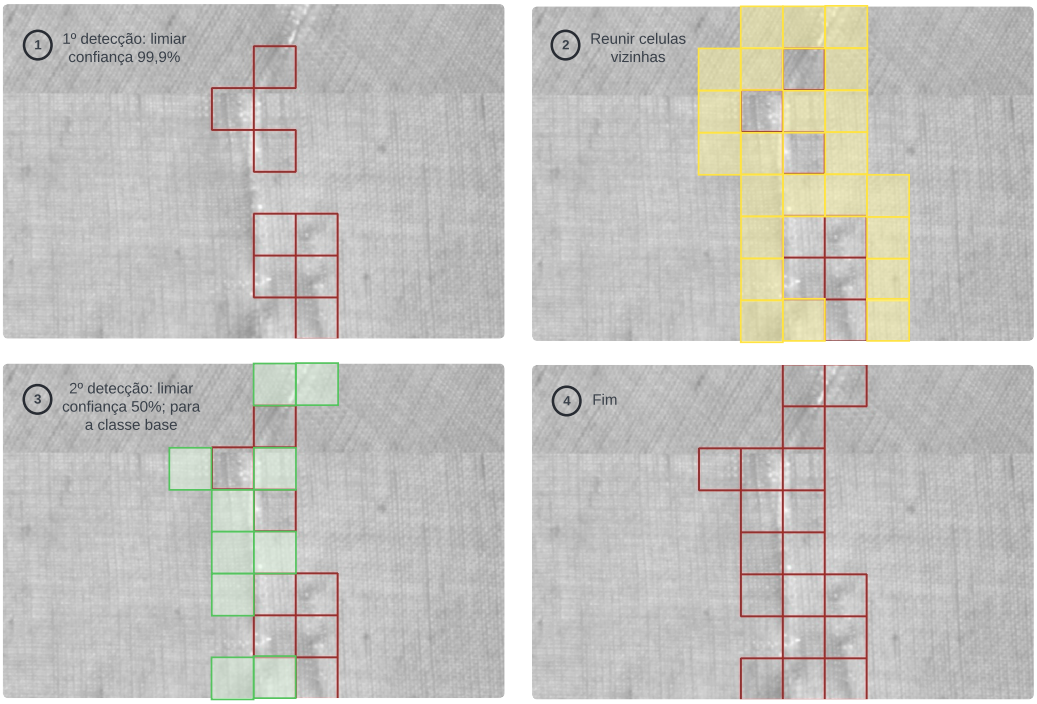

Figura 4.3: Descrição da Detecção com Refinamento em *Ripple*

Algoritmo 2: Algoritmo para detecção de defeitos em grelha com refinamento em *ripple*

Entrada: I: Imagem de entrada com tamanho arbitrário; M, N: dimensões da imagem redimensionada; n: tamanho dos retalhos quadrados; C: rede neural convolucional (CNN) treinada para classificação de defeitos; τ_1: limiar de confiança inicial; τ_2: limiar de confiança para detecção de vizinhos

Saída: $I_{rotulada}$: Imagem com defeitos destacados por bordas coloridas em retalhos classificados como defeituosos

$I_{redimensionada} \leftarrow$ `Redimensionar`(I, M, N);
$indices \leftarrow []$;
$rotulos \leftarrow []$;
para $i = 1$ **até** M/n **faça**
  para $j = 1$ **até** N/n **faça**
    $retalho \leftarrow I_{redimensionada}[i \cdot n : (i+1) \cdot n, j \cdot n : (j+1) \cdot n]$;
    $indices \leftarrow indices \cup (i, j)$;
    $(r,$ `conf`$) \leftarrow C(retalho)$;
    se `conf` $\geq \tau_1$ **então**
      $rotulos \leftarrow rotulos \cup r$;
    senão
      $rotulos \leftarrow rotulos \cup \,'good'$;
    fim
  fim
fim
para $k = 1$ **até** `Comprimento`$(rotulos)$ **faça**
  se $rotulos[k] \neq \,'good'$ **então**
    $(x, y) \leftarrow indices[k]$;
    para cada $(x_v, y_v) \in$ `Vizinhos`(x, y) **faça**
      se (x_v, y_v) *está dentro dos limites de* $I_{redimensionada}$ **então**
        $retalho_v \leftarrow I_{redimensionada}[x_v \cdot n : (x_v+1) \cdot n, y_v \cdot n : (y_v+1) \cdot n]$;
        $(r_v,$ `conf`$_v) \leftarrow C(retalho_v)$;
        se `conf`$_v \geq \tau_2$ *e* $r_v = rotulos[k]$ **então**
          $rotulos[$`indice`$(x_v, y_v)] \leftarrow r_v$;
        fim
      fim
    fim
  fim
fim
para $k = 1$ **até** `Comprimento`$(rotulos)$ **faça**
  se $rotulos[k] \neq' good'$ **então**
    $(x, y) \leftarrow indices[k]$;
    `DesenharBorda`$(I_{redimensionada}, x, y,$ `cor`$(rotulos[k]))$;
  fim
fim
retorna $I_{rotulada}$

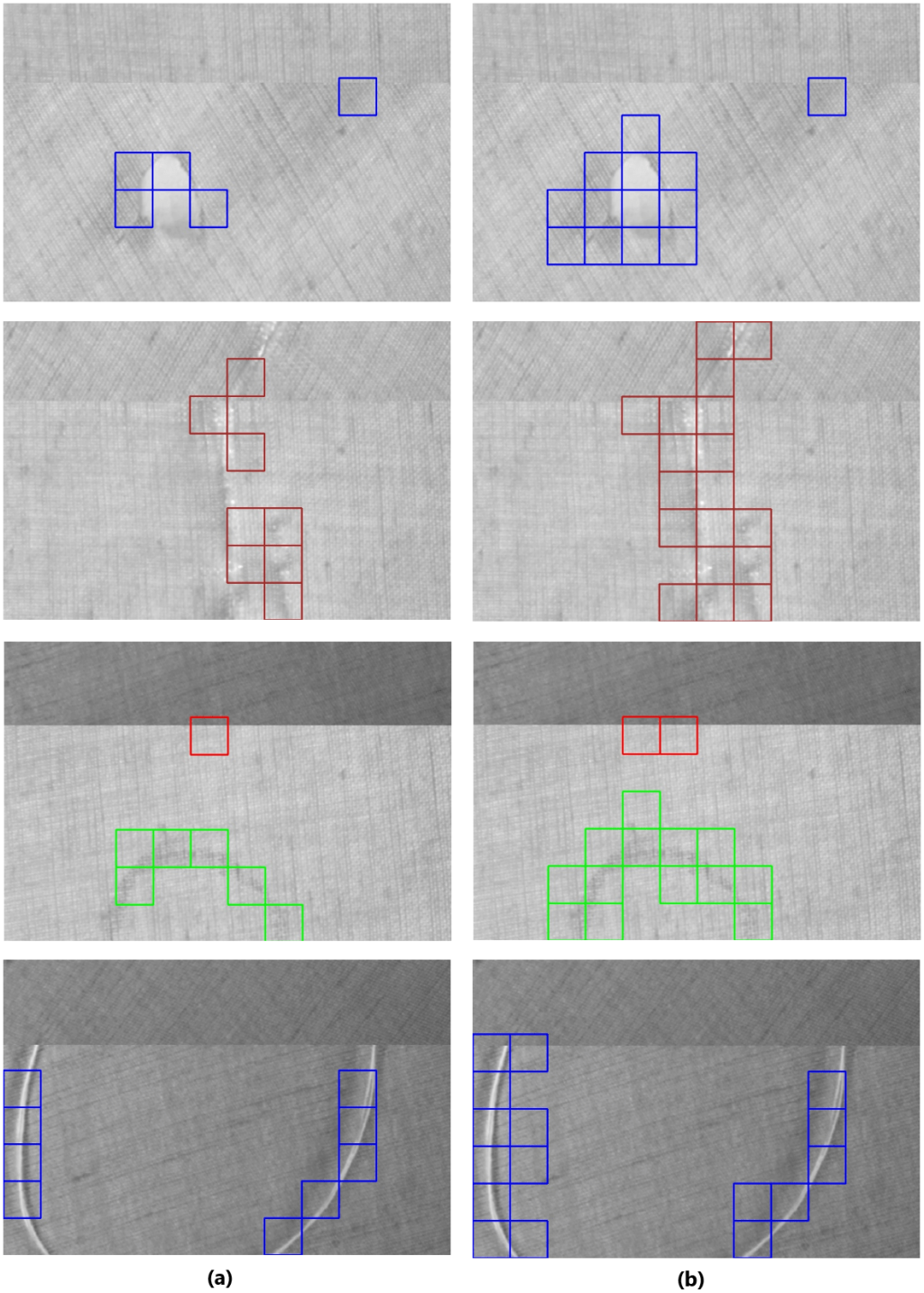

Figura 4.4: (a) Primeira etapa com $\tau_1 = 0.99$; (b) Segunda etapa com $\tau_2 = 0.5$

onamento inicial sobre como sistemas e aplicações voltados para esse setor podem ser estruturados de maneira genérica, servindo como base para a modelagem da plataforma detalhada na Subseção 4.3.1.

Como discutido na Subseção 2.2.1, uma arquitetura de referência funciona como um guia genérico para o desenvolvimento de sistemas alinhados às metas organizacionais de uma empresa, pré-definindo qualidades e funcionalidades desejadas por todos os *stakeholders*. Ela estabelece uma ponte entre os objetivos abstratos e o sistema final, permitindo que arquiteturas sejam projetadas, analisadas, documentadas e implementadas de forma controlada e gerenciada (Bass et al. 2012, de Morais Barroca Filho 2019).

No entanto, é importante destacar que a arquitetura apresentada possui limitações. Uma verdadeira arquitetura de referência demanda uma revisão sistemática abrangente sobre as características de aplicações dentro de determinado domínio, seguida da identificação de requisitos comuns que possam ser instanciados em arquiteturas concretas para atender a necessidades específicas. Sendo assim, um formalismo mais rigoroso seria necessário para caracterizar adequadamente essa arquitetura como uma referência consolidada. Contudo, dada a limitação de tempo para a realização deste estudo mais aprofundado, optou-se por apresentar um modelo que, embora ainda não consolidado como arquitetura de referência, demonstra aplicabilidade e pode servir como ponto de partida para trabalhos futuros.

O referencial apresentado nesta seção é baseado na *Reference Architecture for IoT-based Healthcare Applications* (RAH), de de Morais Barroca Filho (2019), adaptado ao contexto da indústria têxtil com modificações específicas para atender às demandas do setor. O diagrama da arquitetura projetada é apresentado na Figura 4.5. Embora não constitua uma arquitetura de referência formal, este modelo orienta a modelagem da plataforma de IQ, abordada na Subseção 4.3.1, e seu posterior desenvolvimento.

Na Seção A.1, são apresentadas e detalhadas as definições e funções de cada uma das camadas, componentes e serviços da arquitetura ilustrada, ressaltando-se que sua consolidação como referência formal pode ser objeto de estudos futuros.

4.3.1 Arquitetura de Software Concreta

Nesta subseção, detalha-se a lógica por trás da modelagem da plataforma proposta, denominada Plataforma de Inspeção de Qualidade em Tecidos. Essa plataforma foi concebida como uma PoC, com o objetivo específico de confirmar a viabilidade e a factibilidade da solução apresentada. Para isso, seguem-se os passos padrão da engenharia de software para desenvolvimento de sistemas, desde o levantamento de informações sobre o processo operacional até a modelagem do software utilizando elementos visuais, como diagramas UML.

Embora, idealmente, o processo de elicitação de informações incluísse procedimentos como observação em campo na planta operacional ou entrevistas com o pessoal envolvido para identificar dificuldades, melhorias desejadas e funcionalidades principais, neste trabalho essa etapa foi abstraída. Partiu-se diretamente para o levantamento de requisitos. Essa abordagem foi adotada porque a plataforma, embora fundamental, não é o foco principal desta dissertação. Em vez disso, o *software* desenvolvido atua como uma

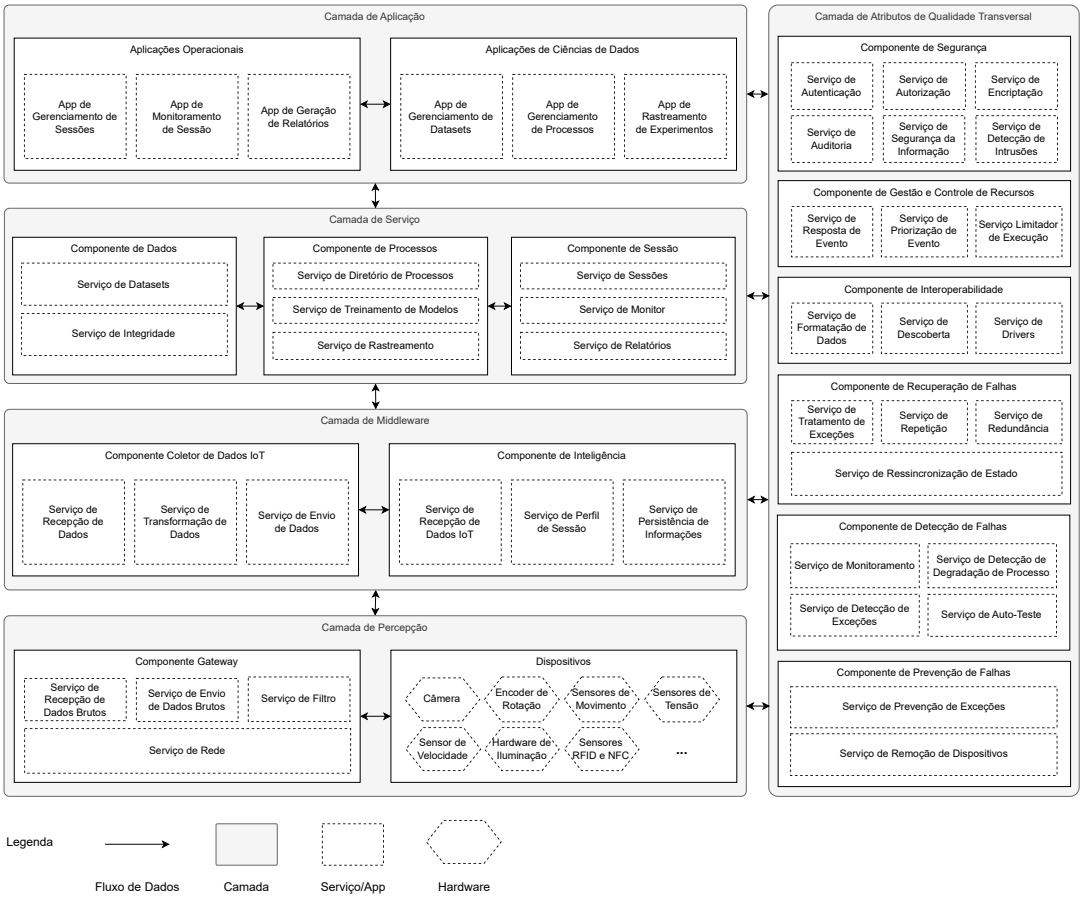

Figura 4.5: Arquitetura de Referência Para Aplicações Inteligentes baseadas em IoT na Indústria Têxtil

ferramenta para demonstrar a viabilidade de uma solução que integra as metodologias propostas, como os algoritmos e modelos de classificação baseados em aprendizado de máquina apresentados.

O objetivo é modelar uma plataforma de IQ baseada em aprendizado profundo que, no mínimo, aborde as questões apontadas por Yu et al. (2022) sobre os aspectos que um método eficaz de detecção de defeitos em tecidos deve considerar:

1. **Desbalanceamento de classes:** O número desigual de amostras de diferentes classes pode dificultar o treinamento de modelos de detecção.
2. **Perda de pequenos defeitos causada pelo escalonamento da imagem:** Pequenos defeitos podem ser ignorados ou distorcidos quando as imagens são redimensionadas.
3. **Sensibilidade ao contexto global:** A capacidade do modelo de detectar defeitos em variados tipos de fundos e em situações onde a distinção entre defeitos e o tecido é sutil.
4. **Preservação da acurácia:** O modelo não deve ser simplificado a ponto de comprometer significativamente a precisão.

A implementação aborda esses desafios por meio de diversas técnicas, incluindo os algoritmos de detecção em grelha e refinamento em *ripple*, que são detalhados na Seção 4.2 sobre algoritmos desenvolvidos. Esses algoritmos oferecem uma solução para o desbalanceamento e a perda de pequenos defeitos, permitindo uma classificação local sem perdas significativas devido ao escalonamento.

Como uma plataforma, o *software* deve ser concebido como um sistema extensível, que fornece funcionalidades centrais compartilhadas por aplicações complementares que interagem com ele. Essas aplicações utilizam as interfaces fornecidas pela plataforma para garantir uma interoperabilidade eficaz (Tiwana 2013). Para atender às demandas de um sistema completo de automação do processo de IQ, é imprescindível que a plataforma incorpore alguns elementos funcionais essenciais.

Por se tratar de um sistema de inspeção visual, a plataforma deve ser capaz de **suportar fluxos de vídeo** em tempo real, inicialmente de arquivos locais preexistentes. Adicionalmente, ela deve incluir mecanismos para o **monitoramento de sessões de inspeção**, oferecer funcionalidades para o **gerenciamento dos modelos** responsáveis pela detecção de defeitos, e possibilitar o **treinamento de novos modelos**. Para suportar esses processos, a plataforma precisa também de ferramentas para o **controle dos dados de treinamento** e o **rastreamento de experimentos** realizados, garantindo um ambiente integrado e eficiente para o ciclo de inspeção e melhoria contínua.

A implementação também deve permitir a priorização de diferentes tipos de defeitos e tecidos conforme as necessidades operacionais específicas. Ela deve incentivar o treinamento e uso de múltiplos modelos especializados para diferentes contextos, abordando as questões de sensibilidade ao contexto global e preservação da acurácia. Essa abordagem também possibilita que os engenheiros responsáveis experimentem diferentes modelos e versões de *datasets*, comparando-os de forma padronizada e eficiente. Ferramentas específicas de rastreamento, como *Weights & Biases*, *ZenML* e *MLFlow*, podem ser utilizadas

para suportar essa funcionalidade. Na Seção 4.1 foi detalhada a ferramenta *MLFlow*, utilizada na implementação da PoC.

Dito isso, o detalhamento dos requisitos funcionais identificados para a plataforma estão apresentados na Tabela 4.3. Esses requisitos também foram associados aos componentes e serviços da RA proposta e introduzida na Seção 4.3, de modo a viabilizar o instanciamento do *software* concreto.

Tabela 4.3: Requisitos Funcionais da Arquitetura de *Software* e Componentes e Serviços da Arquitetura de Referência que os Atendem

ID	**Requisitos Funcionais**	**Componentes e Serviços**
RF01	Consumir fluxo de imagens em tempo real para processamento	Componente de Inteligência: Serviço de Perfil de Sessão
RF02	Registrar cada sessão de inspeção, incluindo imagens processadas e métricas de desempenho	Componente de Sessão: Serviço de Sessões
RF03	Exibir visualmente os resultados em tempo real da detecção de defeitos, destacando áreas com problemas e categorizando o tipo de defeito	Componente de Sessão: Serviço de Monitor
RF04	Facilitar a criação de *datasets* customizados com suporte para inclusão, exclusão e rotulagem de imagens	Componente de Dados: Serviço de *Datasets*
RF05	Permitir treinamento de modelos de aprendizado de máquina por meio de dados adquiridos e/ou conjuntos de dados personalizados	Componente de Processos: Serviço de Treinamento de Modelos
RF06	Armazenar versões de modelos treinados e facilitar a comparação entre eles por métricas, como precisão e tempo de inferência	Componente de Processos: Diretório de Modelos
RF07	Controlar o rastreamento dos experimentos de treinamento, registrando métricas, como acurácia, *recall*, curva de perda, entre outros	Componente de Processos: Serviço de Rastreamento

Para atender a esses requisitos funcionais, foram especificados 24 casos de uso, detalhados na Seção A.2. Esses casos abrangem as funcionalidades dos três principais sistemas que compõem o *backend* da plataforma: o gerenciador de sessões, que contempla RF01, RF02 e RF03 (Figura A.1 e Tabela A.6); o gerenciador de *datasets*, que contempla RF04 (Figura A.2 e Tabela A.7); e o gerenciador de processos, que contempla RF05,

RF06 e RF07 (Figura A.3 e Tabela A.8).

Os requisitos não funcionais (RNFs) identificados são apresentados na Tabela 4.4, estando também associados aos componentes e serviços da arquitetura proposta.

Com relação aos RNFs e à implementação da PoC, conforme detalhado na Seção 5.2, nenhum RNF foi implementado nesta fase inicial. No entanto, sua implementação está prevista para trabalhos futuros, visando aprimorar a disponibilidade e a segurança da plataforma.

Tabela 4.4: Requisitos Não Funcionais da Arquitetura de *Software* e Componentes e Serviços da Arquitetura de Referência que os Atendem

ID	**Requisitos Não Funcionais**	**Componentes e Serviços**
RNF01	Permitir apenas que usuários autenticados tenham acesso à plataforma (Segurança)	Componente de Segurança: Serviço de Autenticação
RNF02	Gerar um sinal de alerta caso seja detectada degradação de algum processo funcional, como performance de algum modelo específico, ao longo de múltiplas sessões, afetando sua consistência (Disponibilidade)	Componente de Detecção de Falhas: Serviço de Detecção de Degradação do Modelo
RNF03	Detectar e tratar exceções durante o processamento de dados ou treinamento de modelos, como ausência de resposta de componente ou serviço ou perda de conexão com o banco de dados (Disponibilidade)	Componente de Recuperação de Falhas: Serviço de Tratamento de Exceções

Na arquitetura de referência apresentada na Figura 4.6, os elementos relevantes aos requisitos identificados estão destacados em vermelho. Dentre os componentes ressaltados, apenas os que compõem a Camada de Serviço e a Camada de Aplicação foram efetivamente implementados na PoC, conforme descrito na Seção 5.2.

Essa decisão se deve à indisponibilidade de acesso, durante a elaboração desta dissertação, às dependências operacionais e aos equipamentos mencionados. Consequentemente, todos os testes foram realizados localmente, utilizando arquivos de *feeds* de vídeo de inspeção simulados.

Na sequência, apresenta-se o diagrama de componentes da plataforma proposta. Como ilustrado na Figura 4.7, o diagrama é instanciado a partir dos blocos relevantes identificados na arquitetura de referência, considerando os requisitos levantados. Esses blocos são detalhados individualmente na Seção A.1. Conforme discutido na Subseção 2.2.3, a principal função desse diagrama é representar a comunicação entre os componentes e especificar as interfaces por meio das quais eles interagem.

Conforme mencionado anteriormente, devido à indisponibilidade de acesso a equipamentos e dependências operacionais, alguns componentes não foram implementados na

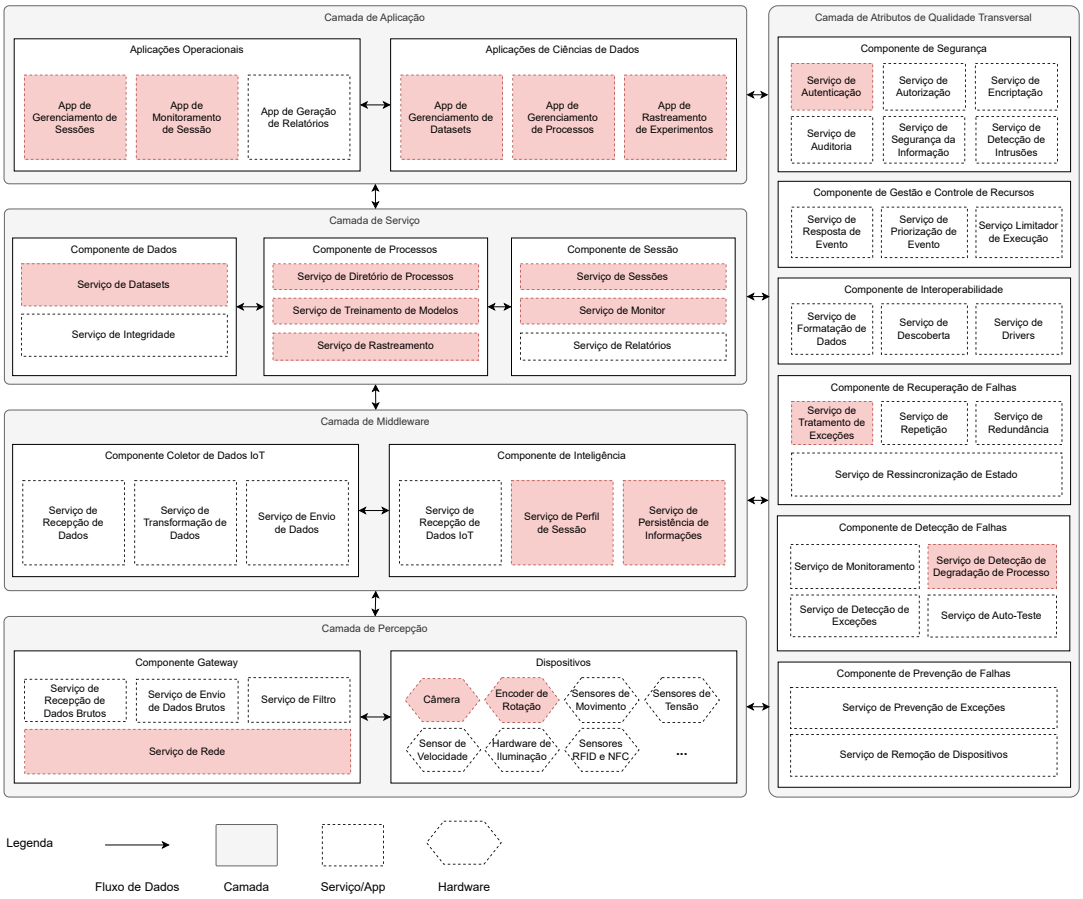

Figura 4.6: Arquitetura de Referência com elementos destacados para Plataforma de Inspeção de Qualidade em Tecidos

plataforma PoC. Em especial, o Componente de Inteligência, o Componente de Gateway e os Dispositivos não foram codificados.

A implementação desses elementos está prevista para futuras etapas, quando se dispor dos recursos necessários para testes completos em um ambiente operacional real.

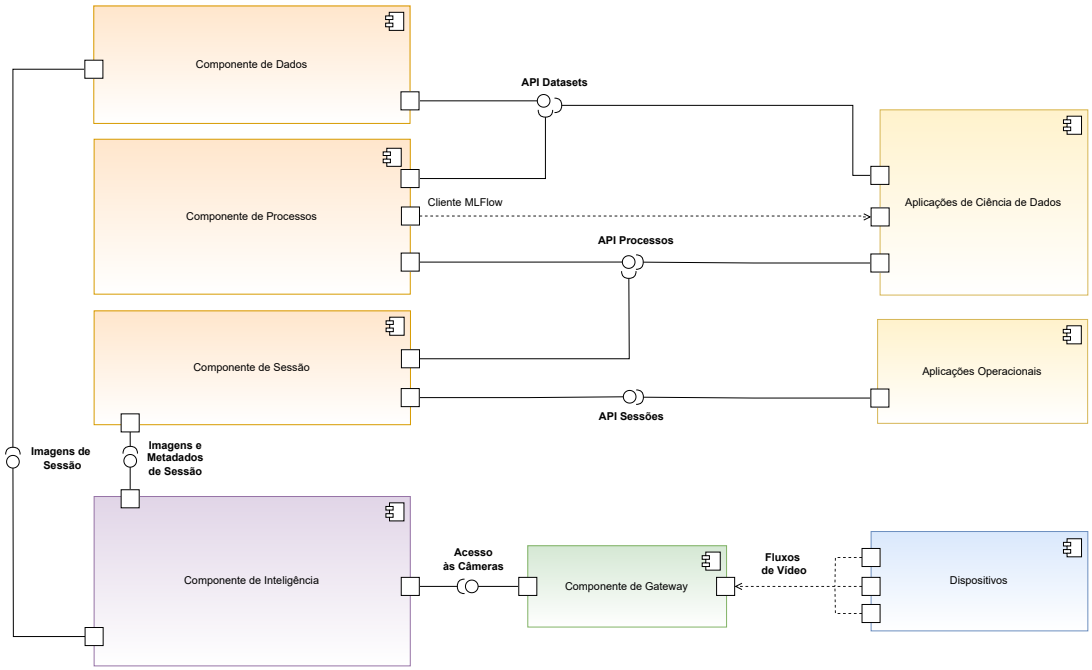

Figura 4.7: Diagrama de Componentes: Plataforma como Instância de Componentes da Arquitetura de Referência

Dito isso, conforme ilustrado na Figura 4.8, o fluxo de dados durante uma operação *online* tem origem nos dispositivos. As câmeras instaladas coletam essas informações, que ficam disponíveis por meio do serviço de rede do Componente *Gateway*. Esse serviço atribui a cada dispositivo um IP público e uma porta específica, utilizando a técnica de *port forwarding*.

O componente de inteligência, conforme ilustrado na Figura 4.9, inicia a coleta do fluxo de vídeo em formato *RAW* por meio do Serviço de Perfil de Sessão, associando-o a uma sessão operacional em andamento. Em seguida, trata esses dados de vídeo, transformando-os em imagens em um formato padrão (PNG, JPEG) e vinculando-os aos metadados da sessão atual, disponibilizando-os em sua interface. Esses metadados incluem um identificador, o caminho de acesso às câmeras associadas e a data de início da operação, em formato padrão (JSON, YAML). Para permitir a coleta simultânea de múltiplos fluxos de vídeo, é criada uma *thread* para cada fluxo, e essa associação é registrada em um banco SQLite.

O serviço de persistência de informações requisita periodicamente esses dados de imagem associados, de forma proporcional à velocidade da inspeção. A ideia é que, quanto mais rápida for a rolagem do tecido durante a tecelagem ou revisão, maior será a

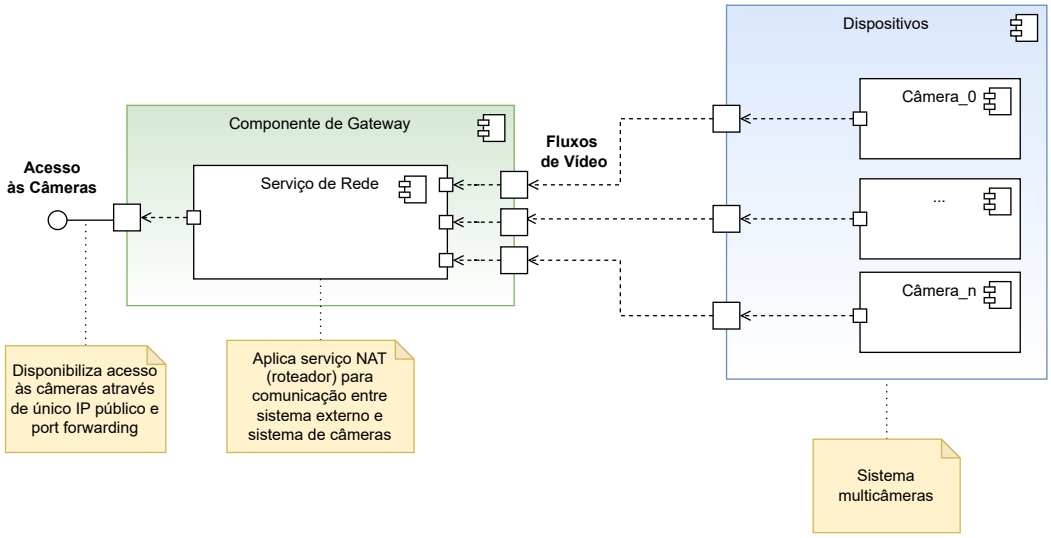

Figura 4.8: Diagrama de Componentes: Detalhamento dos Componentes de *Gateway* e Dispositivos

frequência de captura das imagens.

Ao final, duas interfaces são disponibilizadas: uma contendo todas as imagens da sessão e outra com as imagens da sessão acompanhadas dos metadados associados.

Completando o ciclo operacional *online*, no Componente de Sessão ilustrado na Figura 4.10, tanto o Serviço de Monitor, que opera sobre as informações da sessão atual, quanto o Serviço de Sessões, que acessa os dados persistidos de sessões anteriores, utilizam a interface de imagens e metadados de sessão para disponibilizar a API de sessões. O serviço de monitor também faz uso da API de processos para obter os modelos de classificação utilizados nas tarefas de inferência. A API de sessões gerada pode ser consumida por outras aplicações da plataforma, sendo mais comumente usada por aplicações operacionais, como o Aplicativo de Monitoramento de Sessão e o Aplicativo de Gerenciamento de Sessões. Ambas as aplicações utilizam o Serviço de Autenticação, garantindo que apenas usuários cadastrados possam acessar esses serviços.

O componente de processos, ilustrado na Figura 4.11, é composto pelos Serviços de Diretório de Processos, que agrega, neste caso, uma biblioteca de modelos de classificação a ser disponibilizada ao Componente de Sessão e ao Serviço de Treinamento. Este último realiza o treinamento dos modelos base com o auxílio da API de *datasets*, disponibilizando uma interface de monitoramento para o Serviço de Rastreamento. Essa interface é integrada ao *framework MLflow*, permitindo o acompanhamento dos experimentos, a persistência de parâmetros e métricas em um banco *SQLite*, e o armazenamento de artefatos gerados em um caminho de disco especificado. O serviço também disponibiliza a interface nativa do *MLflow*, possibilitando a visualização dos experimentos e a realização de análises detalhadas pelos engenheiros responsáveis.

Além disso, o componente de processos utiliza o Serviço de Detecção de Degradação

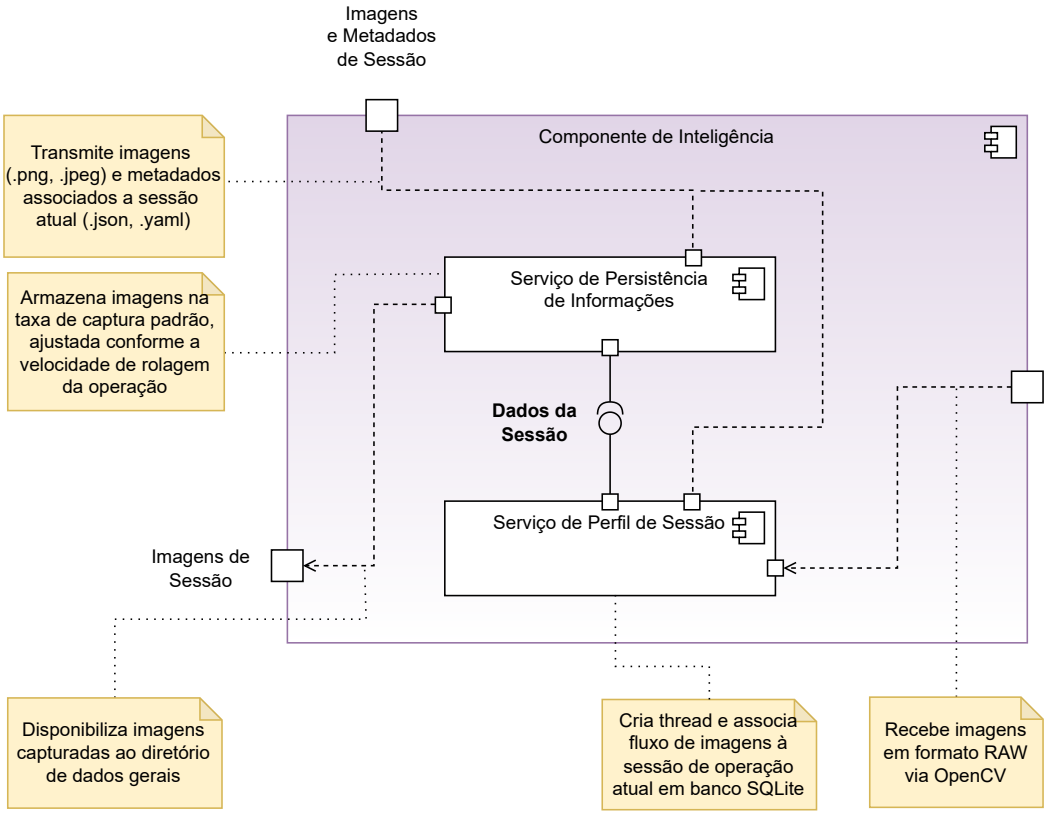

Figura 4.9: Diagrama de Componentes: Detalhamento do Componente de Inteligência

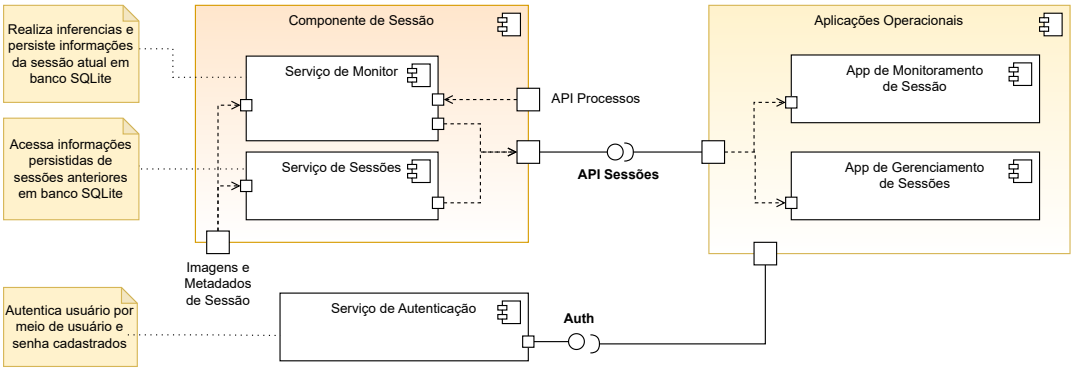

Figura 4.10: Diagrama de Componentes: Detalhamento do Componente de Sessão e Aplicações Operacionais

de Processo, que alerta os responsáveis sobre possíveis degradações no desempenho dos modelos ao longo de múltiplas sessões, e o Serviço de Tratamento de Exceções, que gerencia erros gerados durante os processos *offline* de treinamento de modelos, entre outros.

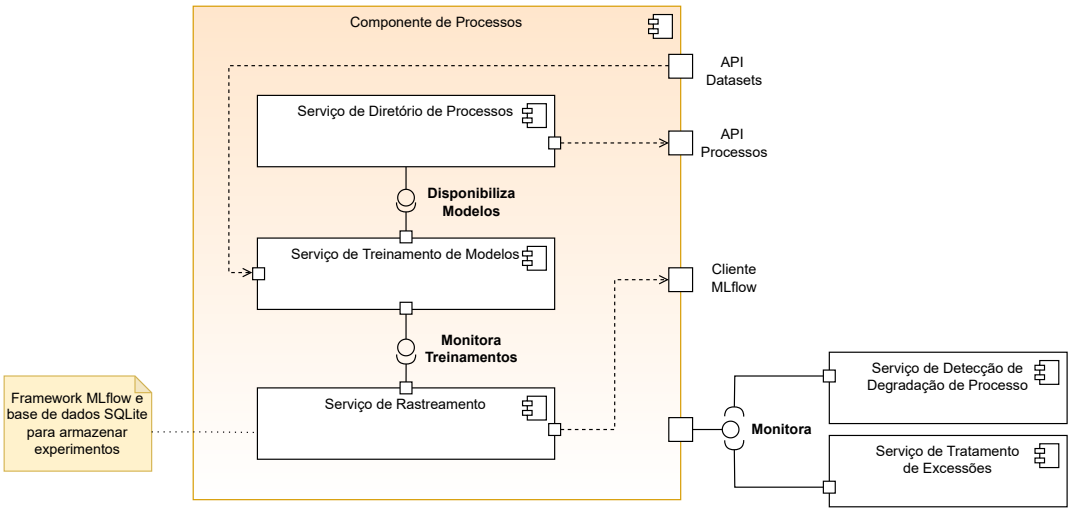

Figura 4.11: Diagrama de Componentes: Detalhamento do Componente de Processos

Por fim, o Componente de Dados, ilustrado na Figura 4.12, é composto pelo Serviço de Dados, responsável por receber imagens de sessão por meio da interface disponibilizada pelo Componente de Inteligência, bem como via API de *dataset*, através de *uploads* realizados diretamente pelo usuário. Essa API oferece funcionalidades para gerenciar os dados disponíveis no processo de treinamento de modelos, abrangendo operações como criação, exclusão, aumento (*augmentação*), mescla e rotulagem de dados.

A API é consumida por aplicações voltadas à ciência de dados, incluindo o Aplicativo de Gerenciamento de *Datasets*, o Aplicativo de Gerenciamento de Processos, que utiliza a API de processos, e o Aplicativo de Rastreamento de Experimentos, que acessa o cliente do *MLflow*. Assim como as aplicações operacionais, as aplicações de ciência de dados utilizam o Serviço de Autenticação para assegurar que apenas usuários autorizados tenham acesso aos serviços internos.

O diagrama de classes apresentado na Figura 4.13 descreve a estrutura de entidades da plataforma, representando as principais classes, seus atributos e os relacionamentos entre elas.

A classe *User* representa os usuários do sistema, que podem iniciar sessões operacionais (*Session*), criar *datasets* (*Dataset*) e versões de modelos (*ModelVersion*).

Cada *Session* deve estar associada a uma ou mais câmeras, registradas na classe *Camera*, que capturam imagens para a detecção de defeitos. Os defeitos detectados durante a *Session* são armazenados na classe *Defect*, que contém informações sobre a posição e a classe do defeito. Esses defeitos ficam associados à *Session* em que foram encontrados. Além disso, para a criação de uma *Session*, é necessário definir uma *ModelVersion* específica, que será utilizada durante a operação.

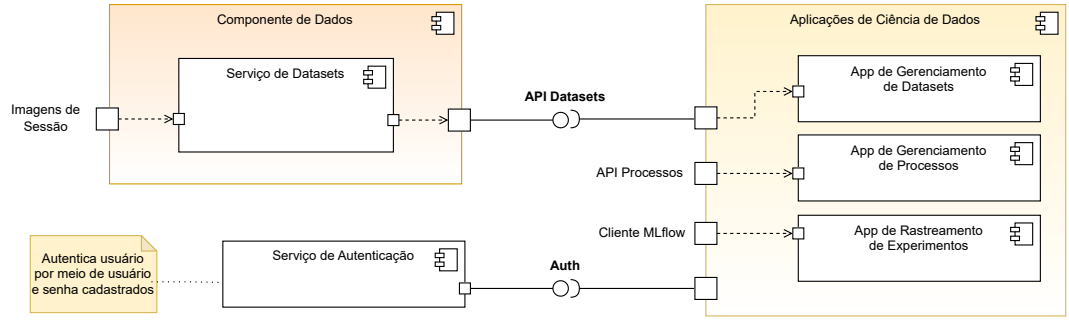

Figura 4.12: Diagrama de Componentes: Detalhamento do Componente de Dados e Aplicações de Ciência de Dados

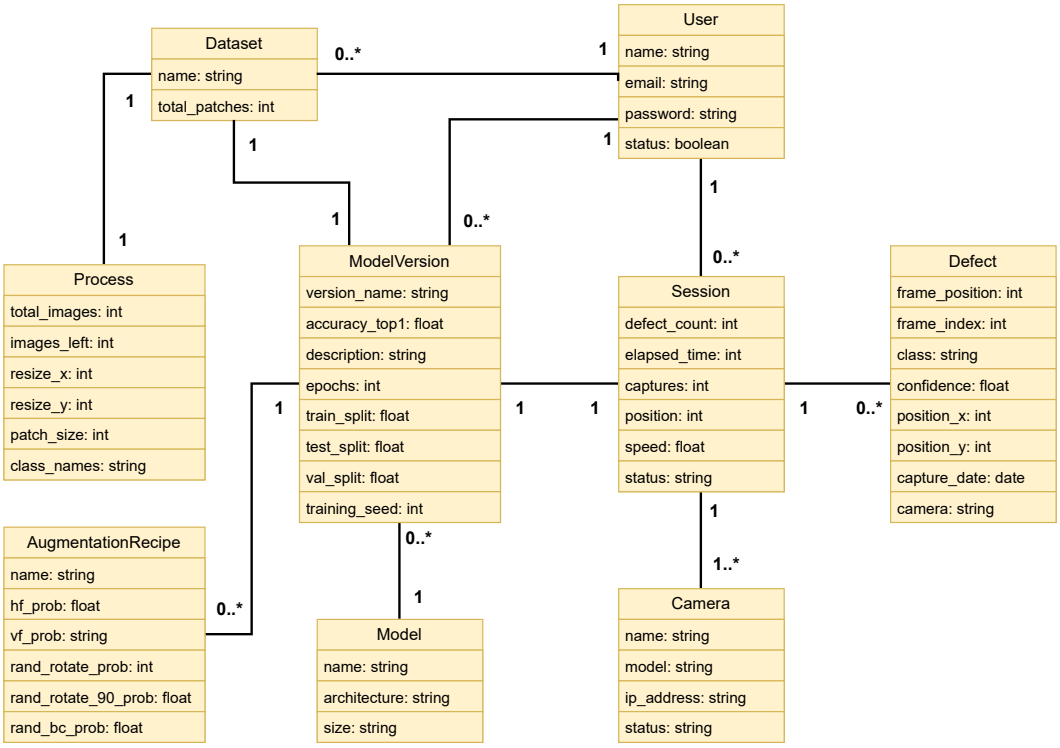

Figura 4.13: Diagrama de Classes

A classe *ModelVersion* representa uma instância de treinamento do modelo, armazenando dados como precisão, configuração de treinamento e o *Dataset* utilizado. Já a classe *Model* descreve o modelo base, incluindo sua arquitetura e características, como tamanho.

Os dados usados para treinar os modelos são organizados em *Datasets*, conjuntos de imagens processadas para o treinamento. Quando um *Dataset* está em processo de rotulamento, ele é gerido pela classe *Process*, que representa um *Dataset* em estado intermediário, ainda não pronto para o treinamento até que as etapas de rotulamento sejam concluídas.

Durante a criação de uma *ModelVersion*, o *Dataset* pode ser associado a uma *AugmentationRecipe*, uma receita de aumento de dados que expande o conjunto de dados aplicando transformações, como espelhamento, rotação e ajuste de brilho/contraste. Essas receitas podem ser configuradas de forma independente e, opcionalmente, associadas a uma *ModelVersion* no momento da configuração do treinamento.

4.4 Criação e Preparação do *Dataset*

Nesta dissertação, foi utilizada uma base de dados adaptada a partir do *Textile Texture Database* (TILDA), desenvolvido pelo grupo alemão de análise de texturas do DFG (*Deutsche Forschungsgemeinschaft*) (DFG Working Group Texture Analysis 1996). O *dataset* original contém 7 classes de defeitos em tecidos e uma classe sem defeitos, com 50 imagens em escala de cinza por classe, totalizando 400 imagens. As imagens têm resolução de 768x512 *pixels*.

Para atender às necessidades do contexto industrial de detecção de defeitos, foi criada uma versão modificada do *dataset*, denominada TILDA 400 (64x64 *patches*)[4]. Devido à alta similaridade entre duas das sete classes originais, estas foram mescladas, resultando em um total de cinco classes: sem defeitos (*good*), buraco (*hole*), objetos (*objects*), mancha de óleo (*oil spot*) e erro de linha (*thread error*). Inicialmente, as imagens foram redimensionadas de 768×512 para 512×512 *pixels*. Em seguida, foram segmentadas em retalhos de 64×64 *pixels*, cada um classificado manualmente com base nas máscaras de classe originais. Esse processo resultou em um aumento significativo no número de imagens, conforme ilustrado na Figura 4.14.

Como esperado, a classe *good* está super-representada, uma vez que, em imagens de tecidos, a maior parte da superfície tende a estar livre de defeitos, resultando em uma distribuição desbalanceada entre as classes.

Para reduzir o desbalanceamento entre as classes, foi realizado um *undersampling* na classe *good*, limitando-a a 1000 amostras. Em seguida, aplicou-se aumento de dados, ou *data augmentation*, para aumentar o número de imagens nas classes menos representadas, utilizando as bibliotecas *OpenCV* e *Albumentations*. Esta última oferece uma ampla gama de transformações avançadas para manipulação de imagens. As transformações aplicadas foram:

[4] https://www.kaggle.com/datasets/angelolmg/tilda-400-64x64-patches

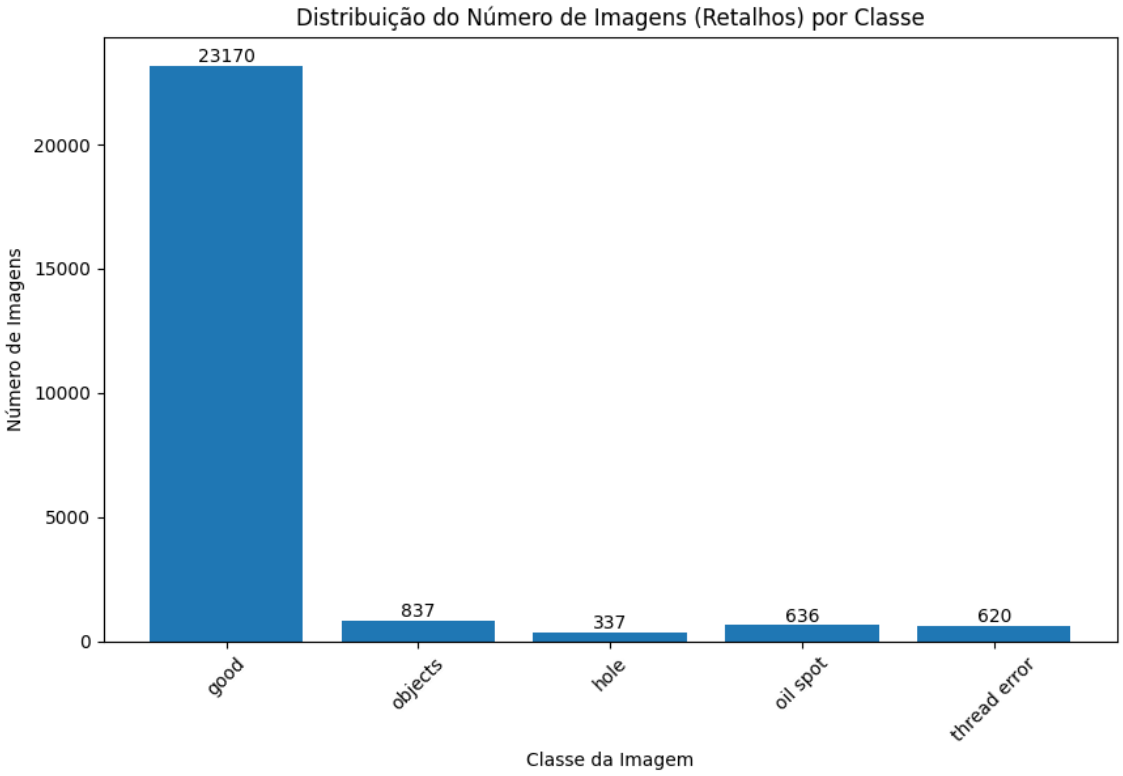

Figura 4.14: Distribuição de imagens (retalhos) por classe para o *dataset* TILDA

- **HorizontalFlip**: Inverte a imagem horizontalmente com uma probabilidade de 50% (`p=0.5`). Isso ajuda a criar variações em que o padrão é refletido lateralmente.
- **VerticalFlip**: Inverte a imagem verticalmente, também com uma probabilidade de 50% (`p=0.5`), gerando variações de padrões refletidos no eixo vertical.
- **RandomRotate90**: Rotaciona a imagem em incrementos de 90 graus com uma probabilidade de 50% (`p=0.5`), permitindo que a orientação da imagem seja alterada sem modificar o conteúdo visual.
- **Rotate**: Rotaciona a imagem por um ângulo aleatório entre 0 e 180 graus (`limit=180`), com uma probabilidade de 75% (`p=0.75`). A interpolação utilizada é `cv2.INTER_LANCZOS4`, que utiliza o método de Lanczos sobre uma vizinhança 8x8; e o modo de borda utilizado é `cv2.BORDER_REFLECT`, que reflete as bordas da imagem para evitar artefatos.
- **RandomBrightnessContrast**: Ajusta aleatoriamente o brilho e o contraste da imagem, com uma probabilidade de 75% (`p=0.75`). Isso simula diferentes condições de iluminação, aumentando a robustez do modelo.

A Tabela 4.5 a seguir resume as transformações aplicadas no processo de aumento de dados:

Tabela 4.5: Transformações aplicadas no processo de aumento de dados utilizando biblioteca *Albumentations*

Transformação	**Descrição**	**Probabilidade (*p*)**
HorizontalFlip	Inversão horizontal da imagem	0.5
VerticalFlip	Inversão vertical da imagem	0.5
RandomRotate90	Rotação em incrementos de 90 graus	0.5
Rotate	Rotação aleatória entre 0 e 180 graus	0.75
RandomBrightnessContrast	Ajuste de brilho e contraste	0.75

Essas transformações garantem que o modelo de classificação receba uma maior diversidade de imagens durante o treinamento, o que contribui para sua capacidade de generalização. Ao aplicar variações controladas, o aumento de dados também ajuda a evitar problemas de *overfitting*, ao mesmo tempo em que simula diferentes condições do mundo real, como variações de orientação e iluminação.

A Tabela 4.6 apresenta a taxa de aumento aplicada a cada classe após o processo de aumento de dados.

Como resultado, obteve-se o total final de imagens por classe, conforme ilustrado na Figura 4.15, totalizando 25125 imagens.

A Figura 4.16 apresenta amostras dos retalhos gerados após o processo de segmentação e *data augmentation*.

Finalmente, o *dataset* foi dividido em três subconjuntos: treino, validação e teste, nas proporções de 70%, 20% e 10%, respectivamente. A distribuição de imagens por classe em cada subconjunto está ilustrada na Figura 4.17.

Tabela 4.6: Taxa de aumento por classe

Classe	Taxa de Aumento
good	5
objects	6
hole	15
oil spot	8
thread error	8

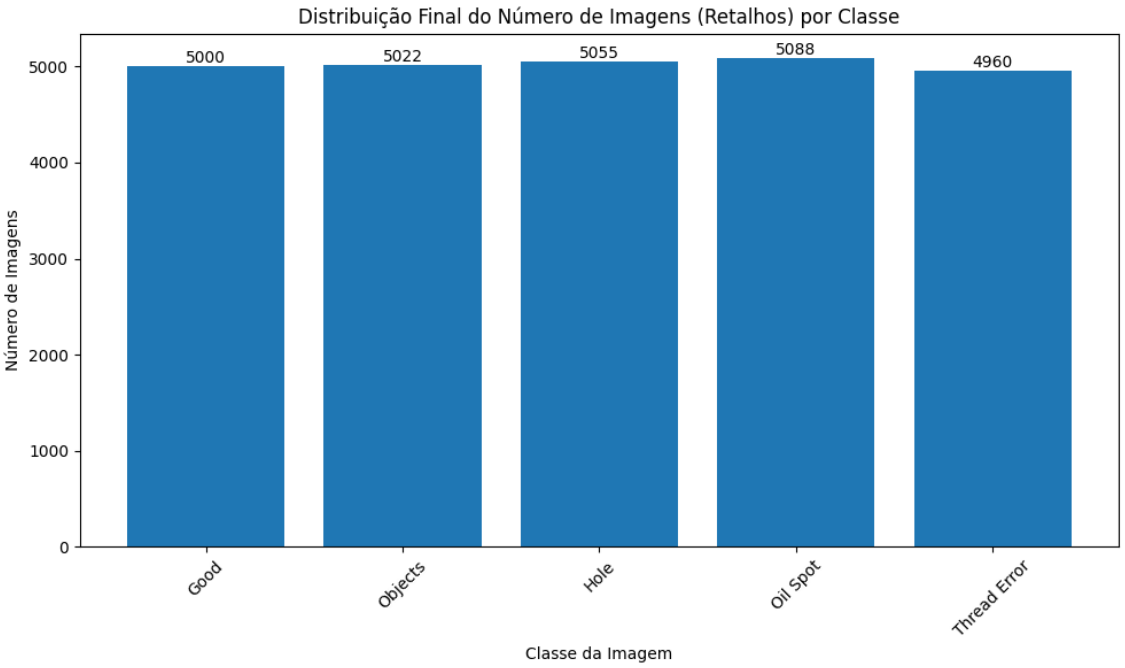

Figura 4.15: Distribuição final de imagens por classe após aumento de dados

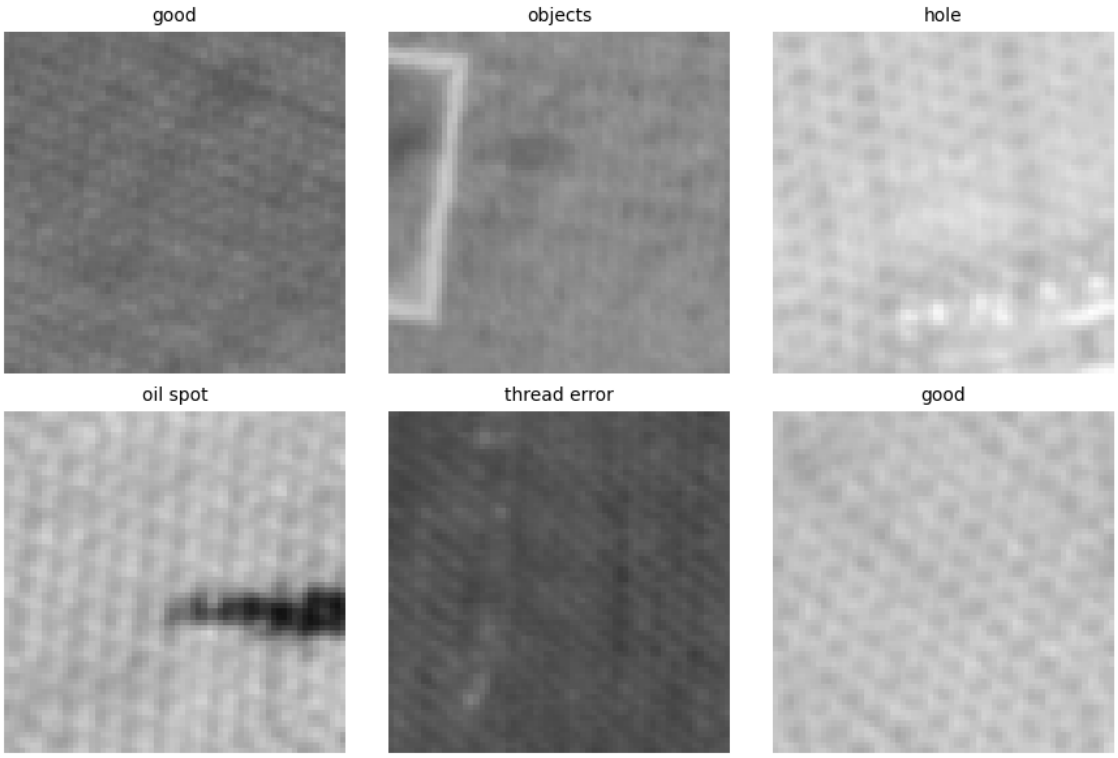

Figura 4.16: Amostras de retalhos após o processo de segmentação e aumento de dados

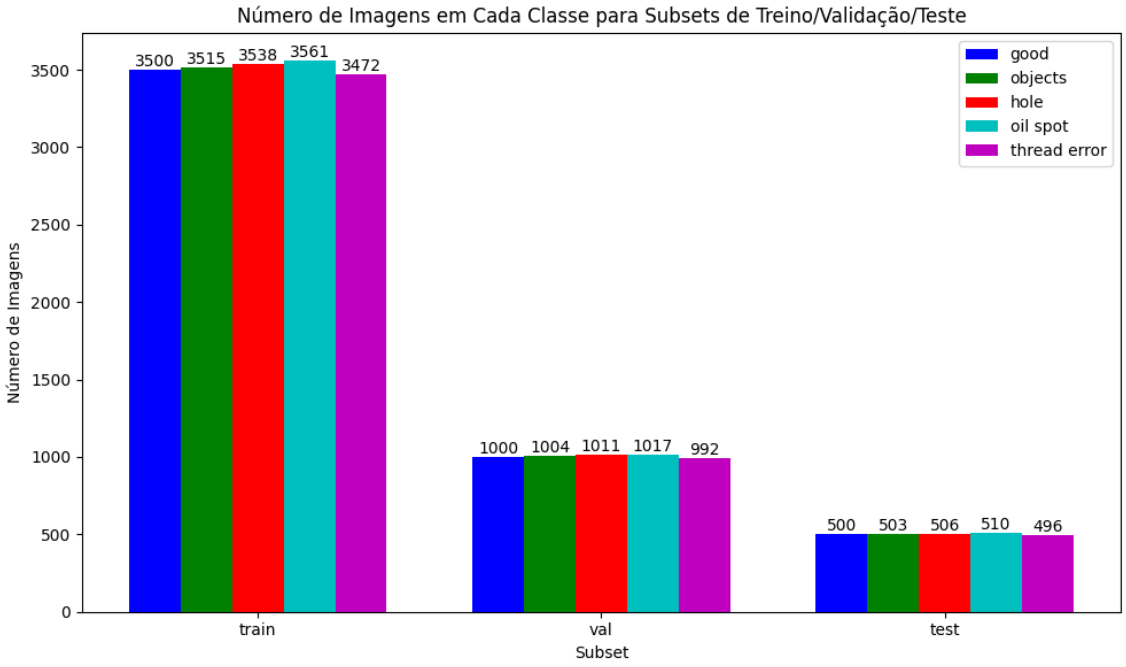

Figura 4.17: Distribuição de imagens por classe nos subconjuntos de treino, validação e teste.

Capítulo 5

Experimentos e Resultados

Este capítulo apresenta os resultados obtidos nos experimentos, a análise comparativa dos modelos e a implementação da PoC da plataforma. A Seção 5.1 mostra as métricas de desempenho e compara o desempenho dos modelos avaliados com o estado da arte, utilizando uma base de dados comum. Na Seção 5.2, detalha-se a implementação da PoC da Plataforma de Inspeção de Qualidade em Tecidos, abordando suas interfaces e funcionalidades.

5.1 Dados Obtidos

5.1.1 Métricas dos Experimentos

A Tabela 5.1 apresenta as métricas de desempenho dos oito modelos testados até 100 épocas de treinamento. Os dois primeiros, Jing et al. e Sandhya et al., foram selecionados da literatura por adotarem metodologias mais alinhadas à abordagem deste experimento.

Tabela 5.1: Métricas de Desempenho de Classificação entre Modelos (100 Épocas)

Modelo	**Precisão (%)**	**Recall (%)**	**F1 Score (%)**	**Acurácia (%)**
Jing et al.	55.66	55.10	55.32	55.12
Sandhya et al.	50.07	44.91	42.25	44.86
YOLOv5n	58.55	49.57	45.08	49.68
YOLOv5s	60.11	50.01	47.34	50.14
YOLOv5m	21.33	26.91	15.22	27.07
YOLOv8n	83.55	79.61	80.13	79.60
YOLOv8s	88.18	86.96	87.18	86.96
YOLOv8m	**90.96**	**90.34**	**90.47**	**90.35**

A Tabela 5.2 apresenta uma comparação das velocidades de execução entre os modelos treinados após 100 épocas. Já a Figura 5.1 oferece uma visualização complemen-

tar, onde os pontos mais próximos do canto superior esquerdo indicam melhores desempenhos, representando modelos que combinam alta acurácia com menor tamanho e/ou tempo de inferência.

Tabela 5.2: Acurácia, Tempos de Inferência, Parâmetros Treináveis e Tempo de Treino entre Modelos (100 Épocas)

Modelo	**Acurácia (%)**	**Inferência (ms)**	**Parâmetros (M)**	**Treino (min)**
Jing et al.	55.12	**0.0016**	6.46	**21.92**
Sandhya et al.	44.86	0.089	54.56	50.57
YOLOv5n	49.68	0.4	**1.9**	23.33
YOLOv5s	50.14	0.6	7.2	23.55
YOLOv5m	27.07	1.0	21.2	30.43
YOLOv8n	79.6	0.4	3.2	52.10
YOLOv8s	86.96	0.4	11.2	52.70
YOLOv8m	**90.35**	0.5	25.9	68.18

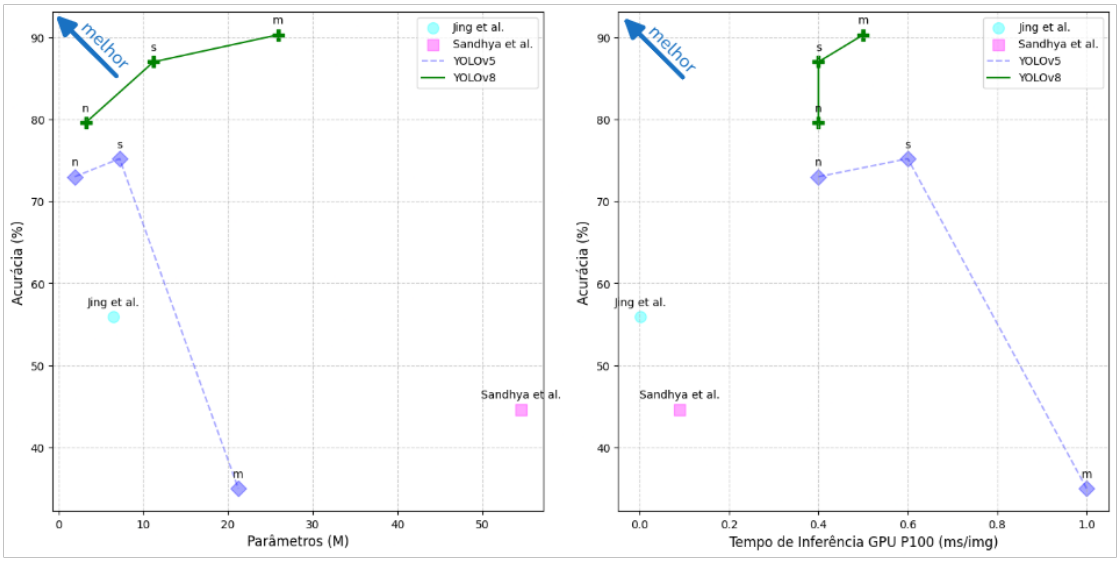

Figura 5.1: Acurácia (%) vs Parâmetros (M) e Acurácia (%) vs Tempo de Inferência (ms) entre Modelos (100 Épocas)

As Figuras 5.2, 5.4 e 5.6 mostram as matrizes de confusão obtidas para os três principais modelos testados no *subset* de validação: Jing et al. Sandhya et al. e YOLOv8m. Já as Figuras 5.3, 5.5 e 5.7 apresentam as curvas de *loss* (perda) e acurácia durante o treinamento desses mesmos modelos. As matrizes de confusão e curvas referentes aos demais modelos estão disponíveis no Apêndice A.

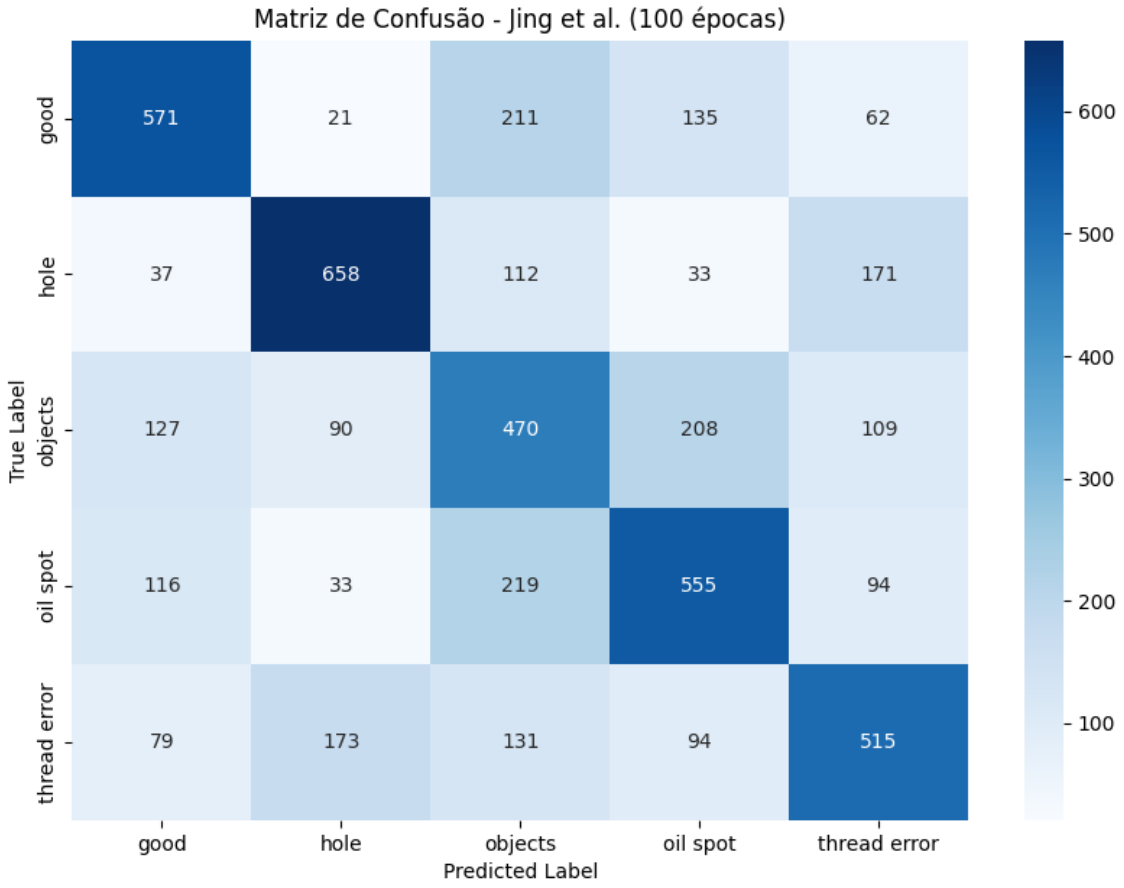

Figura 5.2: Matriz de Confusão - Jing et al. (100 Épocas)

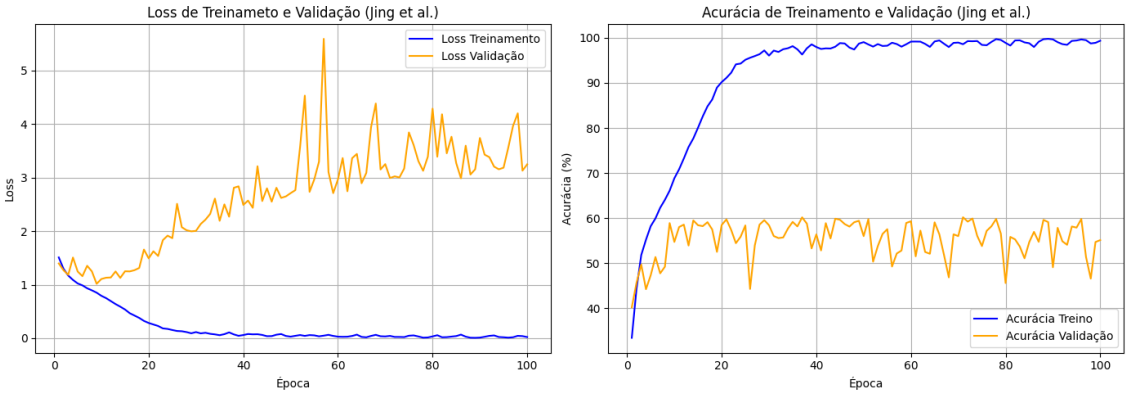

Figura 5.3: Curva de Loss e Acurácia para Modelo Jing et al.

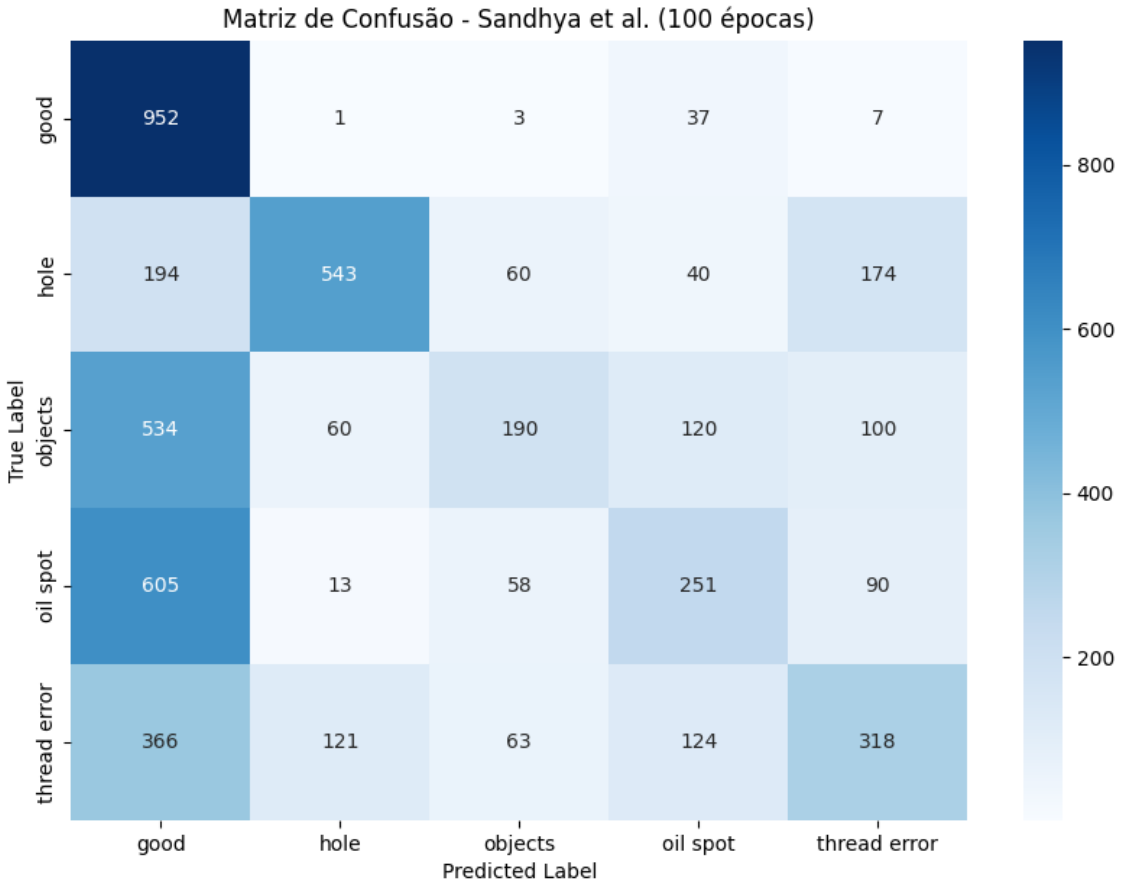

Figura 5.4: Matriz de Confusão - Sandhya et al. (100 Épocas)

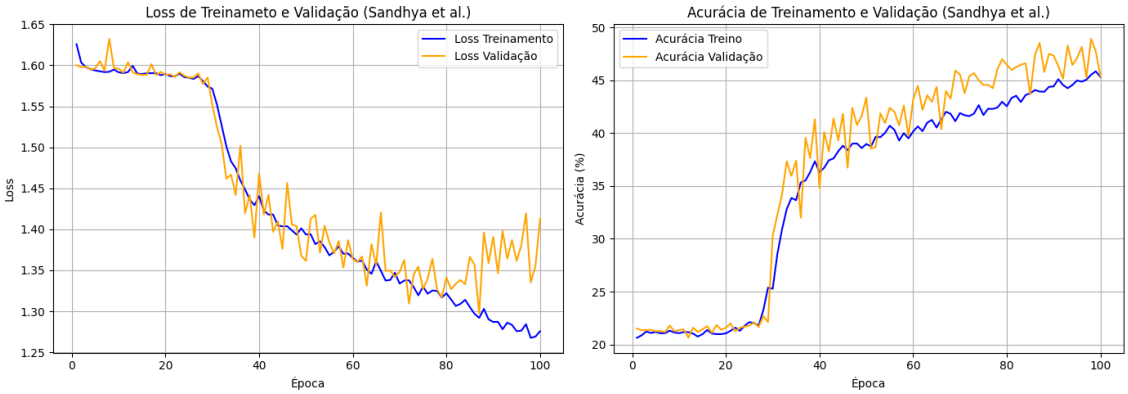

Figura 5.5: Curva de Loss e Acurácia para Modelo Sandhya et al.

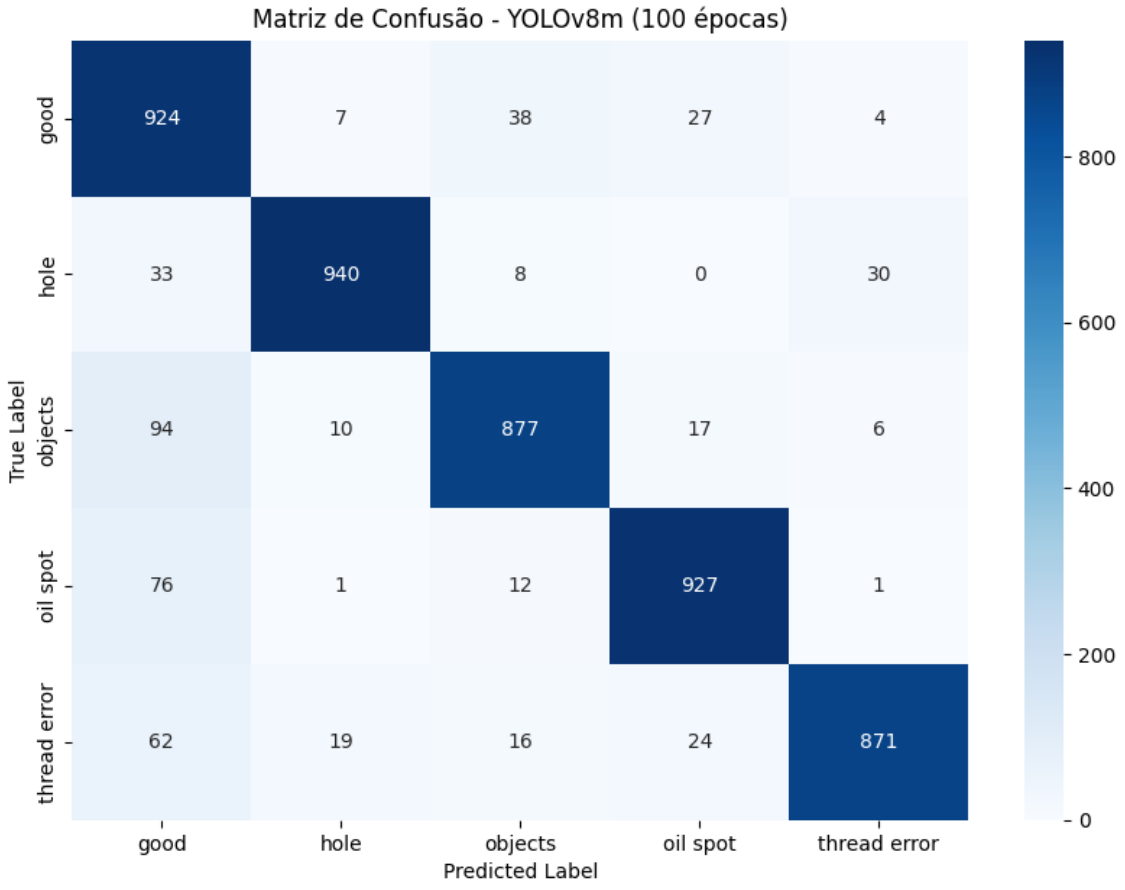

Figura 5.6: Matriz de Confusão - Modelo YOLOv8m (100 Épocas)

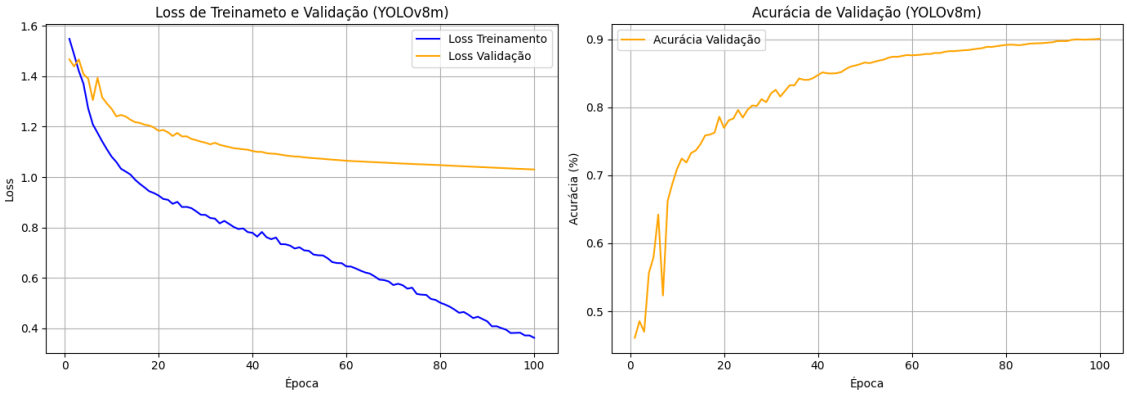

Figura 5.7: Curva de *Loss* e Acurácia para Modelo YOLOv8m

5.1.2 Análise e Interpretação

Sobre o *dataset*, a aplicação de aumento de dados no TILDA é recomendada, uma vez que o número de exemplos em certas classes é bastante reduzido (*e.g.* 337 para classe *hole* e 620 para *thread error*), como mostrado na Figura 4.14. Com poucos exemplos por classe, os modelos podem sofrer de *underfitting*, comprometendo seu desempenho.

As transformações descritas na Tabela 4.5 foram selecionadas empiricamente, buscando refletir possíveis variações de rotação e iluminação observadas em uma instalação padrão. Essas transformações auxiliam na capacidade de generalização do modelo, mas não constituem uma lista exaustiva — outras transformações podem ser escolhidas conforme a situação. Por essa razão, na implementação da arquitetura de referência ilustrada na Figura 4.5, foi incluído um módulo neutro e opcional de receitas de augmentação, permitindo que as transformações sejam ajustadas às necessidades específicas da operação.

Dito isso, os valores de aumento de dados descritos na Tabela 4.6, assim como a proporção de *split* para os *subsets* de treino, validação e teste (70/20/10), são considerados razoáveis conforme as práticas comuns.

Ainda sobre o *dataset*, a justificativa para o uso do algoritmo de detecção em grelha é que ele gera um número ainda maior de exemplos de treinamento, uma vez que não há uma grande quantidade de bases de defeitos em tecidos disponíveis para treinar modelos com imagens inteiras. Além disso, como a detecção ocorre regionalmente em cada imagem, sendo cada região independente, a solução se torna "embaraçosamente paralela", permitindo sua divisão em múltiplas *threads* de inferência. Sabendo-se o número de *patches* de entrada para cada imagem, o paralelismo pode ser explorado para acelerar ainda mais o procedimento, distribuindo a carga de inferência regional entre várias *threads* e/ou processos.

O modelo que mais se destacou em possíveis aplicações práticas foi o YOLOv8s. Conforme os resultados apresentados na Tabela 5.1, esse modelo obteve a segunda melhor acurácia, alcançando 86,96%, ficando atrás apenas do YOLOv8m. Esse valor supera a média de 80% de acurácia de inspetores humanos. Além disso, o YOLOv8s demonstrou a melhor relação entre tamanho e acurácia, bem como entre latência e acurácia, como ilustrado na Figura 5.1.

Com um tempo de inferência de 0,4 ms por *patch* de 64×64 *pixels*, podemos calcular a velocidade teórica de detecção ao processar uma imagem completa de 512×512 *pixels*. O número total de *patches* por imagem é:

$$\frac{512 \times 512}{64 \times 64} = 64$$

Assim, o tempo total de detecção por imagem (T_{dec}) e a frequência de detecção (F_{dec}) são dados por:

$$\begin{aligned} T_{dec} &= 0,0004 \times 64 = 0,0256\,\text{s/imagem} \\ F_{dec} &= \frac{1}{T_{dec}} = 39,0625\,\text{imagens/s} \end{aligned} \tag{5.1}$$

Agora, considerando que o tecido em questão possui 2 metros de largura, e que cada

imagem de 512×512 *pixels* cobre 20 cm², precisamos de 10 imagens para cobrir toda a largura do tecido (ou seja, em uma faixa horizontal de 20 cm de altura), sendo a área total de:

$$\text{área total da faixa horizontal} = 2\,\text{m} \times 20\,\text{cm} = 4000\,\text{cm}^2$$

O número de faixas detectadas por segundo é:

$$\text{faixas/s} = \frac{\text{imagens/s}}{\text{imagens/faixa}} = \frac{39,0625}{10} = 3,90625\,\text{faixas/s} \tag{5.2}$$

Logo, o número de faixas processadas por minuto será:

$$\text{faixas/min} = 3,90625\,\text{faixas/s} \times 60\,\text{s/min} = 234,375\,\text{faixas/min} \tag{5.3}$$

Sabendo que cada faixa, ou área de detecção completa, possui 20 cm de altura, podemos calcular a velocidade máxima teórica de deslocamento do tecido (V_{dec}), expressa em metros por minuto:

$$V_{dec} = 234,375\,\text{faixas/min} \times 0,2\,\text{m/faixa} = 46,875\,\text{m/min} \tag{5.4}$$

Isso significa que, sob as condições descritas, a inspeção pode teoricamente operar a uma velocidade de aproximadamente 46,875 metros por minuto em uma GPU NVIDIA Tesla P100.

Esse valor é consideravelmente superior à velocidade máxima observada na inspeção manual, que varia entre 15 e 20 m/min. No entanto, é importante destacar que esses cálculos desconsideram potenciais *overheads*, como os custos computacionais adicionais relacionados à preparação, transmissão e processamento dos dados, bem como o campo de visão da câmera e o comprimento total do tecido, fatores que podem impactar as métricas de desempenho final. Por outro lado, técnicas de otimização pós-treinamento, como a poda de modelos (*pruning*), podem reduzir significativamente o tamanho do modelo e diminuir o tempo de inferência, desativando conexões e neurônios pouco utilizados.

Modelos apresentados na literatura, como os de Jing et al. e Sandhya et al., demonstram tempos de inferência significativamente menores em comparação com os modelos da família YOLO, sendo 250 e 4,5 vezes mais rápidos, respectivamente. No entanto, seu desempenho em termos de classificação foi notavelmente inferior. Em cenários mais simples, com menos classes ou padrões mais evidentes, modelos como o de Jing et al. podem atingir uma acurácia aceitável, justificando seu uso pela significativa redução de latência. Além disso, testes subsequentes com o modelo Jing et al. indicaram que a aplicação de *dropout* na saída da primeira camada FC aumentou a acurácia em cerca de 8%, embora ainda inferior aos modelos mais recentes da família YOLO.

Já Sandhya et al. não obteve resultados satisfatórios, alcançando uma acurácia final de 44,86% e gerando um número elevado de falsos negativos, especialmente para as classes *'objects'*, *'oil spot'* e *'thread error'*, conforme ilustrado na matriz de confusão na Figura 5.4. Embora o modelo tenha o maior número de parâmetros e baixa velocidade de inferência em comparação com os demais, sua má performance sugere que o *dataset*, composto por 17587 imagens de treinamento, não seja grande o bastante para saturar a

rede, resultando em *underfitting*. Isso parece plausível ao comparar com o treinamento original do AlexNet, que utilizou cerca de 1,2 milhões de imagens de alta resolução para 1000 classes diferentes (Krizhevsky et al. 2012).

Vale destacar também o problema de falsos positivos inerente ao algoritmo de grelha, conforme observável na Figura 5.6. Como o algoritmo assume que todo retalho pertence inicialmente à classe padrão (*good*), alterando essa classificação apenas se outro defeito superar um limiar de confiança τ_1, os valores para essa classe tendem a ser superestimados. Esse efeito é reconhecido como uma limitação e merece uma análise mais aprofundada em trabalhos futuros.

5.2 Implementação da Plataforma

Nesta seção, é detalhada a implementação da PoC da Plataforma de Inspeção de Qualidade em Tecidos, destacando as interfaces que a compõem e suas principais funcionalidades. Essas interfaces proporcionam o ambiente de interação para os profissionais operacionais e os engenheiros/técnicos responsáveis por operar e alimentar o sistema. Além disso, todo o código-fonte do sistema implementado está disponível publicamente no repositório do GitHub[1].

A seguir, são apresentadas as principais telas da plataforma:

a) A **Tela Inicial** é a principal interface operacional da aplicação, onde o usuário controla a sessão e visualiza seus elementos descritivos essenciais. Nesta tela, o usuário pode selecionar o *feed* de vídeo da câmera conectada e escolher um modelo de classificação dentre as opções disponíveis no menu suspenso vinculado ao diretório de modelos. A interface exibe informações essenciais da sessão, incluindo o número da sessão, tempo decorrido em segundos, número de capturas da câmera, velocidade do tecido rolante em metros por minuto, posição do tecido desde o início da sessão em metros e o número total de defeitos detectados até o momento.

No lado esquerdo da tela inicial, a captura de vídeo atual é exibida, com os defeitos detectados destacados. No lado direito, um gráfico de dispersão fornece uma representação visual da localização dos defeitos detectados. Os eixos do gráfico correspondem à posição vertical e horizontal de cada defeito ao longo do tecido. Diferentes cores são usadas para representar diferentes classes de defeitos, garantindo que o usuário possa diferenciá-los facilmente. A Figura 5.8 demonstra a implementação da tela inicial descrita.

b) A **Tela de Detalhes** oferece um sumário básico sobre cada um dos defeito detectados durante a sessão atual. Esta tela exibe informações abrangentes para cada defeito, incluindo a posição do quadro, que indica o índice do quadro capturado onde o defeito apareceu, e o índice do quadro, que mostra a posição específica do quadro dentro do *feed* de vídeo onde o defeito foi detectado. Detalhes adicionais incluem o nome da câmera que capturou o defeito, a classificação do defeito e o nível de confiança na classificação.

Além disso, a tela exibe as coordenadas X e Y do defeito dentro do quadro original, medidas em *pixels*, bem como a data e hora exatas da captura, formatadas como dd/MM/aa hh:mm. Por fim, cada entrada também conta com a imagem do defeito, per-

[1] `https://github.com/angelolmg/textile-defect-detection-app`

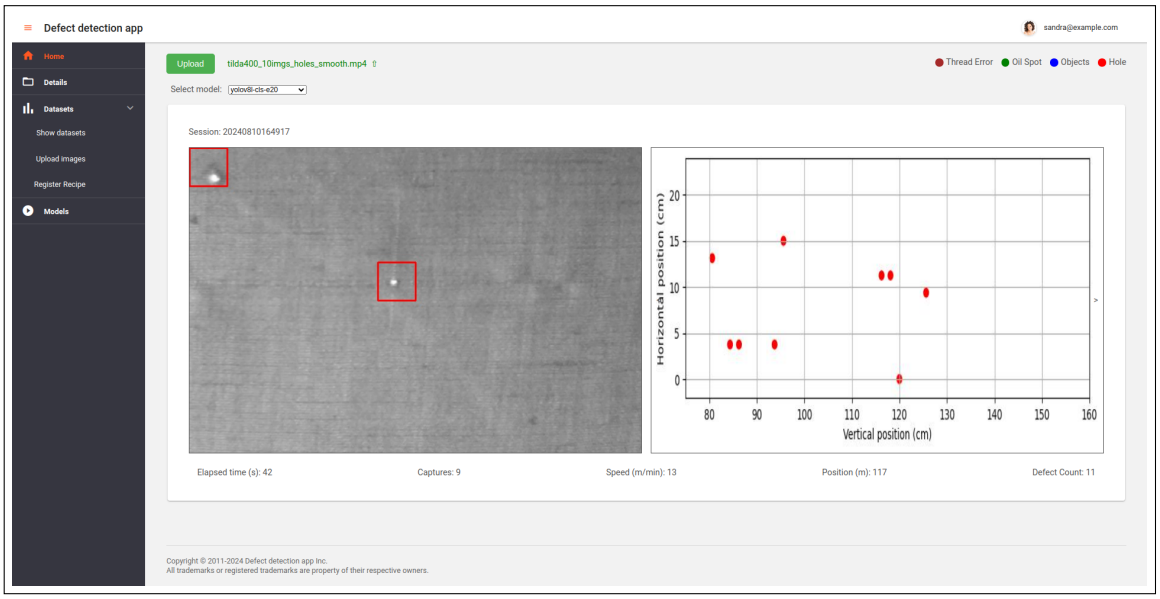

Figura 5.8: Tela de Início

mitindo que o usuário ateste visualmente o mesmo. A tela de detalhes é ilustrada na Figura 5.9.

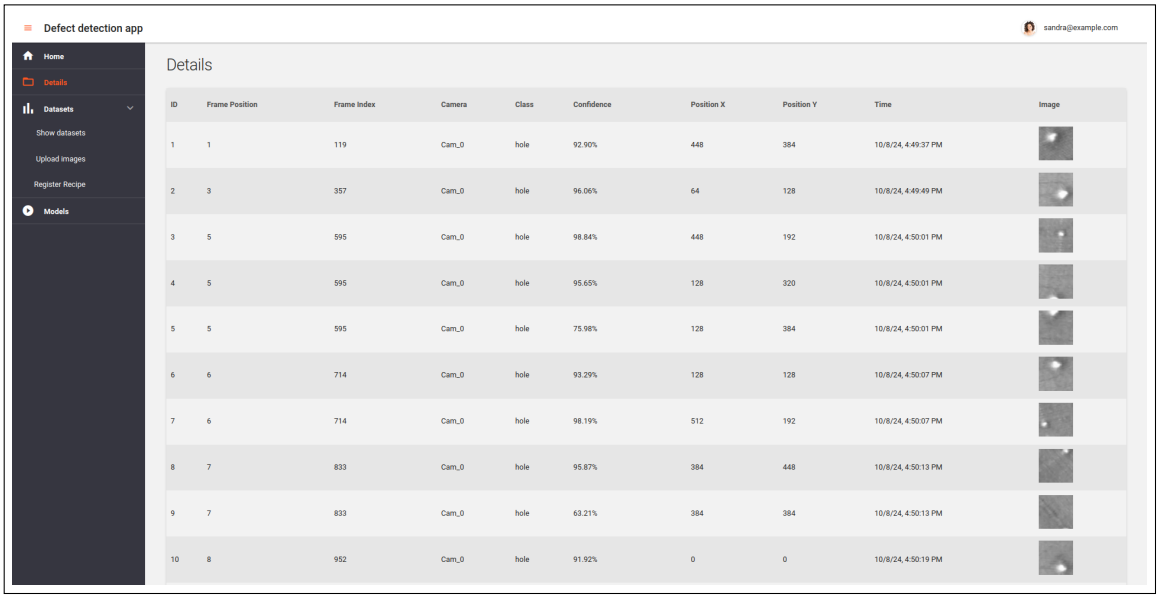

Figura 5.9: Tela de Detalhes

c) A **Tela de Listagem de Datasets** oferece uma visão organizada de todos os *datasets* disponíveis que podem ser utilizados para o treinamento de novos modelos. Cada *dataset* é acompanhado de informações-chave, como o ID do *dataset*, nome, número de *patches* (ou amostras), tamanho dos *patches* em *pixels* e nomes das classes incluídas no *dataset*.

Além disso, um botão "*View*" (visualizar) permite que os usuários inspecionem as imagens dentro de cada *dataset*. A listagem é apresentada em estilo paginado, facilitando a navegação e visualização de amostras.

Abaixo da listagem de *datasets*, há uma seção dedicada aos processos. Um processo refere-se a um *dataset* que está atualmente em construção. Isso ocorre quando novos dados foram carregados e um novo *dataset* está sendo criado, mas as imagens ainda não foram completamente recortadas e rotuladas. Para cada processo, a tela fornece detalhes como o número total de imagens, o número de imagens restantes a serem recortadas e rotuladas, a largura e altura de redimensionamento (indicando como as imagens originais foram redimensionadas) e um botão "*Patch*" (recortar), que permite ao usuário retomar o processo de rotulagem. A Figura 5.10 mostra a tela de listagem implementada.

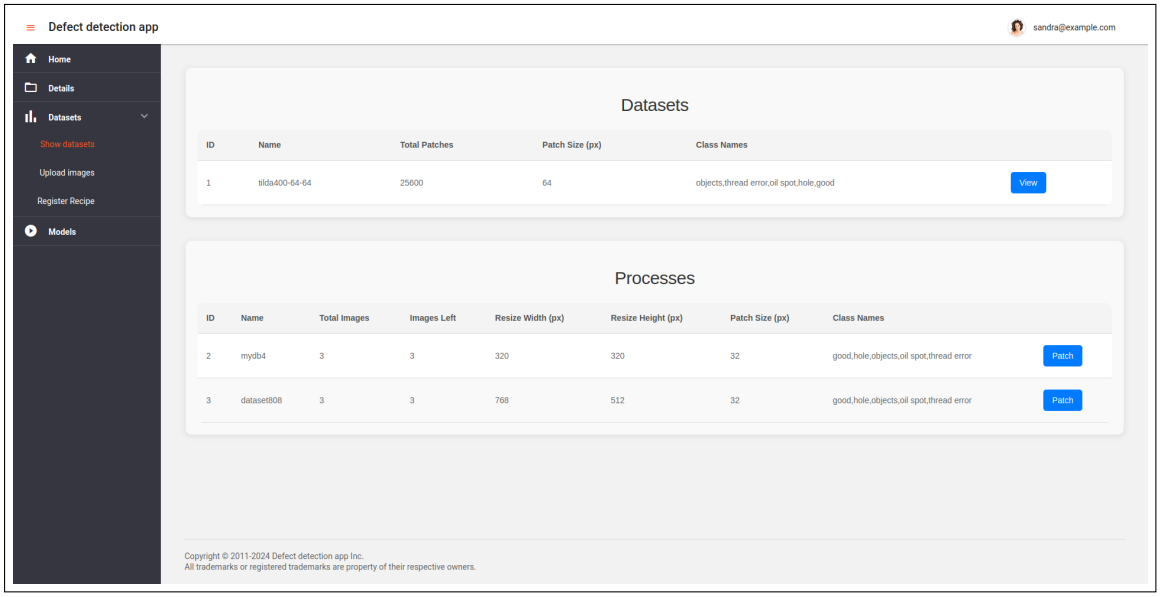

Figura 5.10: Tela de Listagem de Bases de Dados

d) A **Tela de *Upload*** permite ao usuário carregar imagens brutas para recorte. Nessa tela, o usuário pode criar um novo *dataset* inserindo ou selecionando um nome, ou optar por um *dataset* já existente para receber as imagens recortadas. Ao escolher um *dataset* existente, é necessário herdar características como o tamanho dos *patches* e os nomes das classes, garantindo o correto funcionamento do processo de fusão.

Além disso, o usuário pode optar por fazer o *upload* de um *dataset* totalmente estruturado, já recortado e rotulado. Nesse caso, é necessário incluir o arquivo de metadados correspondente, denominado `metadata.json`. Tanto o *dataset* quanto os metadados devem estar juntos em um arquivo compactado, como `.zip`, conforme detalhado na Figura 5.11.

e) A **Tela de Recorte** foi projetada para permitir que o usuário destaque manualmente os defeitos em cada imagem, dividindo-a em seções quadradas, semelhante a um CAPTCHA. O usuário pode selecionar a região específica onde um defeito está localizado dentro desses quadrados. Botões de navegação estão disponíveis para avançar e retroceder entre as imagens, permitindo que o usuário revise e destaque os defeitos em cada

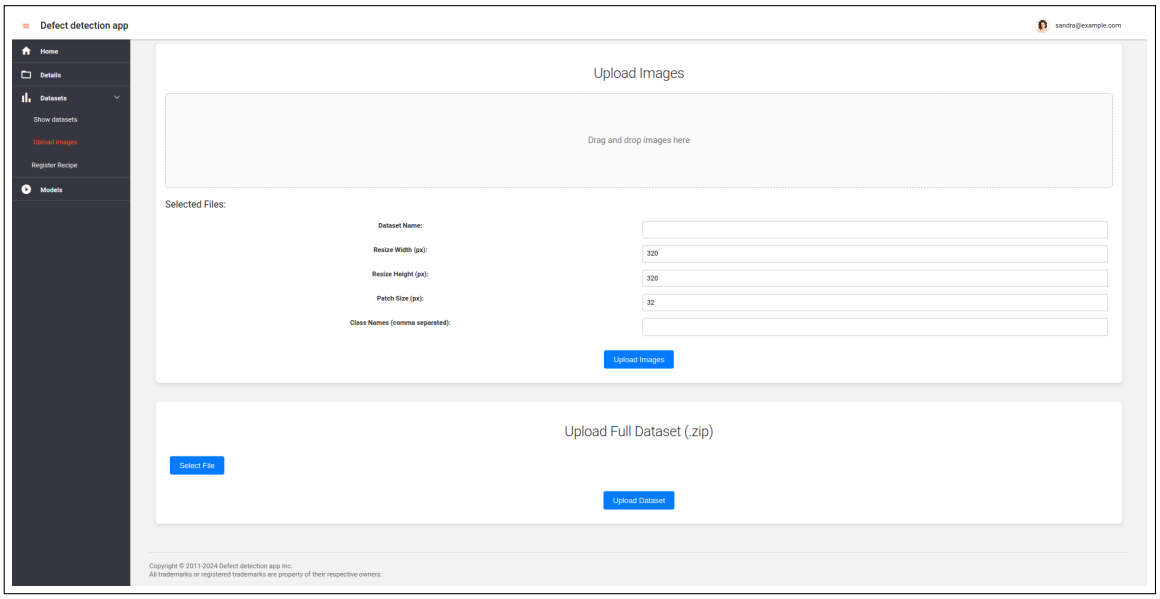

Figura 5.11: Tela de *Upload* de Imagem

uma. Um menu suspenso permite selecionar qual defeito destacar, e um botão "*Submit*" (submeter) finaliza os destaques.

Após a submissão, essas informações são utilizadas para criar um *dataset* final de *patches*. Os quadrados que contêm defeitos destacados são separados em diferentes pastas correspondentes a cada classe de defeito. As imagens que não forem destacadas são automaticamente colocadas em uma pasta de classe padrão, tipicamente a classe "*good*" (boa), que representa imagens sem defeitos.

Além de criar o *dataset* de *patches*, o sistema gera metadados em formato JSON. Esses metadados incluem informações básicas como o nome do *dataset*, o número total de imagens originais, as dimensões de redimensionamento e o número de *patches* em cada classe. Caso aplicado, esses metadados são necessários para o serviço de integridade, que verifica a integridade do *dataset* antes de usá-lo para treinamento.

Se o usuário sair da tela antes de submeter, a sessão de recorte atual é salva como um processo. Essa sessão pode ser retomada posteriormente clicando no botão "*Patch*" (recortar) no respectivo processo, como mostrado na Figura 5.10. A tela de recorte é ilustrada na Figura 5.12.

f) A **Tela de Registro de Receita de Aumento de Dados** permite que o usuário insira uma receita de aumento, que pode ser adicionada ao diretório de receitas. Uma receita de aumento, ou receita de augmentação, é uma etapa opcional nos dados utilizados durante o processo de treinamento, onde uma "receita" — uma série de operações potenciais — é aplicada a cada imagem no *dataset* final para aumentar o número de amostras.

Nesta tela, o usuário pode definir o nome da receita e configurar as probabilidades (p) para vários tipos de aumentos a serem aplicados, sendo 0% o valor padrão. Entre os aumentos disponíveis estão *flips* verticais e horizontais, rotação aleatória, ajustes aleatórios de brilho e contraste, efeitos de desfoque, entre outros.

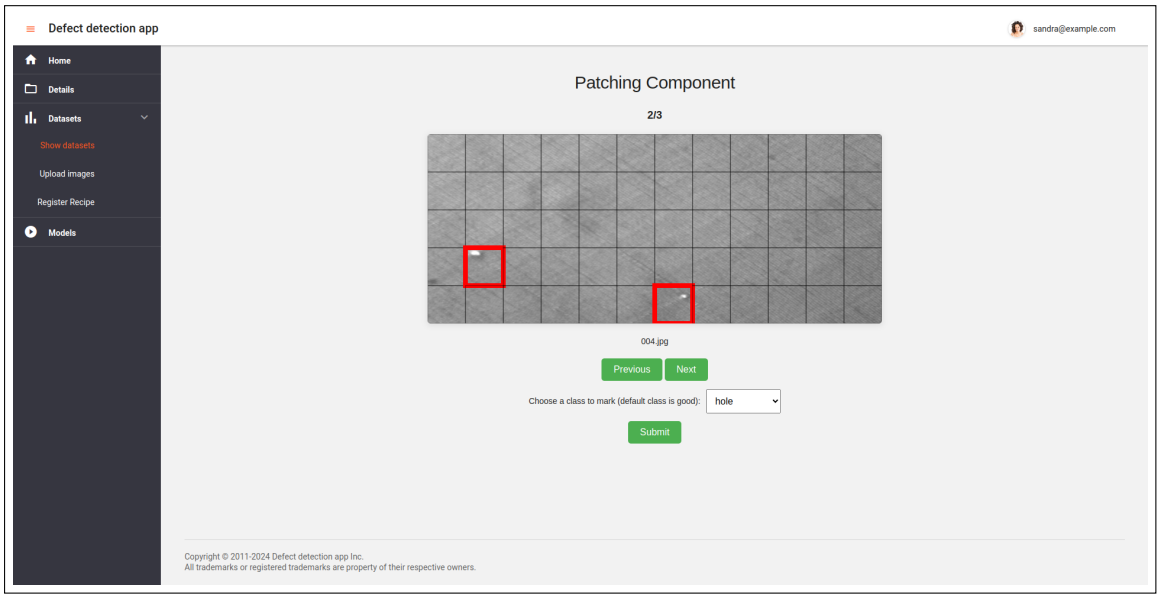

Figura 5.12: Tela de Recorte de Imagem

Essa personalização permite que os usuários apliquem diferentes níveis de aumento de dados conforme as necessidades do treinamento do modelo, melhorando assim a diversidade do *dataset* de treinamento. A tela é ilustrada na Figura 5.13 conforme descrição.

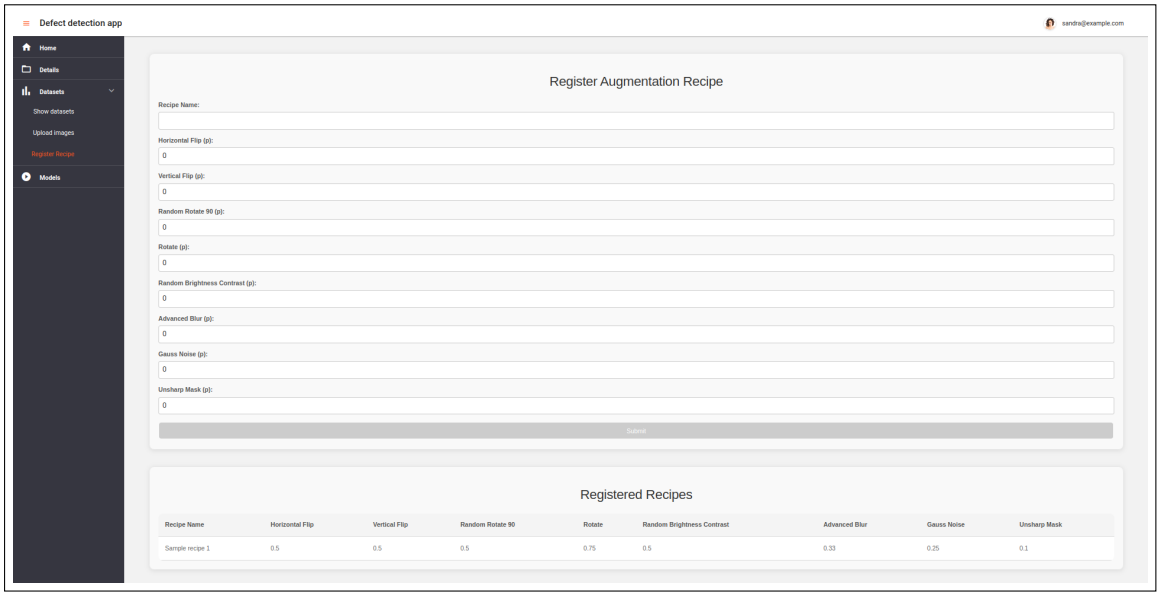

Figura 5.13: Tela de Receitas de Augmentação

g) A **Seção de Listagem de Modelos**, localizada na tela do gerenciador de modelos, oferece ao usuário uma lista de modelos treinados disponíveis para inferência. Nesta listagem, o usuário pode visualizar informações essenciais sobre cada modelo, como o

nome do modelo, precisão *top-1*, *dataset* de treinamento utilizado, receita de aumento aplicada (se houver), descrição do modelo e arquitetura do modelo.

Além disso, há um botão de ação ao lado de cada modelo, permitindo que o usuário exclua permanentemente modelos específicos do diretório. Essa ação removerá o modelo e seus artefatos associados. No entanto, as informações do experimento relacionadas ao modelo excluído ainda estarão disponíveis no serviço de rastreamento de experimentos e precisarão ser excluídas manualmente, se desejado. A Figura 5.14 ilustra o componente seção de listagem de modelos.

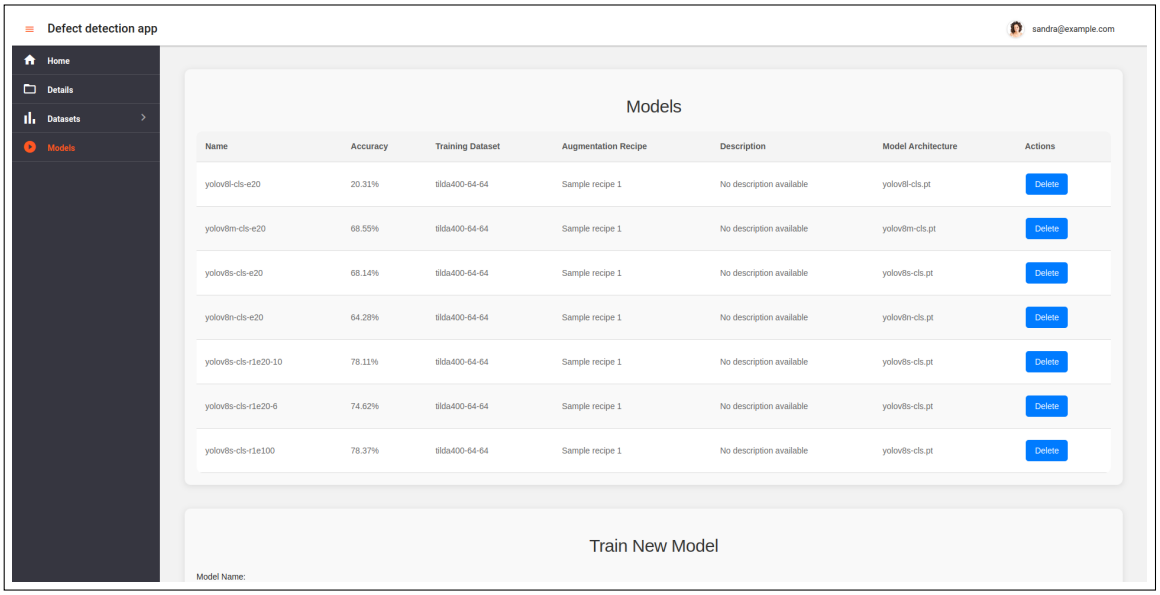

Figura 5.14: Seção de Listagem de Modelos

h) A **Seção de Treinamento de Modelos**, dentro da tela do gerenciador de modelos, permite ao usuário treinar novos modelos que estarão disponíveis através do serviço de diretório de modelos. Nesta seção, o usuário pode configurar vários aspectos do processo de treinamento do modelo.

Em relação ao modelo, o usuário pode definir o nome do modelo, selecionar a arquitetura desejada e especificar o número de épocas para o treinamento. Quanto ao *dataset*, o usuário pode escolher qual *dataset* utilizar (pelo nome), optar por aplicar uma receita opcional de aumento de dados e definir a proporção de divisão entre treino, validação e teste. Além disso, o usuário pode configurar opcionalmente uma semente para o processo de treinamento e aumento de dados, assegurando a reprodutibilidade dos resultados.

Após o usuário configurar essas opções, as informações são enviadas para o serviço de *dataset* e para o serviço de modelos para processamento. Ao término do treinamento, o modelo estará disponível através do serviço de diretório de modelos. A seção de treinamento de modelos é ilustrada na Figura 5.15.

i) A **Tela de Listagem de Experimentos** é voltada para os analistas de dados e pessoal responsável pela confecção dos modelos. Esta oferece uma visão ampla de todas as informações relacionadas aos modelos que foram treinados a partir da tela de treina-

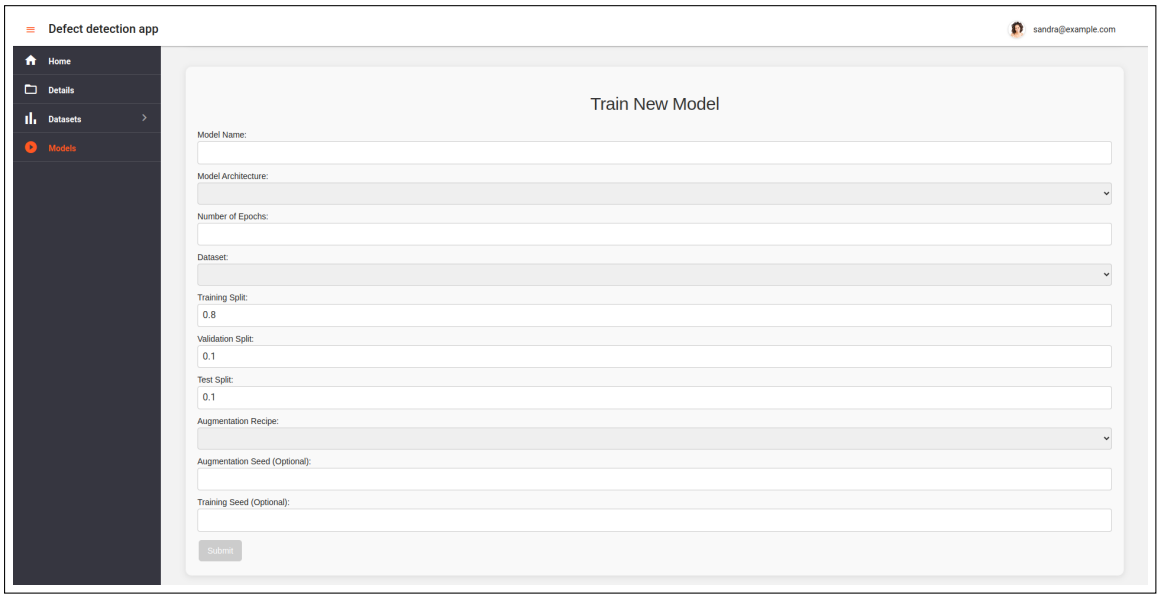

Figura 5.15: Seção de Treinamento de Modelos

mento de modelos, mostrada na Figura 5.15. Esta seção utiliza o *framework* MLflow para armazenar e gerenciar informações sobre cada execução ("*run*") de treinamento.

Nesta tela, o usuário pode acessar dados como a duração de cada execução, os parâmetros do modelo, as métricas coletadas durante o treinamento e os artefatos gerados. Esses artefatos incluem os pesos do modelo para cada execução e metadados sobre o ambiente de treinamento, garantindo que os experimentos sejam reprodutíveis.

Essa apresentação organizada dos dados experimentais permite que os usuários revisem e analisem o desempenho de diferentes modelos, comparem resultados e selecionem os modelos mais adequados para implantação ou desenvolvimento adicional. A tela mencionada é demonstrada na Figura 5.16

j) A **Tela de Comparação de Execuções** é projetada para visualizar e comparar métricas e parâmetros de diferentes execuções de treinamento. Esta tela utiliza-se também do *framework* MLFlow para oferecer ferramentas como gráficos de dispersão e gráficos de coordenadas paralelas, permitindo que os usuários analisem de forma eficaz o desempenho e as características de várias execuções lado a lado.

Essas visualizações permitem que os usuários identifiquem tendências, correlações e *outliers* entre diferentes execuções, facilitando a avaliação de quais modelos ou configurações de treinamento produzem os melhores resultados. Ao comparar essas métricas e parâmetros, os usuários podem tomar decisões informadas sobre a seleção de modelos e otimizações futuras. A Figura 5.17 mostra a tela de comparação de execuções.

As principais telas que compõem a plataforma permitem que os profissionais responsáveis acessem os serviços de IQ disponíveis, obtenham maior controle sobre as operações e desenvolvam suas próprias soluções de forma mais ágil e eficiente.

Para facilitar o desenvolvimento, a manutenção e futuras modificações, foi utilizado

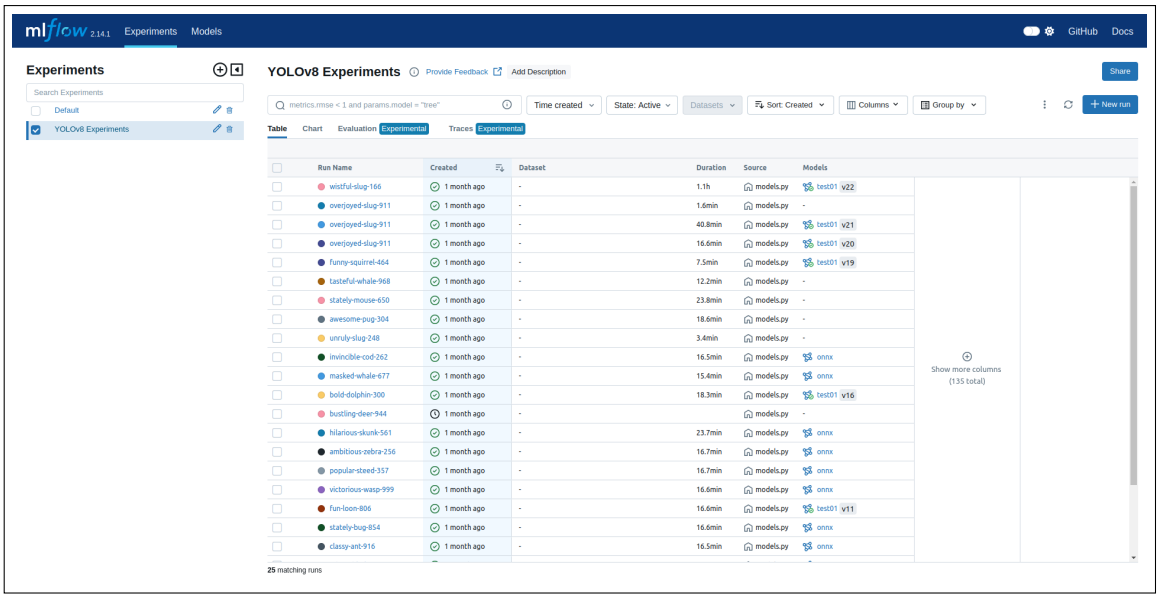

Figura 5.16: Tela de Listagem de Experimentos

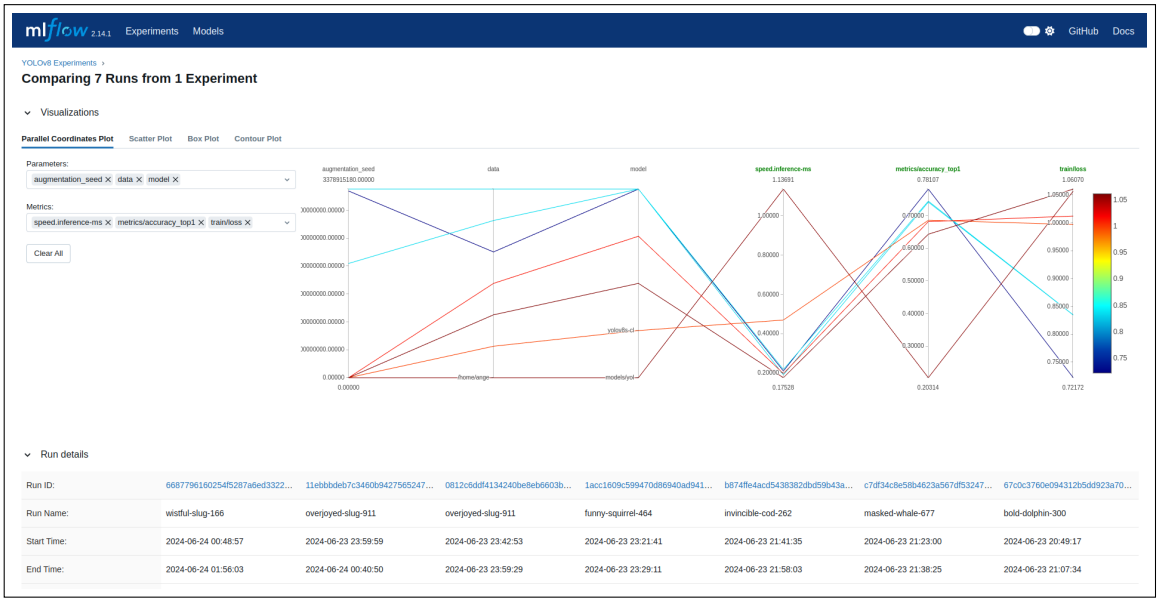

Figura 5.17: Tela de Comparação de Execuções

um template[2] na construção das telas. Essa abordagem permite maior organização do código e reutilização de componentes, agilizando o processo de desenvolvimento.

Atualmente, o módulo de autenticação de usuário ainda não foi implementado. Por esse motivo, as telas exibem apenas um usuário genérico padrão. Essa limitação, no entanto, não compromete os testes iniciais do sistema, uma vez que a autenticação pode ser adicionada em fases futuras do desenvolvimento.

Além disso, como mencionado na Subseção 4.3.1, alguns componentes ilustrados na Figura 4.7 ainda não foram implementados em código, como o Componente de Inteligência, *Gateway* e Dispositivos, bem como as classes correspondentes do diagrama de classes apresentado na Figura 4.13. Isso ocorre porque o presente trabalho não foi apoiado no uso de equipamento industrial, como câmeras, *encoders* de rotação e uma infraestrutura de redes operacionais. Todos os testes foram realizados localmente, utilizando arquivos de *feeds* de vídeo de inspeção simulados.

Apesar dessas limitações, os demais componentes foram implementados com sucesso. Há perspectivas futuras para a implementação dos módulos restantes, em um ambiente operacional real em que seja possível realizar este tipo de teste de automação.

[2] `https://js.devexpress.com/Angular/Templates/UITemplates/`

Capítulo 6

Conclusão

Nesta seção de conclusão, serão revisadas as contribuições alcançadas ao longo desta dissertação e discutidas as principais limitações observadas durante seu desenvolvimento. Em seguida, serão sugeridos possíveis caminhos para futuras pesquisas, com destaque para abordagens que não foram exploradas aqui, mas que podem ser investigadas em estudos posteriores. Por fim, são mencionados os artigos produzidos e publicados concomitantemente à elaboração desta dissertação, ao longo do programa.

6.1 Contribuições da Dissertação

Recapitulando os objetivos específicos apresentados na Seção 1.2, agora serão destacadas as principais contribuições desta dissertação:

1. **Investigar Técnicas de Aprendizado Profundo Estado da Arte para Detecção em Imagens**: O modelo YOLO, sua arquitetura, algoritmo e linha evolutiva são apreciados na Seção 2.1.3. Outros modelos e abordagens são introduzidas no Capítulo 3 sobre trabalhos relacionados, como o método PTIT e técnicas propostas por Jing et al. e Sandhya et al.
2. **Elaborar Algoritmos de Detecção e Refinamento**: Foram formalizados dois algoritmos específicos para a detecção de defeitos: a heurística de detecção em grelha, que segmenta a imagem original em *patches* e realiza a classificação de defeitos localmente; e o refinamento em *ripple*, que explora o contexto espacial local dos defeitos ao executar uma segunda classificação nas células vizinhas aos primeiros positivos, utilizando um limiar de confiança reduzido para a mesma classe do vizinho de origem. Essa abordagem de refinamento considera que uma célula defeituosa tem maior probabilidade de possuir vizinhos também com defeito. A Seção 4.2 apresenta o detalhamento desses algoritmos e alguns resultados práticos obtidos experimentalmente.
3. **Documentar Processo de Preparação do *Dataset* para Treinamento dos Modelos**: Foi detalhado o processo de preparação do *dataset* utilizado no treinamento dos modelos de detecção, abrangendo as etapas de aquisição, rotulação, estruturação e aumento de dados, conforme descrito na Seção 4.4. Os experimentos foram conduzidos utilizando o *dataset* TILDA 400 (64x64 *patches*), uma versão derivada do

TILDA original, que está disponível publicamente[1] para futuras pesquisas na área de detecção de defeitos em tecidos.

4. **Análisar e Comparar Modelos de Detecção de Defeitos**: No Subseção 5.1.1 foi realizada uma análise comparativa entre diferentes modelos de detecção de defeitos, com o objetivo de avaliar sua performance em termos de acurácia e latência.
   A comparação incluiu modelos YOLOv5 (*nano*, *small*, *medium*) e YOLOv8 (*nano*, *small*, *medium*), além dos modelos propostos por Jing et al. e Sandhya et al., baseados em redes neurais pré-treinadas LeNet5 e AlexNet. Os resultados evidenciam a superioridade do YOLOv8 em termos de acurácia e latência, conforme ilustrado na Figura 5.1. O modelo YOLOv8 *medium* obteve uma acurácia de 90,35% e tempo de inferência de 0,5ms (GPU Tesla P100), superando significativamente a média de 70% de acurácia alcançada por um inspetor humano, como discutido na Subseção 1.1.2. Além disso, o YOLOv8 *small* demonstrou uma velocidade teórica de inspeção de 46,875 m/min com uma única GPU, superando categoricamente a velocidade máxima de inspeção humana, que varia entre 15 e 20 m/min.
5. **Desenvolver uma Plataforma de IQ**: Foi desenvolvida uma plataforma completa capaz de processar imagens em tempo real, identificar e reportar defeitos, criar e gerenciar *datasets*, treinar modelos de aprendizado profundo e rastrear experimentos. A modelagem do *software* é descrita na Subseção 4.3.1 e Seção A.2, com a implementação das várias telas da plataforma ilustradas na Seção 5.2.

Convém afirmar que as contribuições desta dissertação representam avanços relevantes à área de inspeção de qualidade para a indústria têxtil, unindo embasamento teórico e técnico sólido à aplicação de soluções práticas. Desde a análise dos processos produtivos até a implementação de plataformas completas, cada etapa deste trabalho busca, de forma prática e objetiva, aprimorar a eficiência e a confiabilidade no âmbito da detecção de defeitos.

Por fim, a análise dos resultados obtidos em aspectos como a criação de algoritmos especializados e a comparação entre modelos não apenas gerou resultados concretos, mas também abriu caminhos para novas pesquisas e o desenvolvimento de novas aplicações industriais. Desse modo, este trabalho estabelece uma base consistente para o avanço de sistemas automatizados com rigor técnico e embasamento teórico, contribuindo para atender às crescentes demandas da indústria de maneira eficaz.

6.2 Limitações e Trabalhos Futuros

A principal limitação deste trabalho foi a falta de acesso a equipamentos reais durante a pesquisa, o que impossibilitou a validação dos resultados em um ambiente de produção. A implementação prática é fundamental para compreender os desafios operacionais e realizar os ajustes necessários para sua aplicação no contexto industrial.

Um dos principais desafios nesse cenário é a variabilidade dos dispositivos e das condições de operação. Diferenças entre sensores de captura de imagem, capacidade de processamento dos dispositivos, resoluções e taxas de captura, níveis de ruído e restrições na

[1] `https://www.kaggle.com/datasets/angelolmg/tilda-400-64x64-patches`

infraestrutura de rede podem impactar significativamente o desempenho da plataforma. Esses fatores devem ser considerados para garantir uma metodologia robusta e uma solução resiliente às condições reais de uso.

Dessa forma, em trabalhos futuros, pretende-se implantar a plataforma e conduzir testes *in situ*, averiguando e alinhando a adaptabilidade do sistema às particularidades do ambiente industrial, garantindo sua viabilidade prática.

Sobre as métricas obtidas dos modelos testados, estas foram exclusivamente derivadas dos experimentos realizados com o *dataset* TILDA. Embora o TILDA seja uma referência consolidada em diversos estudos na área de detecção de defeitos em tecidos, como discutido na Seção 3.2, sua abrangência é limitada. Para validar e generalizar os resultados, seria necessário conduzir experimentos com outros *datasets*, ampliando a diversidade de cenários e tipos de defeitos. Essa abordagem permitiria avaliar a robustez da plataforma em diferentes condições e aumentar a confiabilidade do sistema em aplicações reais.

A pesquisa também não analisou detalhadamente os *overheads* associados ao processamento de imagem, como a segmentação em *patches*, os algoritmos de detecção baseados em grelha e o refinamento em *ripple*, nem seu impacto sobre os resultados finais. Avaliar o custo computacional dessas etapas e sua influência no desempenho geral do sistema é uma questão relevante para investigações futuras. Além disso, explorar como técnicas de otimização, como *pruning*, podem beneficiar o desempenho seria uma linha promissora para estudos subsequentes.

Ainda no contexto de otimizações, a distribuição da carga de inferência em múltiplas *threads* ou processos, facilitada pela técnica de detecção em grelha, foi identificada como uma oportunidade promissora. No entanto, essa abordagem não foi explorada nesta dissertação, o que abre espaço para aprimoramentos futuros.

É possível comparar a abordagem de classificação por *patches* com a detecção utilizando *bounding boxes*. O artigo *"Classification and tracking of items on a moving conveyor belt using convolutional networks and image processing"* (De Góes et al. 2024), produzido pelo autor e relacionado à presente dissertação, trata da detecção de objetos, mas não realiza comparações diretas com outras técnicas, como a PTIT, necessitando de uma análise mais aprofundada. Essa análise contribuirá para uma compreensão mais detalhada de como essas estratégias afetam as métricas de desempenho, ajudando a identificar a técnica que proporciona os melhores resultados no contexto da inspeção de defeitos em tecidos, além de avaliar qual delas possui maior potencial de aceleração por meio de paralelismo.

Em relação às licenças, há limitações para transformar a aplicação desenvolvida em um produto comercial, principalmente devido à licença AGPL-3.0 dos modelos YOLO disponibilizados pela Ultralytics. Para aplicações nesse contexto, modelos alternativos como YOLOv4 *Darknet*, EfficientNet e ResNet oferecem licenças mais permissivas.

Por fim, como discutido na Subseção 5.1.2, há uma preocupação com o problema inerente de falsos positivos observado no algoritmo de detecção em grelha. Assim, pretende-se investigar possíveis estratégias para mitigar esse problema em pesquisas futuras.

6.3 Publicações

Como mencionado na seção anterior, durante o desenvolvimento desta dissertação foi publicado o artigo *"Classification and tracking of items on a moving conveyor belt using convolutional networks and image processing"* (De Góes et al. 2024) na *International Conference on Artificial Intelligence, Computer, Data Sciences and Applications* 2024 (ACDSA 2024), organizada pela IEEE em parceria com a Universidade de Seychelles.

O artigo descreve o desenvolvimento de um protótipo de visão computacional e *deep learning* voltado para otimização do processo de *checkout* em supermercados. A solução utiliza uma câmera suspensa sobre uma esteira móvel para detectar, classificar e contar itens em tempo real. Tecnologias como YOLOv4 *tiny*, YOLOv5 *small*, OpenCV e Roboflow foram combinadas para construir um sistema capaz de alcançar 77% de precisão média (mAP@[0.5:0.95]) na detecção de dois itens distintos, com base em um *dataset* híbrido de imagens coletadas *in vitro* e geradas por simulação.

Além disso, foi desenvolvida uma interface gráfica que permite ajustar hiperparâmetros e monitorar o processamento do sistema, destacando o potencial dessa abordagem para reduzir custos operacionais de *checkout* em estabelecimentos de médio e grande porte, bem como melhorar a experiência do usuário final.

Referências Bibliográficas

Angelov, Samuil, Jos J. M. Trienekens & Paul Grefen (2008), Towards a method for the evaluation of reference architectures: Experiences from a case, *em* R.Morrison, D.Balasubramaniam & K.Falkner, eds., 'Software Architecture', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 225–240.

Angelov, Samuil, Paul Grefen & Danny Greefhorst (2012), 'A framework for analysis and design of software reference architectures', *Information and Software Technology* **54**(4), 417–431.
URL: *https://www.sciencedirect.com/science/article/pii/S0950584911002333*

Arikan, Cihat (2019), 'Developing an intelligent automation and reporting system for fabric inspection machines', *TEKSTİL VE KONFEKSİYON* **29**.

Ashraf, Rehan, Yasir Ijaz, Muhammad Asif, Khurram Zeeshan Haider, Toqeer Mahmood & Muhammad Owais (2022), 'Classification of woven fabric faulty images using convolution neural network', *Mathematical Problems in Engineering* **2022**(1), 2573805.

Bass, L., P. Clements & R. Kazman (2012), *Software Architecture in Practice: Software Architect Practice_c3*, SEI Series in Software Engineering, Pearson Education.
URL: *https://books.google.com.br/books?id=-II73rBDXCYC*

Bochkovskiy, Alexey, Chien-Yao Wang & Hong-Yuan Mark Liao (2020), 'Yolov4: Optimal speed and accuracy of object detection'.

Booch, G. & J. Rumbaugh (2006), *UML: guia do usuário*, Elsevier.
URL: *https://books.google.com.br/books?id=ddWqxcDKGF8C*

Bosch, Jan (1999), Design and use of software architectures, pp. 404–404.

Clements, P. (2003), *Documenting Software Architectures: Views and Beyond*, SEI series in software engineering, Addison-Wesley.
URL: *https://books.google.com.br/books?id=ASc9HYPkr4sC*

Czimmermann, Tamás, Gastone Ciuti, Mario Milazzo, Marcello Chiurazzi, Stefano Roccella, Calogero Maria Oddo & Paolo Dario (2020), 'Visual-based defect detection and classification approaches for industrial applications—a survey', *Sensors* **20**(5), 1459.

De Góes, Angelo Leite Medeiros, Adrião Duarte Dória Neto, Matheus Gomes Diniz Andrade & Thiago Theiry De Oliveira (2024), Classification and tracking of items on a moving conveyor belt using convolutional networks and image processing, *em* '2024 International Conference on Artificial Intelligence, Computer, Data Sciences and Applications (ACDSA)', pp. 1–6.

de Morais Barroca Filho, Itamir (2019), Architectural design of IoT-based healthcare applications, Tese (doutorado em ciência da computação), Centro de Ciências Exatas e da Terra, Universidade Federal do Rio Grande do Norte, Natal.
URL: *https://repositorio.ufrn.br/handle/123456789/26767*

de Oliveira, Tainá Caionara & Jandir Ferrera de Lima (2017), 'A distribuição espacial da indústria têxtil no estado do paraná', *Revista da FAE* **20**(1), 171–184.
URL: *https://revistafae.fae.edu.br/revistafae/article/view/170*

DFG Working Group Texture Analysis (1996), 'Tilda textile texture-database', `https://lmb.informatik.uni-freiburg.de/resources/datasets/tilda.en.html`. Version 1.0.

Dlamini, Sifundvolesihle, Chih-Yuan Kao, Shun-Lian Su & Chung-Feng Jeffrey Kuo (2022), 'Development of a real-time machine vision system for functional textile fabric defect detection using a deep yolov4 model', *Textile Research Journal* **92**(5-6), 675–690.

Drury, C. G. & M. A. Sinclair (1983), 'Human and machine performance in an inspection task', *Human Factors* **25**, 391–399.

Ge, Zheng, Songtao Liu, Feng Wang, Zeming Li & Jian Sun (2021), 'Yolox: Exceeding yolo series in 2021'.
URL: *https://arxiv.org/abs/2107.08430*

Gomaa, H. (2011), *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*, Cambridge University Press.
URL: *https://books.google.com.br/books?id=j4GLAQAACAAJ*

Gonzalez, Dibet Garcia, Yusbel Chavez Castilla, Somayeh Shaharadaby, Ana Mackay, Lúcia Soares, Pedro Guimarães, Francisco Morais, Joel Puga, Filipe Meneses, Adriano Moreira, Miguel Machado, Telmo Adão, Luís Magalhães & Miguel Angel Guevara López (2022), 'A ubiquitous service-oriented automatic optical inspection platform for textile industry', *Procedia Computer Science* **196**, 217–225. International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies 2021.
URL: *https://www.sciencedirect.com/science/article/pii/S1877050921022316*

Gonçalves, Enyo José Tavares & Mariela Inés Cortés (2015), *Análise e Projeto de Sistemas*, Computação, 3ª edição, EdUECE, Fortaleza, CE.

Group, Object Management (2017), 'Unified modeling language specification 2.5.1'.
URL: *https://www.omg.org/spec/UML/2.5.1/PDF*

Guedes, G.T.A. (2018), *UML 2 - Uma Abordagem Prática - 3ª Edição*, Novatec Editora.
URL: *https://books.google.com.br/books?id=RUdLDwAAQBAJ*

Jing, Jun-Feng, Hao Ma & Huan-Huan Zhang (2019), 'Automatic fabric defect detection using a deep convolutional neural network', *Coloration Technology* **135**(3), 213–223.

Jocher, Glenn (2020), 'Ultralytics yolov5'.
URL: *https://github.com/ultralytics/yolov5*

Jocher, Glenn, Ayush Chaurasia & Jing Qiu (2023), 'Ultralytics yolov8'.
URL: *https://github.com/ultralytics/ultralytics*

Jun, Xiang, Jingan Wang, Jian Zhou, Shuo Meng, Ruru Pan & Weidong Gao (2021), 'Fabric defect detection based on a deep convolutional neural network using a two-stage strategy', *Textile Research Journal* **91**(1-2), 130–142.

Krizhevsky, Alex, Ilya Sutskever & Geoffrey E Hinton (2012), Imagenet classification with deep convolutional neural networks, *em* F.Pereira, C.Burges, L.Bottou & K.Weinberger, eds., 'Advances in Neural Information Processing Systems', Vol. 25, Curran Associates, Inc.
URL: *https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf*

Li, Chuyi, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei & Xiaolin Wei (2022), 'Yolov6: A single-stage object detection framework for industrial applications'.
URL: *https://arxiv.org/abs/2209.02976*

Lin, Guijuan, Keyu Liu, Xuke Xia & Ruopeng Yan (2022), 'An efficient and intelligent detection method for fabric defects based on improved yolov5', *Sensors* **23**(1), 97.

Pereira, Gislaine de Souza (2009), 'Materiais e processos têxteis', Material didático. Acesso em [data de acesso].
URL: *https://wiki.ifsc.edu.br/mediawiki/images/temp/0/07/20090218180450!MPTEX6.pdf*

Redmon, Joseph & Ali Farhadi (2016), 'YOLO9000: better, faster, stronger', *CoRR* **abs/1612.08242**.
URL: *http://arxiv.org/abs/1612.08242*

Redmon, Joseph & Ali Farhadi (2018), 'Yolov3: An incremental improvement', *arXiv preprint arXiv:1804.02767* .

Redmon, Joseph, Santosh Divvala, Ross Girshick & Ali Farhadi (2015), 'You only look once: Unified, real-time object detection'.
URL: *https://arxiv.org/abs/1506.02640*

Sandhya, N, Nihal Mathew Sashikumar, M Priyanka, Sebastian Maria Wenisch & Kunaraj Kumarasamy (2021), 'Automated fabric defect detection and classification: a deep learning approach', *Text Leath Rev* **4**, 315–335.

Santos, Beatrice Paiva, Agostinho Alberto, Tânia Daniela Felgueiras Miranda Lima & Fernando Manuel Bigares Charrua-Santos (2018), 'Industry 4.0: Challenges and opportunities', *Revista Produção e Desenvolvimento* **4**(1), 111–124.
URL: *https://revistas.cefet-rj.br/index.php/producaoedesenvolvimento/article/view/e316*

Schneider, Yurgen, Cleber Ferreira Shimizu, Leandro dos Santos Canestraro, Mateus Eduardo Braz & Cassiana Fagundes da Silva (2020), 'Proposta de software de controle de processos da indústria de fabricação de uniformes', *Inova+ Cadernos de Graduação* .
URL: *http://app.fiepr.org.br/revistacientifica/index.php/inovamais/article/view/592*

Sengottuvelan, P, Amitabh Wahi & A Shanmugam (2008), 'Automatic fault analysis of textile fabric using imaging systems', *Research Journal of Applied Sciences* .

Sommerville, I. (2011), *Software Engineering*, International Computer Science Series, Pearson.
URL: *https://books.google.com.br/books?id=l0egcQAACAAJ*

Souza, Guilherme Dilio de & Luís Eduardo de Campo (2023), Prototipagem de interface de sistema de gestão: estudo de caso de controle de estoque em indústria de confecção, Trabalho de conclusão de curso, Faculdade de Tecnologia de Franca - "Dr. Thomaz Novelino", Franca.
URL: *https://ric.cps.sp.gov.br/handle/123456789/17589*

Tiwana, A. (2013), *Platform Ecosystems: Aligning Architecture, Governance, and Strategy*, Morgan Kaufmann.
URL: *https://books.google.com.br/books?id=IYDhAAAAQBAJ*

Unhelkar, B. (2018), *Software Engineering with UML*, CRC Press.
URL: *https://books.google.com.br/books?id=fZ5rtAEACAAJ*

Vakili, Meysam, Mohammad Ghamsari & Masoumeh Rezaei (2020), 'Performance analysis and comparison of machine and deep learning algorithms for iot data classification', *CoRR* **abs/2001.09636**.
URL: *https://arxiv.org/abs/2001.09636*

van Lamsweerde, A. (2014), *Requirements Engineering: From System Goals to UML Models to Software Specifications*, Wiley.
URL: *https://books.google.com.br/books?id=qC1EDwAAQBAJ*

Wang, Chien-Yao, Alexey Bochkovskiy & Hong-Yuan Mark Liao (2022), 'Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors', *arXiv preprint arXiv:2207.02696* .

Wang, Chien-Yao, I-Hau Yeh & Hong-Yuan Mark Liao (2021), 'You only learn one representation: Unified network for multiple tasks'.
URL: *https://arxiv.org/abs/2105.04206*

Wang, Hongyu, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang & Furu Wei (2022), 'Deepnet: Scaling transformers to 1,000 layers'.
URL: *https://arxiv.org/abs/2203.00555*

Yu, Kai, Wentao Lyu, Xuyi Yu, Chenqun Wang, Weiqiang Xu, Qing Guo & Di Zhou (2022), 'Fa-Yolo: A High-Precision and Efficient Method for Fabric Defect Detection in Textile Industry', *SSRN Electronic Journal* .
URL: *https://ssrn.com/abstract=4272230*

Zhao, Shuxuan, Li Yin, Jie Zhang, Junliang Wang & Ray Zhong (2020), 'Real-time fabric defect detection based on multi-scale convolutional neural network', *IET Collaborative Intelligent Manufacturing* **2**(4), 189–196.
URL: *https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cim.2020.0062*

Apêndice A

Informações Adicionais

Neste capítulo, serão apresentadas informações complementares sobre o projeto da RA proposta, especificações dos casos de uso do *software* implementado e a análise dos modelos estudados. Embora essas informações não tenham sido incluídas no corpo principal do texto por não acrescentarem dados essenciais para o entendimento central da dissertação, sua apresentação aqui visa garantir a completude das informações. Além disso, no caso da Seção A.3, busca-se esclarecer a origem de algumas métricas discutidas na Subseção 5.1.1.

A.1 Especificação da Arquitetura de Referência

A.1.1 Camada de Percepção

A Camada de Percepção é responsável por coletar dados relevantes à operação têxtil e é composta pelos seguintes componentes:

1. **Dispositivos:** Componentes de *hardware* utilizados na coleta de dados e configuração do ambiente operacional têxtil. Incluem câmeras, sensores de velocidade, de movimento, de tensão, RFID, NFC, *encoders* de rotação, *hardware* de iluminação, entre outros.
2. **Componente de *Gateway*:** Componente de *software* responsável por receber dados dos dispositivos e transmiti-los para a Camada *Middleware*, além de transmitir dados e comandos das camadas superiores para os dispositivos.

Os serviços e suas respectivas funções na Camada de Percepção estão detalhados na Tabela A.1.

Tabela A.1: Componentes e Serviços da Camada de Percepção

Nome do Componente	Nome do Serviço	Função
Dispositivos	Câmeras	Capturam imagens na operação. Podem ser câmeras digitais ou industriais, como modelos *line scan*, CCD ou CMOS, etc.
	Sensores de Velocidade	Monitoram velocidade de esteira ou objetos na operação, a fim de promover controle e sincronização dos processos.
	Encoder de Rotação	Medem o ângulo e/ou a velocidade de rotação dos eixos para controle de rolagem. Por exemplo, em processos de revisão assistida, tecelagem e malharia.
	Hardware de Ilumina-ção	Garante a iluminação adequada. Por exemplo, em operações de revisão autônoma, utilizando LEDs ou fontes de luz controláveis para eliminar sombras e melhorar a qualidade das imagens capturadas.
	Sensores de Movimento	Detectam movimentações anômalas ou interrupções no fluxo de fiação ou tecelagem, mantendo a precisão dos processos.
	Sensores RFID e NFC	Sensores de localização utilizados no gerenciamento de estoque e cadeia de suprimentos. Por exemplo, para rastrear o fluxo de produtos ao longo da cadeia de suprimentos.
	Sensores de Tensão	Monitoram a tensão aplicada nos fios ou tecidos para evitar rupturas ou deformação, garantindo a uniformidade do processo.

Componente de *Gateway*	Serviço de Rede	Define o método de remapeamento do espaço de endereços IP dos dispositivos (e.g., NAT, PAT, *gateway* de nível de aplicação).
	Serviço de Envio de Dados Brutos	Envia os dados brutos para a camada *middleware*. Durante este envio, ele localiza o serviço de recepção de dados do Coletor de Dados IoT utilizando o serviço de descoberta.
	Serviço de Filtro	Responsável por eliminar possíveis ruídos dos sinais medidos pelos dispositivos. Aplica o filtro de acordo com a característica do sinal (por exemplo, sensor de velocidade e filtro passa-baixa).
	Serviço de Recepção de Dados Brutos	Recebe os dados brutos dos dispositivos. Utiliza o Serviço de Autorização para verificar se o dispositivo que está enviando dados é autorizado. Utiliza o Serviço de Filtro para eliminar ruídos nos dados recebidos.

A.1.2 Camada *Middleware*

A Camada *Middleware* é responsável por receber os dados dos sensores e de imagem da camada de dispositivos, filtrá-los, processá-los e persistí-los, tornando-os disponíveis para a Camada de Serviços (de Morais Barroca Filho 2019). Esta camada é composta pelos seguintes componentes:

1. **Componente Coletor de Dados IoT:** Recebe os dados brutos enviados pelo Componente de *Gateway*, processa e transforma esses dados em formatos compreensíveis para o Componente de Inteligência. Utilizado quando se está lidando com dados brutos que não podem ser utilizados diretamente.
2. **Componente de Inteligência:** Recebe dados do Componente Coletor de Dados IoT, gerencia informações associadas a sessões específicas, e persiste essas informações em diretórios apropriados, tornando-as disponíveis para a Camada de Serviços.

Os serviços e suas respectivas funções na Camada de *Middleware* estão detalhados na Tabela A.2.

Tabela A.2: Componentes e Serviços da Camada *Middleware*

Nome do Componente	Nome do Serviço	Função
Componente Coletor de Dados IoT	Serviço de Recepção de Dados	Recebe dados no Componente Coletor de Dados IoT. Verifica a autorização do componente de origem e envia os dados para o Serviço de Transformação.
	Serviço de Transformação de Dados	Converte dados brutos em formatos compreensíveis pelo Componente de Inteligência (e.g., JPEG, PNG, JSON). Ele utiliza o serviço de formatação de dados para analisar e formatar os dados recebidos. Além disso, pode dividir dados maiores em partes menores, como imagens em alta resolução, para melhor ingestão pelos serviços.
	Serviço de Envio de Dados	Envia os dados para o Componente de Inteligência. Durante o envio, ele localiza o Serviço de Recebimento de Dados IoT do componente de Inteligência utilizando o Serviço de Descoberta.
Componente de Inteligência	Serviço de Recepção de Dados IoT	Recebe dados do Componente Coletor de Dados IoT, verifica a autorização do componente de origem usando o Serviço de Autorização e envia os dados para o Serviço de Perfil de Sessão.

	Serviço de Perfil de Sessão	Gerencia sessões e atribui *stream* de dados a sessões específicas. Uma sessão compreende toda a informação gerada durante um período específico de operação.
	Serviço de Persistência de Informações	Carrega fluxo de dados de sensores e câmeras em diretórios específicos às sessões, permitindo que esses dados sejam apresentados e consumidos pela Camada de Serviços.

A.1.3 Camada de Serviço

A Camada de Serviços é responsável por estabelecer um conjunto de operações disponíveis relacionadas ao consumo dos dados; regras de negócio de processos distintos, como algoritmos de visão e inferência; e acesso a informações de sessão pelas aplicações e demais serviços. Ela centraliza o acesso ao processamento no sistema, funcionando como uma ponte entre as aplicações na Camada de Aplicação e a Camada *Middleware* (de Morais Barroca Filho 2019). Esta camada é composta pelos seguintes componentes:

1. **Componente de Dados:** Responsável por oferecer serviços relacionados ao gerenciamento e à integridade dos dados do sistema, garantindo sua organização e confiabilidade.
2. **Componente de Processos:** Fornece serviços relacionados aos algoritmos utilizados em processos específicos, incluindo diretório de processos, treinamento de modelos e rastreamento.
3. **Componente de Sessão:** Fornece serviços relacionados a sessões de operação, incluindo monitoramento, gestão de sessões e relatórios.

Os serviços e suas respectivas funções na Camada de Serviço estão detalhados na Tabela A.3.

Tabela A.3: Componentes e Serviços da Camada de Serviço

Nome do Componente	Nome do Serviço	Função
Componente de Dados	Serviço de *Datasets*	Gerencia e preserva os *datasets* utilizados por processos específicos, como o treinamento de algoritmos de aprendizado. Define uma estrutura de diretórios padronizada para o armazenamento das amostras e assegura a conformidade com o esquema de dados estabelecido.
	Serviço de Integridade	Realiza verificações automáticas para assegurar a integridade dos dados armazenados, incluindo a validação da contagem de amostras e a identificação de possíveis inconsistências, como dados corrompidos ou fora do padrão. Essas medidas evitam falhas em etapas críticas, como o treinamento de modelos de aprendizado.
Componente de Processos	Serviço de Diretório de Processos	Armazena e gerencia o acesso a algoritmos de processos específicos. Cada algoritmo implementado é registrado como uma caixa preta; por exemplo, sendo cadastrado seu esquema geral de entrada e saída de dados e artefatos de acesso (*scripts*, arquivos de configuração, documentos, etc).

	Serviço de Treinamento de Modelos	Orquestra e gerencia o treinamento de algoritmos de detecção, classificação, segmentação, entre outros. Usa dados fornecidos através de interface com Serviço de *Datasets*. Pode fornecer modelos para o Serviço de Diretório de Processos.
	Serviço de Rastreamento	Gerencia e armazena dados (parâmetros, métricas e artefatos) de experimentos de treinamento de modelos ou testes com outros algorítmos específicos. Permite registrar, atualizar, consultar e comparar informações sobre experimentos.
Componente de Sessão	Serviço de Sessões	Gerencia e persiste informações relevantes e metadados das sessões de operação. Permite iniciar, pausar, retomar, parar e listar sessões antigas, além de permitir a exclusão dessas informações.
	Serviço de Monitor	Fornece dados em tempo real sobre a sessão atual. Numa operação de inspeção de defeitos, por exemplo, poderia disponibilizar o *feed* de vídeo das câmeras conectadas e contagem dos defeitos encontrados.
	Serviço de Relatórios	Gera relatórios sobre as sessões de operação. Usa dados fornecidos através de interface com Serviço de Sessões. Permite solicitar e baixar relatórios com informações vinculadas a sessões específicas.

A.1.4 Camada de Aplicação

A Camada de Aplicação compreende as principais funcionalidades de sistemas de inspeção baseados em IoT. Essas funcionalidades estão organizadas em dois grupos principais: **aplicações operacionais** e **aplicações de ciência de dados**. Em um fluxo completo de operação, ambos os tipos de aplicações interagem de maneira complementar: as aplicações operacionais são voltadas para os operadores, enquanto as aplicações de ciência de dados são direcionadas a engenheiros e analistas responsáveis por gerenciar e aprimorar as ferramentas inteligentes do sistema. A camada é composta pelos seguintes grupos de aplicações:

1. **Aplicações Operacionais:** Focadas no suporte ao monitoramento e à gestão das sessões de inspeção, essas aplicações facilitam a operação em tempo real e o registro de informações relevantes.
2. **Aplicações de Ciência de Dados:** Voltadas para a análise e manipulação de dados, essas aplicações permitem a criação e o gerenciamento de *datasets*, o rastreamento de experimentos e o treinamento de modelos de aprendizado de máquina.

Os serviços e suas respectivas funções na Camada de Aplicação estão detalhados na Tabela A.4.

Tabela A.4: Componentes e Serviços da Camada de Aplicação

Grupo de Aplicação	Nome da Aplicação	Função
Aplicações Operacionais	Gerenciamento de Sessões	Permite a criação, pausa, parada, salvamento e exclusão de sessões. Permite controle sobre as sessões de operação e facilita o gerenciamento de sessões anteriores.
	Monitoramento de Sessão	Proporciona uma visualização em tempo real do processo, com informações resumidas. Por exemplo, o número de defeitos detectados numa inspeção, velocidade da esteira e gráficos relevantes.

	Geração de Relatórios	Oferece um sumário detalhado das sessões realizadas, possibilitando o *download* de informações operacionais. Por exemplo, defeitos detectados num inspeção, horários de captura, dados de desempenho e tempo total de operação.
Aplicações de Ciência de Dados	Gerenciamento de *Datasets*	Facilita a criação, *upload*, rotulação e manipulação de *datasets*. Permite preparar os dados que serão utilizados no treinamento de modelos de aprendizado de máquina.
	Gerenciamento de Processos	Permite a configuração, visualização e definição de processos e serviços específicos por meio de uma interface que interage diretamente com o Serviço de Diretório de Processos.
	Rastreamento de Experimentos	Proporciona a visualização e comparação entre diferentes experimentos, como treinos de modelos de aprendizado de máquina. Inclui interfaces com gráficos e tabelas comparativas para análise de desempenho.

A.1.5 Camadas de Atributos de Qualidade Transversal

A Camada de Atributos de Qualidade Transversal é responsável por funcionalidades que tornam as aplicações de inspeção baseadas em IoT seguras, disponíveis e eficientes (de Morais Barroca Filho 2019). Seus componentes abordam disponibilidade, desempenho e segurança. É importante enfatizar que, devido à responsabilidade desta camada, ela interage com as camadas de Aplicações, Serviços, *Middleware* e Percepção (de Morais Barroca Filho 2019). É composta pelos seguintes componentes:

1. **Componente de Segurança:** Componente de *software* responsável por proteger dados e informações sensíveis contra acesso não autorizado, enquanto fornece acesso a usuários, sistemas e serviços autorizados (de Morais Barroca Filho 2019). Inclui

serviços de autenticação, autorização, auditoria, detecção de intrusões, segurança da informação e encriptação.

2. **Componente de Interoperabilidade:** Componente de *software* responsável por permitir que as aplicações da industria têxtil baseadas em IoT tenham a capacidade de trocar dados (interoperabilidade sintática) com os dispositivos, bem como interpretar os dados que estão sendo trocados (interoperabilidade semântica) (de Morais Barroca Filho 2019). Ele é composto pelos serviços de formatação de dados, descoberta e drivers.
3. **Componente de Controle e Gerenciamento de Recursos:** Componente de *software* responsável pelo desempenho das aplicações da indústria têxtil baseadas em IoT. Esse desempenho refere-se ao tempo e à capacidade das aplicações de atenderem aos requisitos de tempo (de Morais Barroca Filho 2019). Ele é composto pelos serviços de resposta de eventos, priorização de eventos e limitador de execução.
4. **Componente de Recuperação de Falhas:** Componente de *software* relacionado à disponibilidade e recuperação de falhas (de Morais Barroca Filho 2019) das aplicações de inspeção na industria têxtil baseadas em IoT, composto pelos serviços de redundância, tratamento de exceções, ressincronização de estado e repetição.
5. **Componente de Detecção de Falhas:** Componente de *software* relacionado à disponibilidade e detecção de falhas (de Morais Barroca Filho 2019) das aplicações de inspeção na industria têxtil baseadas em IoT, composto pelos serviços de monitoramento, auto-teste, detecção de exceções e detecção de degradação do modelo.
6. **Componente de Prevenção de Falhas:** Componente de *software* relacionado à disponibilidade e prevenção de falhas (de Morais Barroca Filho 2019) das aplicações de inspeção na industria têxtil baseadas em IoT, composto pelos serviços de prevenção de exceções e remoção de dispositivos.

Os serviços e suas respectivas funções na Camada de Atributos de Qualidade Transversal estão detalhados na Tabela A.5.

Tabela A.5: Componentes e Serviços da Camada de Atributos de Qualidade Transversal

Nome do Componente	Nome do Serviço	Função
Componente de Segurança	Serviço de Autenticação	Verifica a identidade dos usuários, validando credenciais e consultando o banco de dados de usuários.
	Serviço de Autorização	Garante que usuários autenticados tenham permissão para acessar e modificar dados ou serviços.
	Serviço de Auditoria	Registra ações dos usuários, coletando dados sobre entradas, saídas, datas e horários, e armazenando-os em *logs*.

	Serviço de Encriptação	Encripta dados trocados entre os componentes, protegendo-os contra acessos não autorizados. Implementa confidencialidade por meio de técnicas como VPN ou SSL, adicionando proteção além da autorização.
	Serviço de Segurança da Informação	Notifica administradores sobre ataques em andamento, que podem exigir ações. Utiliza o serviço de detecção de intrusão para identificar problemas de segurança e alertar os responsáveis.
	Serviço de Detecção de Intrusões	Monitora o tráfego da rede para identificar intrusões, comparando com assinaturas de comportamento malicioso.
Componente de Interoperabilidade	Serviço de Formatação de Dados	Formata e interpreta dados trocados entre dispositivos e componentes do sistema, garantindo conformidade com padrões como JSON, XML e HL7.
	Serviço de Descoberta	Mantém um diretório de serviços registrados, localizáveis por atributos como tipo, nome, localização ou componente. Permite o registro de novos serviços no sistema.
	Serviço de *Drivers*	Gerencia *drivers* necessários para comunicação entre dispositivos e aplicações. Traduz dados brutos enviados por dispositivos para um formato compreensível pela aplicação, utilizando o serviço de formatação de dados quando necessário.

Componente de Gestão e Controle de Recursos	Serviço de Resposta de Eventos	Garante que eventos sejam enfileirados quando o componente estiver ocupado, processando-os posteriormente em ordem de chegada, sem descartá-los.
	Serviço de Priorização de Eventos	Estabelece prioridades para eventos, garantindo que os mais importantes sejam processados primeiro. Utiliza o serviço de resposta para enfileirar ou ignorar eventos de baixa prioridade, dependendo dos recursos disponíveis.
	Serviço Limitador de Execução	Limita o tempo de execução alocado a componentes para responder a eventos. Caso o tempo seja excedido, o componente é marcado como ocupado e os próximos eventos são processados pelo serviço de resposta.
Componente de Recuperação de Falhas	Serviço de Redundância	Configura redundância em serviços, podendo utilizar redundância ativa (*hot spare*), passiva (*warm spare*) ou em espera (*cold spare*), garantindo continuidade mesmo em caso de falhas.
	Serviço de Tratamento de Exceções	Gerencia exceções, fornecendo informações detalhadas para correção e novas tentativas de operação.
	Serviço de Ressincronização de Estado	Sincroniza o estado dos componentes redundantes. Em configurações ativas, a sincronização ocorre automaticamente com entradas idênticas; em passivas, ocorre por atualizações periódicas enviadas pelo componente ativo.

	Serviço de Repetição	Estabelece e executa o número máximo de tentativas para processar eventos antes de declarar uma falha permanente.
Componente de Detecção de Falhas	Serviço de Monitoramento	Monitora o estado de saúde dos serviços e componentes, detectando falhas ou congestionamentos e alertando administradores.
	Serviço de Auto-Teste	Realiza auto-testes para verificar o funcionamento correto dos serviços e componentes.
	Serviço de Detecção de Exceções	Detecta e registra exceções que alteram o fluxo normal de execução.
	Serviço de Detecção de Degradação de Processo	Monitora e avalia periodicamente a degradação no desempenho de processos específicos ao longo de multiplas sessões, podendo utilizar o Serviço de Monitoramento.
Componente de Prevenção de Falhas	Serviço de Prevenção de Exceções	Previne exceções utilizando técnicas como classes de exceção, tipos abstratos de dados e programação defensiva.
	Serviço de Remoção de Dispositivos	Remove componentes e serviços para evitar falhas, colocando-os no estado "fora de serviço" e registrando a mudança através do Serviço de Monitoramento.

A.2 Especificações dos Casos de Uso

Em sequencia serão datalhados os casos de uso que atendem os requisitos funcionais descritos na Subseção 4.3.1.

A.2.1 Gerenciamento de Sessões

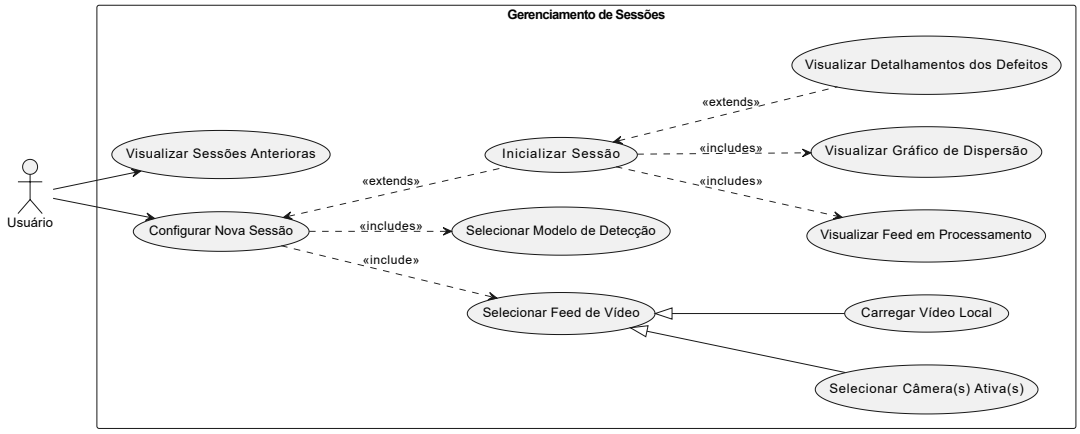

Figura A.1: Diagrama de Caso de Uso para Gerenciamento de Sessões

Tabela A.6: Casos de Uso - Gerenciamento de Sessões

Caso de Uso	**Configurar Nova Sessão**
ID	UC 001
Descrição	Permitir ao usuário configurar os parâmetros para iniciar uma nova sessão, como seleção do *feed* de vídeo e modelo de detecção.
Ator Primário	Usuário
Pré-condição	O usuário deve estar autenticado no sistema.
Cenário Principal	1. O usuário acessa a funcionalidade de configuração de nova sessão. 2. O sistema solicita que o usuário selecione o *feed* de vídeo e o modelo de detecção. 3. O usuário confirma as configurações.
Pós-condição	A nova sessão é configurada e pronta para ser inicializada.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Selecionar *Feed* de Vídeo**
ID	UC 002
Descrição	Permitir ao usuário selecionar a fonte do *feed* de vídeo para processamento, seja câmera(s) ativa(s) ou um vídeo local.
Ator Primário	Usuário
Pré-condição	O caso de uso UC 001 "Configurar Nova Sessão" deve estar em execução.

Cenário Principal	1. O usuário escolhe a funcionalidade de seleção de *feed* de vídeo. 2. O sistema exibe opções de câmera(s) ativa(s) e de *upload* de vídeo local. 3. O usuário realiza a seleção.
Pós-condição	A fonte do *feed* de vídeo é selecionada e associada a sessão.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Selecionar Modelo de Detecção**
ID	UC 003
Descrição	Permitir ao usuário selecionar um modelo de detecção previamente treinado para ser usado na sessão.
Ator Primário	Usuário
Pré-condição	O caso de uso UC 001 "Configurar Nova Sessão" deve estar em execução.
Cenário Principal	1. O usuário acessa a funcionalidade de seleção de modelo de detecção. 2. O sistema exibe a lista de modelos disponíveis. 3. O usuário seleciona o modelo desejado.
Pós-condição	O modelo de detecção selecionado é configurado para a sessão.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Inicializar Sessão**
ID	UC 004
Descrição	Permitir ao usuário inicializar a sessão configurada, executando o processamento do *feed* de vídeo.
Ator Primário	Usuário
Pré-condição	A configuração da nova sessão deve estar concluída.
Cenário Principal	1. O usuário acessa a funcionalidade de inicialização da sessão, visualizando um resumo das configurações executadas. 2. O sistema começa o processamento do *feed* de vídeo.
Pós-condição	A sessão é inicializada e o processamento em tempo real entra em execução.
Cenário Alternativo	Não aplicável.

Caso de Uso	**Visualizar *Feed* em Processamento**
ID	UC 005
Descrição	Exibir o *feed* de vídeo em tempo real com os defeitos detectados destacados.
Ator Primário	Usuário
Pré-condição	A sessão deve estar inicializada (UC 004).
Cenário Principal	1. O sistema exibe o *feed* de vídeo processado em tempo real. 2. Defeitos detectados são destacados na interface de vídeo.
Pós-condição	O *feed* processado é exibido ao usuário.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Visualizar Gráfico de Dispersão**
ID	UC 006
Descrição	Exibir um gráfico de dispersão com as coordenadas dos defeitos detectados ao longo do tecido processado.
Ator Primário	Usuário
Pré-condição	A sessão deve estar inicializada (UC 004).
Cenário Principal	1. O sistema gera o gráfico de dispersão em tempo real as detecções. 2. O gráfico é exibido ao usuário.
Pós-condição	O gráfico de dispersão é exibido ao usuário em tempo real.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Visualizar Detalhamentos dos Defeitos**
ID	UC 007
Descrição	O usuário pode visualizar o detalhamento dos defeitos detectados durante a sessão, incluindo a classe do defeito, imagem, localização e outros dados relevantes.
Ator Primário	Usuário
Pré-condição	O sistema deve ter realizado a detecção de pelo manos um defeito e o usuário deve estar visualizando uma sessão inicializada (UC 004).
Cenário Principal	1. O usuário acessa a tela de detalhamento de defeitos. 2. O sistema exibe informações sobre os defeitos detectados.

Pós-condição	O usuário visualiza os detalhes dos defeitos detectados, como a classe e a localização de cada um.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Visualizar Sessões Anterioras**
ID	UC 008
Descrição	O usuário pode acessar e visualizar detalhes das sessões anterioras para análise ou comparação.
Ator Primário	Usuário
Pré-condição	O usuario deve estar autenticado e o sistema deve ter sessões anteriores registradas.
Cenário Principal	1. O usuário seleciona uma sessão anterior para visualização. 2. O sistema exibe os detalhes da sessão selecionada.
Pós-condição	O usuário visualiza os dados e resultados das sessões anteriores.
Cenário Alternativo	Não aplicável.

A.2.2 Gerenciamento de *Datasets*

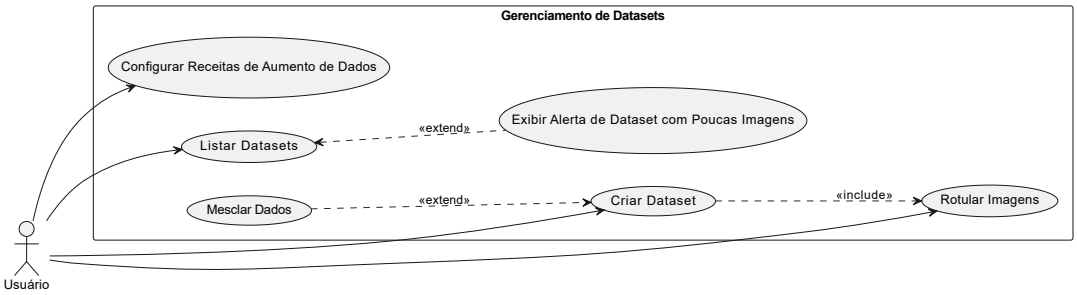

Figura A.2: Diagrama de Caso de Uso para Gerenciamento de *Datasets*

Tabela A.7: Casos de Uso - Gerenciamento de *Datasets*

Caso de Uso	**Criar *Dataset***
ID	UC 009
Descrição	O sistema deve permitir que o usuário crie um novo *dataset*, especificando os atributos, como nome, descrição, classes das imagens e, em especial, a fonte dos dados e os rótulos.
Ator Primário	Usuário

Pré-condição	O usuário deve estar autenticado no sistema.
Cenário Principal	1. O caso de uso inicia quando o usuário acessa a funcionalidade de criação de *dataset*. 2. O usuário escolhe a fonte dos dados, podendo optar por dados já cadastrados no sistema ou realizar o *upload* de novos dados. 3. O usuário cadastra os rótulos para cada imagem. 4. O usuário preenche os atributos necessários para o registro do *dataset*, como nome, descrição e classes. 5. O sistema registra o *dataset* e exibe uma confirmação da operação concluída com sucesso.
Pós-condição	Um novo *dataset* é criado e armazenado no sistema.
Cenário Alternativo	1. O usuário não cadastra os rótulos do *dataset*. 2. O *dataset* é cadastrado como um "processo" com pendências de rotulação.
Caso de Uso	**Rotular Imagens**
ID	UC 010
Descrição	O sistema deve permitir que o usuário rotule as imagens carregadas, associando-as às classes definidas no *dataset*.
Ator Primário	Usuário
Pré-condição	O *dataset* deve ter sido criado ou carregado previamente (UC 009).
Cenário Principal	1. O caso de uso inicia quando o usuário seleciona a funcionalidade de rotulagem de imagens. 2. O sistema exibe as imagens do *dataset* e as classes disponíveis. 3. O usuário associa cada imagem, ou *patch* de imagem, a uma classe. 4. O sistema salva as associações realizadas.
Pós-condição	Todas as imagens carregadas são rotuladas e associadas às classes definidas no *dataset*.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Mesclar Dados**
ID	UC 011

Descrição	O sistema deve permitir que o usuário mescle os dados de dois ou mais *datasets*/processos existentes em um.
Ator Primário	Usuário
Pré-condição	Deve haver resgistrado pelo menos dois *datasets*/processos previamente criados ou carregados (UC 009).
Cenário Principal	1. O caso de uso inicia quando o usuário seleciona a funcionalidade de mesclagem de dados. 2. O sistema exibe uma lista de *datasets*/processos disponíveis. 3. O usuário seleciona os elementos a serem mesclados. 4. O sistema mescla os dados e cria um novo *dataset*/processo.
Pós-condição	Um novo *dataset*/processo contendo os dados combinados é criado e armazenado no sistema.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Configurar Receitas de Aumento de Dados**
ID	UC 012
Descrição	O sistema deve permitir que o usuário configure receitas de aumento de dados, especificando as probabilidades e os tipos de transformações a serem aplicadas às imagens do *dataset*.
Ator Primário	Usuário
Pré-condição	O usuário deve estar autenticado no sistema.
Cenário Principal	1. O caso de uso inicia quando o usuário seleciona a funcionalidade de configuração de receitas de aumento de dados. 2. O sistema exibe os tipos de transformações disponíveis. 3. O usuário seleciona as transformações desejadas e define as probabilidades para cada uma. 4. O sistema salva a receita configurada.
Pós-condição	A receita de aumento de dados é configurada e armazenada no sistema.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Listar *Datasets***
ID	UC 013
Descrição	O sistema deve permitir que o usuário visualize a lista de *datasets* armazenados, com informações como nome, número de imagens e classes.
Ator Primário	Usuário

Pré-condição	O usuário deve estar autenticado no sistema.
Cenário Principal	1. O caso de uso inicia quando o usuário acessa a funcionalidade de listagem de *datasets*. 2. O sistema exibe a lista de *datasets* disponíveis, com suas informações principais.
Pós-condição	O usuário obtém uma visão geral dos *datasets* disponíveis no sistema.
Cenário Alternativo	Não aplicável.

A.2.3 Gerenciamento de Processos

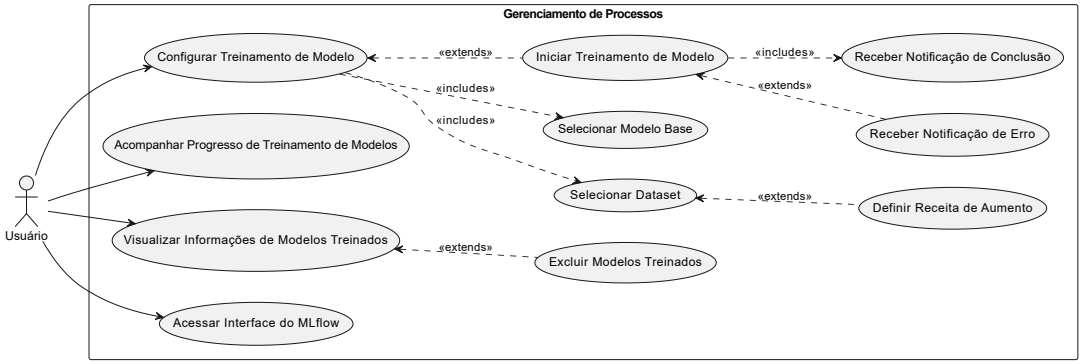

Figura A.3: Diagrama de Caso de Uso para Gerenciamento de Processos

Tabela A.8: Casos de Uso - Gerenciamento de Processos

Caso de Uso	**Configurar Treinamento de Modelo**
ID	UC 014
Descrição	Oferece opções para definir parâmetros do treinamento, como número de épocas, proporções de treino/validação/teste e *seed* de inicialização.
Ator Primário	Usuário
Pré-condição	O usuário deve ter selecionado um modelo base e um *dataset*.

Cenário Principal	1. O caso de uso inicia quando o usuário acessa a configuração de treinamento. 2. O sistema exibe os parâmetros disponíveis. 3. O usuário ajusta os parâmetros conforme necessário. 4. O sistema salva as configurações realizadas.
Pós-condição	As configurações de treinamento são armazenadas e prontas para uso.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Selecionar *Dataset***
ID	UC 015
Descrição	Permite ao usuário escolher um *dataset* existente para configurar o treinamento do modelo.
Ator Primário	Usuário
Pré-condição	O *dataset* deve estar previamente cadastrado no sistema (UC 009).
Cenário Principal	1. O caso de uso inicia quando o usuário acessa a funcionalidade de seleção de *dataset*. 2. O sistema lista os *datasets* disponíveis. 3. O usuário escolhe o *dataset* desejado.
Pós-condição	O *dataset* selecionado é associado à configuração de treinamento.
Cenário Alternativo	1. O usuário não encontra o *dataset* desejado. 2. O sistema orienta a criação de um novo *dataset*.
Caso de Uso	**Definir Receita de Aumento**
ID	UC 016
Descrição	Permite ao usuário configurar uma receita de aumento de dados (*data augmentation*) associada ao *dataset* selecionado.
Ator Primário	Usuário
Pré-condição	O usuário deve ter selecionado um *dataset* previamente.
Cenário Principal	1. O caso de uso inicia após a seleção de um *dataset*. 2. O usuário define os parâmetros da receita de aumento. 3. O sistema armazena a receita configurada.
Pós-condição	A receita de aumento é salva no sistema e associada ao *dataset*.

Cenário Alternativo	1. O usuário opta por usar uma receita existente (UC 012). 2. O sistema aplica a receita selecionada ao *dataset*.
Caso de Uso	**Selecionar Modelo Base**
ID	UC 017
Descrição	Permite ao usuário selecionar o modelo base que será utilizado para o treinamento.
Ator Primário	Usuário
Pré-condição	O arquivo do modelo base deve estar previamente registrado no sistema.
Cenário Principal	1. O caso de uso inicia quando o usuário acessa a funcionalidade de seleção de modelo base. 2. O sistema apresenta os modelos base disponíveis. 3. O usuário seleciona o modelo desejado.
Pós-condição	O modelo base selecionado é associado à configuração de treinamento.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Iniciar Treinamento de Modelo**
ID	UC 018
Descrição	Permite ao usuário iniciar o treinamento do modelo configurado, integrando o *dataset* e a receita de aument previamente selecionados.
Ator Primário	Usuário
Pré-condição	O usuário deve ter configurado o treinamento do modelo (UC 014), selecionado *dataset* (UC 015), modelo base (UC 017) e, opcionalmente, uma receita de aumento (UC 016).
Cenário Principal	1. O caso de uso inicia quando o usuário confirma o início do treinamento. 2. O sistema valida as configurações realizadas. 3. O treinamento é iniciado. 4. O progresso do treinamento é registrado e monitorado.
Pós-condição	O treinamento é iniciado, e o sistema está preparado para gerar notificações de progresso ou erros.

Cenário Alternativo	1. As configurações são inválidas ou incompletas. 2. O sistema exibe mensagens de erro solicitando correções antes de prosseguir.
Caso de Uso	**Receber Notificação de Conclusão**
ID	UC 019
Descrição	Permite ao usuário receber uma notificação ao término do treinamento do modelo, incluindo informações sobre o desempenho obtido.
Ator Primário	Usuário
Pré-condição	O treinamento do modelo deve ter sido iniciado (UC 018) e concluído com sucesso.
Cenário Principal	1. O caso de uso inicia quando o treinamento é concluído. 2. O sistema gera uma notificação contendo detalhes do desempenho do modelo (*e.g.* métricas de acurácia, perda, etc). 3. O usuário visualiza a notificação.
Pós-condição	O usuário é informado sobre o término do treinamento e pode acessar os resultados para análise ou uso posterior.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Receber Notificação de Erro**
ID	UC 020
Descrição	Permite ao usuário ser notificado em caso de erros durante o treinamento do modelo, indicando o motivo e as ações recomendadas.
Ator Primário	Usuário
Pré-condição	Um erro deve ocorrer durante o treinamento do modelo.
Cenário Principal	1. O caso de uso inicia quando um erro é detectado durante o treinamento. 2. O sistema registra os detalhes do erro, por meio de tratamento de excessão. 3. O usuário recebe uma notificação com informações sobre o erro e possíveis soluções.
Pós-condição	O usuário é informado do erro e pode tomar ações corretivas ou revisar as configurações do treinamento.

Cenário Alternativo	Não aplicável.
Caso de Uso	**Acompanhar Progresso de Treinamento de Modelos**
ID	UC 021
Descrição	Permite ao usuário monitorar o progresso do treinamento de um modelo, ou multiplos modelos, em tempo real e paralelamente. Inclui exibição de informações como métricas de desempenho, taxa de erro e status atual do treinamento.
Ator Primário	Usuário
Pré-condição	Um treinamento deve estar em andamento (UC 018).
Cenário Principal	1. O caso de uso inicia quando o usuário acessa a tela de acompanhamento. 2. O sistema exibe informações sobre o progresso do treinamento, como *epochs*, métricas, e tempo restante estimado. 3. O usuário pode visualizar atualizações em tempo real.
Pós-condição	O usuário visualiza o progresso do treinamento em tempo real.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Visualizar Informações de Modelos Treinados**
ID	UC 022
Descrição	Permite ao usuário visualizar informações sobre os modelos previamente treinados, como métricas de desempenho, hiperparâmetros utilizados e outros dados coletados.
Ator Primário	Usuário
Pré-condição	O sistema deve possuir modelos previamente treinados armazenados.
Cenário Principal	1. O caso de uso inicia quando o usuário acessa a listagem de modelos treinados. 2. O sistema exibe as informações de cada modelo, como nome, data de treinamento, métricas, etc. 3. O usuário pode selecionar um modelo para visualizar detalhes específicos.
Pós-condição	O usuário obtém informações sobre os modelos treinados.
Cenário Alternativo	Não aplicável.
Caso de Uso	**Excluir Modelos Treinados**
ID	UC 023

Descrição	Permite ao usuário excluir modelos treinados e artefatos associados.
Ator Primário	Usuário
Pré-condição	O sistema deve possuir modelos treinados armazenados.
Cenário Principal	1. O caso de uso inicia quando o usuário seleciona a opção de exclusão de modelos. 2. O sistema exibe uma lista de modelos disponíveis para exclusão. 3. O usuário seleciona um modelo e confirma a exclusão. 4. O sistema remove o modelo selecionado.
Pós-condição	O modelo é removido do sistema.
Cenário Alternativo	1. O usuário cancela a operação antes de confirmar a exclusão.
Caso de Uso	**Acessar Interface do MLflow**
ID	UC 024
Descrição	Permite ao usuário acessar a interface do MLflow para realizar análise de experimentos, comparação entre modelos e visualização avançada de métricas.
Ator Primário	Usuário
Pré-condição	O sistema deve estar integrado ao MLflow e possuir experimentos registrados.
Cenário Principal	1. O caso de uso inicia quando o usuário seleciona a opção de acessar a interface do MLflow. 2. O sistema redireciona o usuário para a interface do MLflow. 3. O usuário pode navegar pelos experimentos, visualizar métricas e realizar comparações.
Pós-condição	O usuário acessa a interface do MLflow e obtém informações sobre os experimentos registrados.
Cenário Alternativo	Não aplicável.

A.3 Artefatos de Treino e Validação

As Figuras A.4, A.6, A.8, A.10 e A.12 exibem as matrizes de confusão para os modelos YOLOv5n, YOLOv5s, YOLOv5m, YOLOv8n e YOLOv8s, respectivamente. Além disso, as Figuras A.5, A.7, A.9, A.11 e A.13 mostram as curvas de perda de treinamento

e validação, bem como as curvas de acurácia de validação para esses modelos, após o treinamento de 100 épocas.

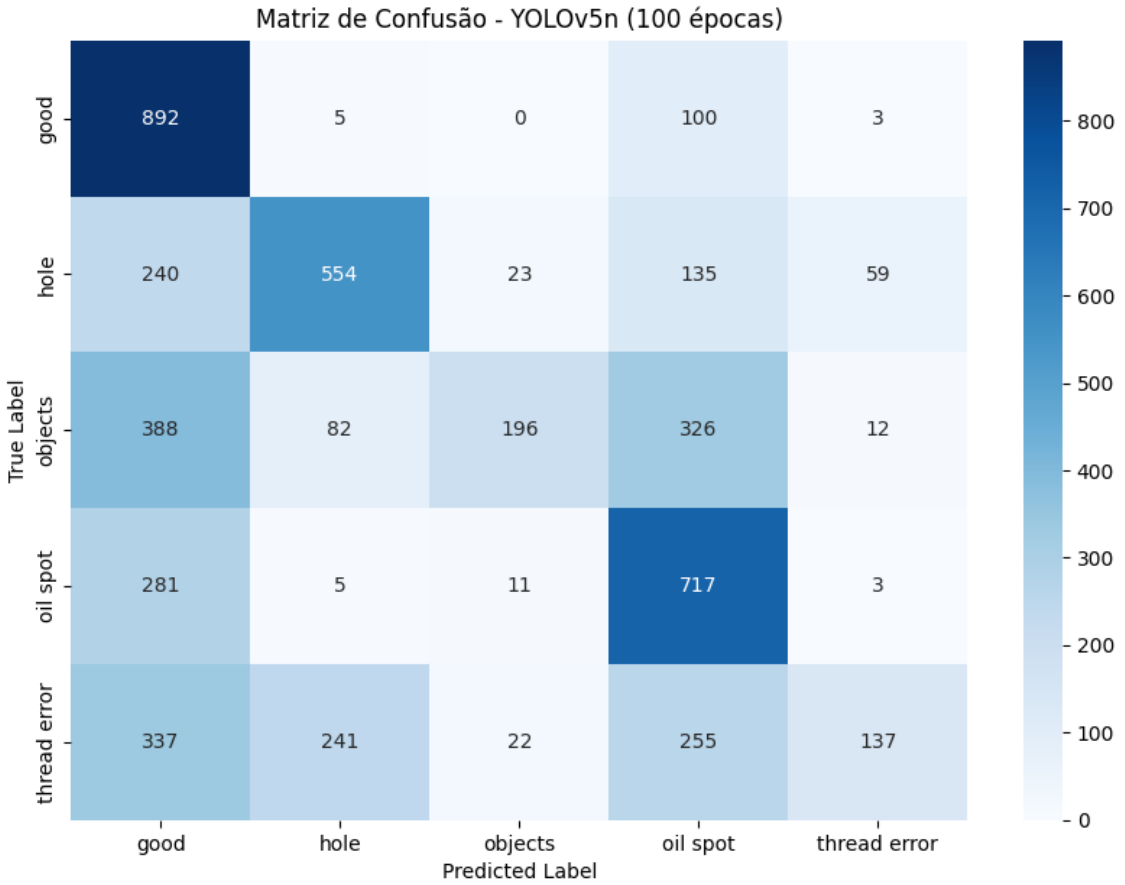

Figura A.4: Matriz de Confusão - Modelo YOLOv5n (100 Épocas)

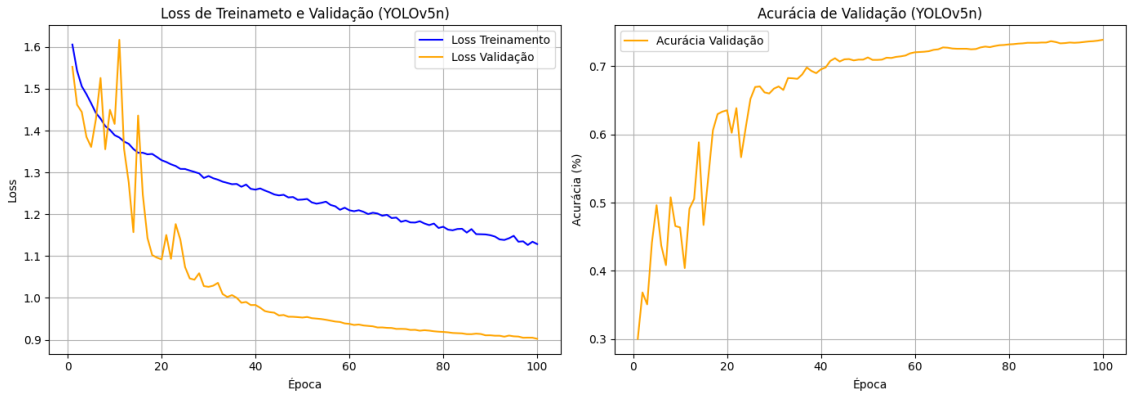

Figura A.5: Curva de Loss e Acurácia para Modelo YOLOv5n

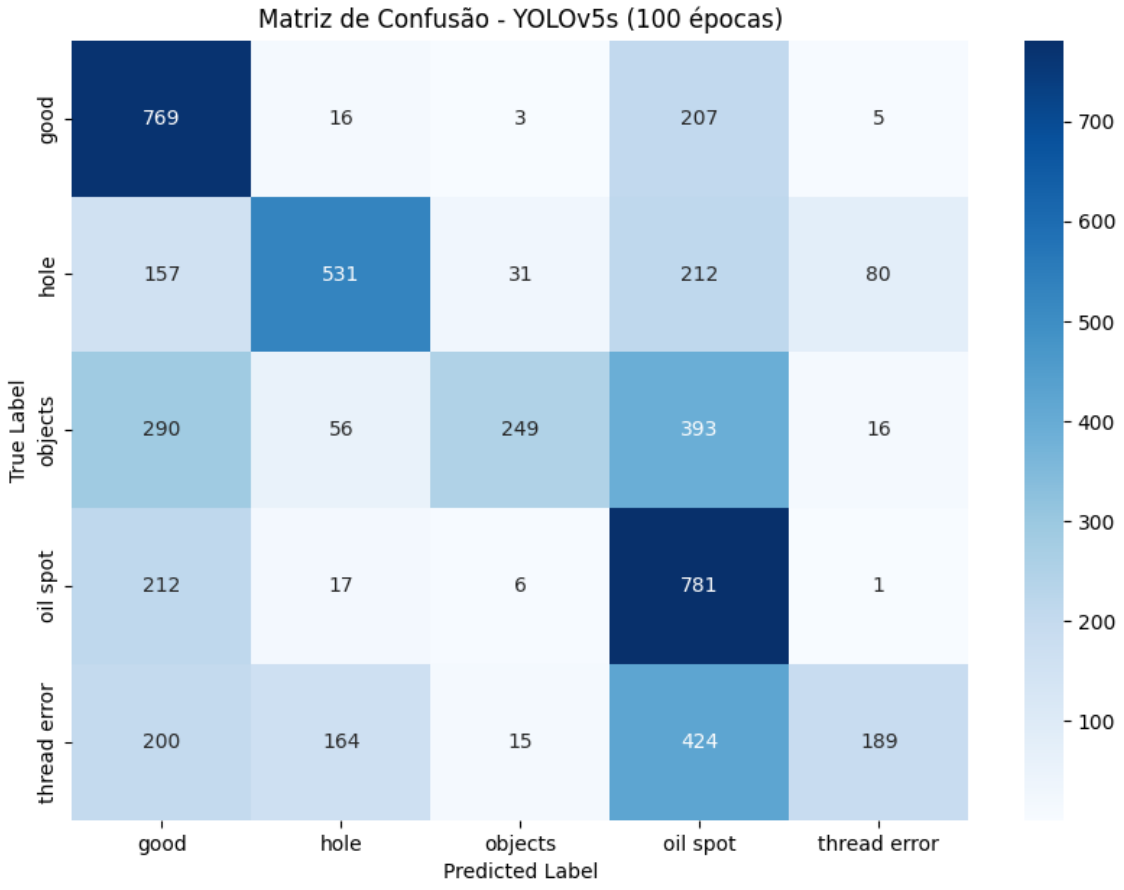

Figura A.6: Matriz de Confusão - Modelo YOLOv5s (100 Épocas)

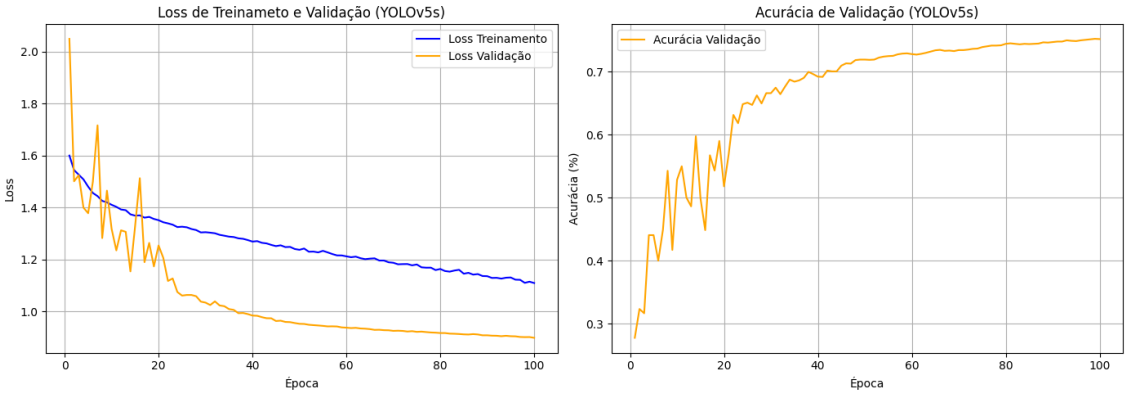

Figura A.7: Curva de Loss e Acurácia para Modelo YOLOv5s

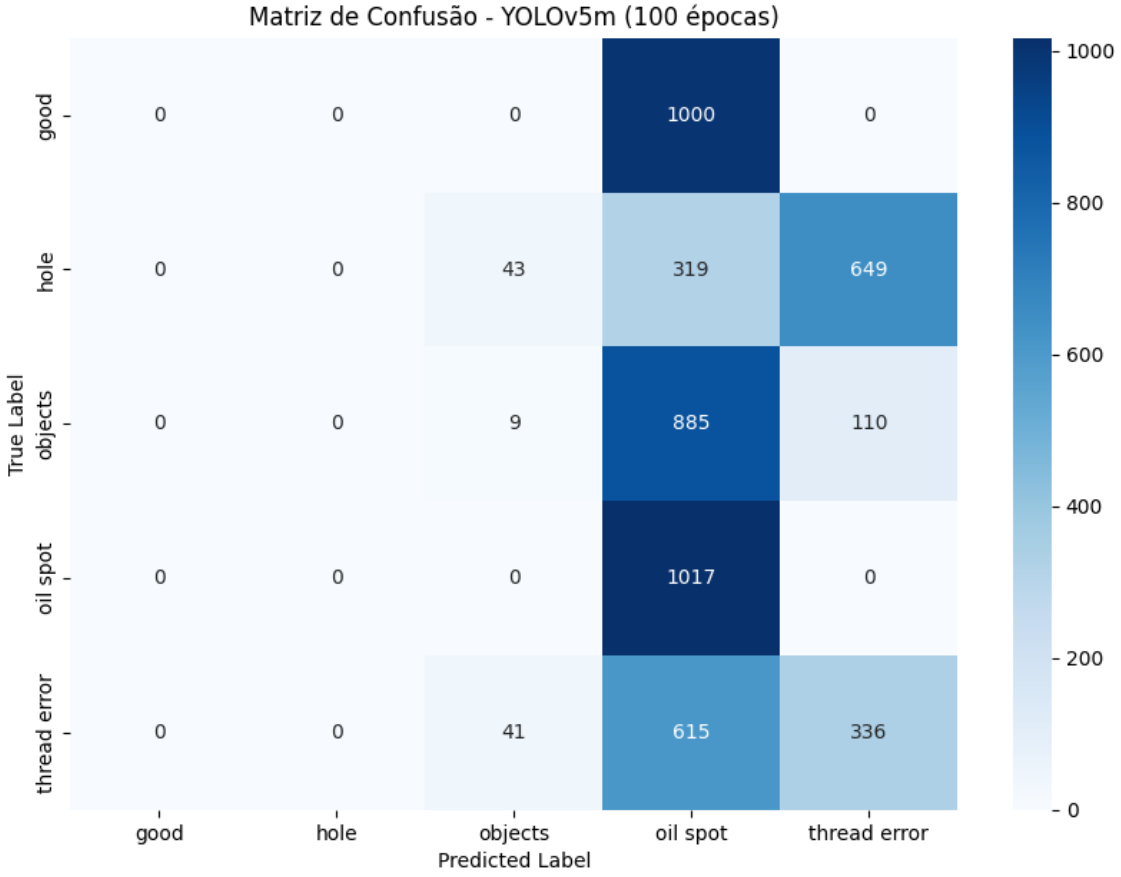

Figura A.8: Matriz de Confusão - Modelo YOLOv5m (100 Épocas)

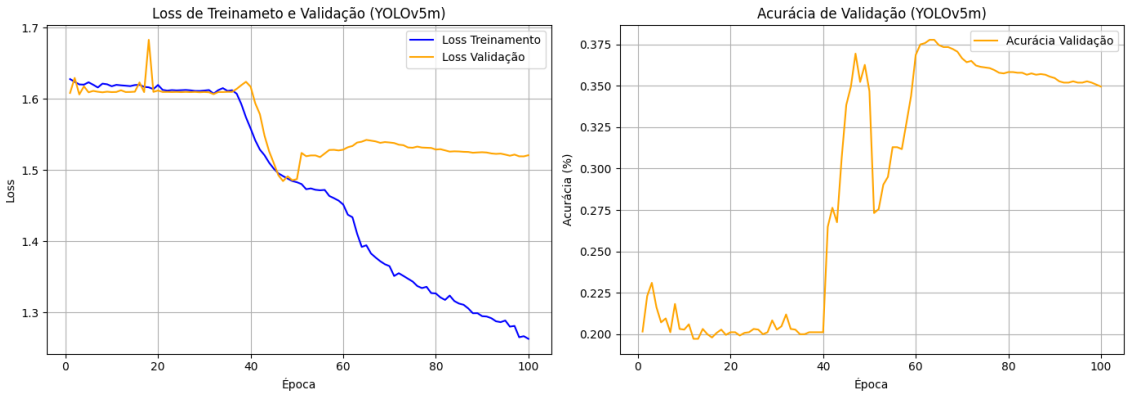

Figura A.9: Curva de Loss e Acurácia para Modelo YOLOv5m

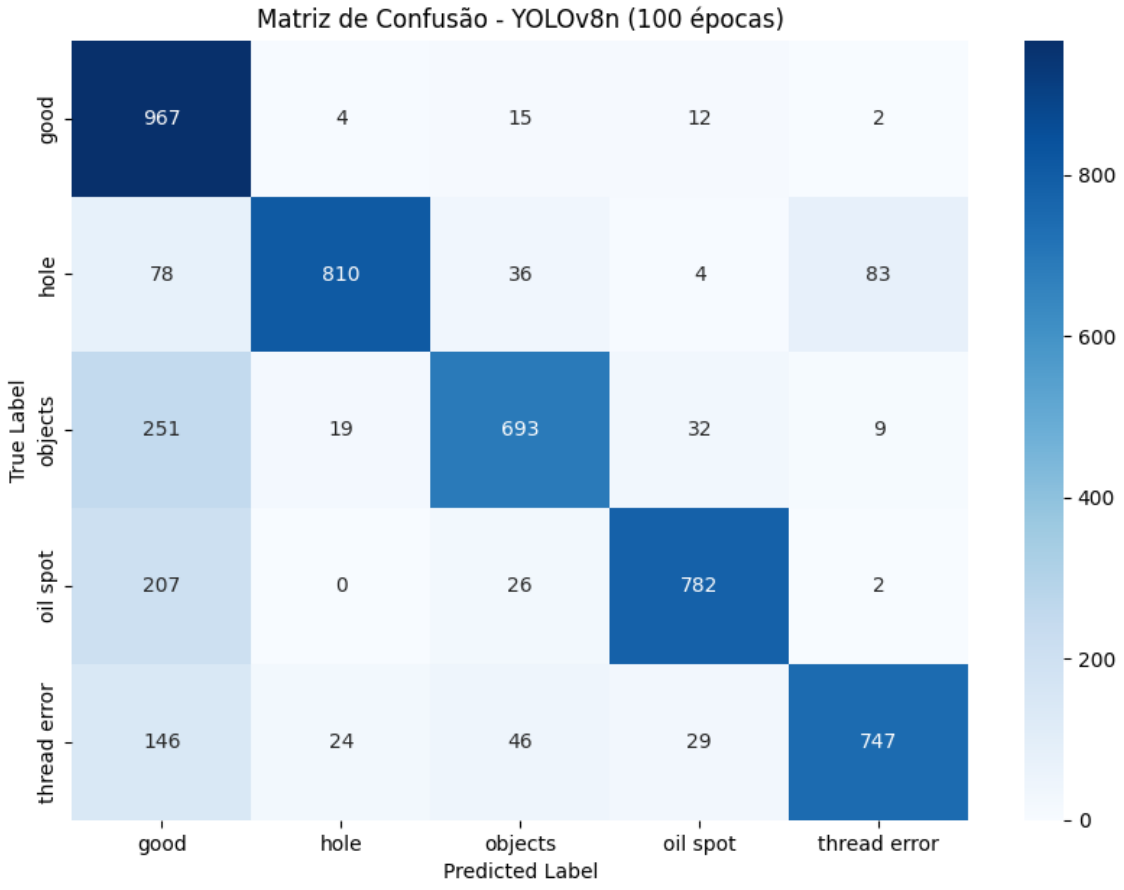

Figura A.10: Matriz de Confusão - Modelo YOLOv8n (100 Épocas)

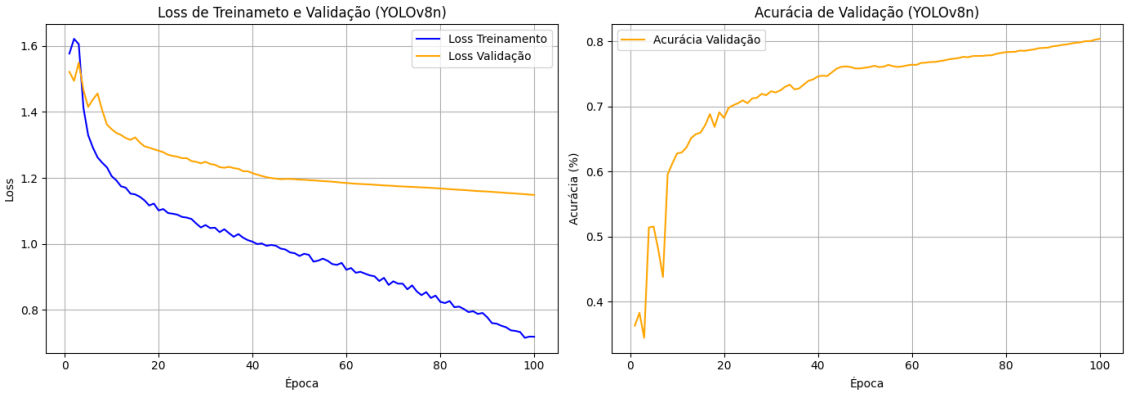

Figura A.11: Curva de Loss e Acurácia para Modelo YOLOv8n

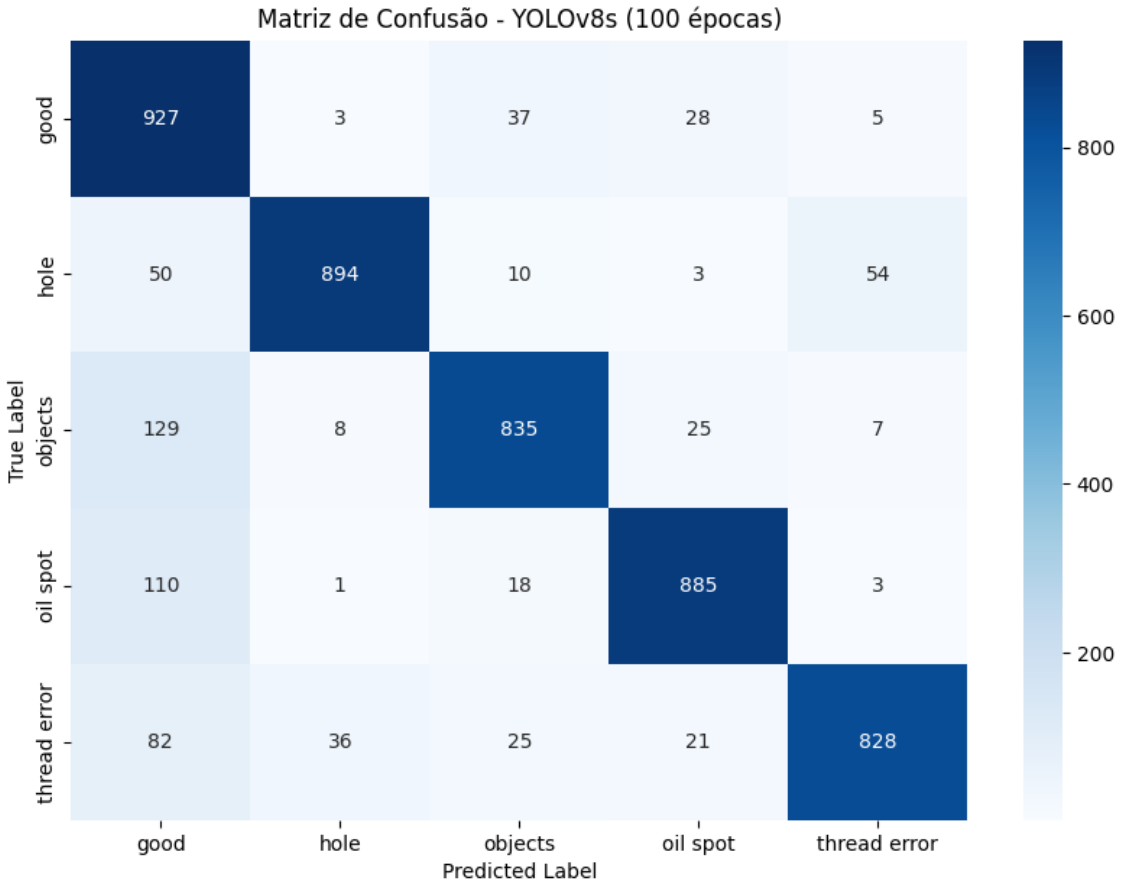

Figura A.12: Matriz de Confusão - Modelo YOLOv8s (100 Épocas)

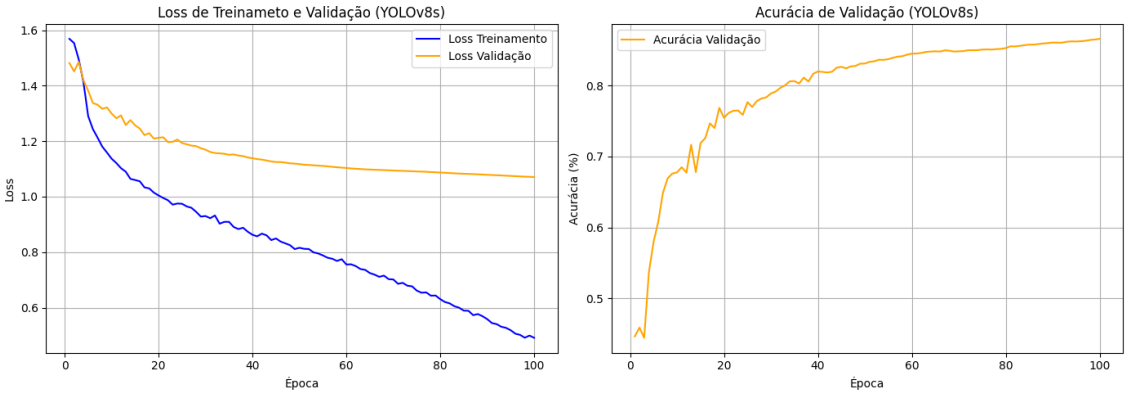

Figura A.13: Curva de Loss e Acurácia para Modelo YOLOv8s