Aplicação Web para detecção de objetos e rastreamento de trajetórias automatizada utilizando YOLOv8

Stefano Tomei

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, PB

2024

Stefano Tomei

Aplicação Web para detecção de objetos e rastreamento de trajetórias automatizada utilizando YOLOv8

Monografia apresentada ao curso Engenharia de Computação
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em Engenheiro de Computação

Orientadora: Thaís Gaudencio do Rêgo

Novembro de 2024

Catalogação na publicação
Seção de Catalogação e Classificação

T656a Tomei, Stefano.
Aplicação web para detecção de objetos e rastreamento de trajetórias automatizada utilizando YOLOv8 / Stefano Tomei. - João Pessoa, 2024.
52 f. : il.

Orientação: Thaís Gaudencio do Rêgo.
TCC (Graduação) - UFPB/CI.

1. Detecção de objetos. 2. YOLO. 3. YOLOv8. 4. Rastreamento de objetos. 5. Redes neurais convolucionais. 6. BoT-SORT. 7. ByteTrack. I. Rêgo, Thaís Gaudencio do. II. Título.

UFPB/CI CDU 004.8

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

Trabalho de Conclusão de Curso de Engenharia de Computação intitulado ***Aplicação Web para detecção de objetos e rastreamento de trajetórias automatizada utilizando YOLOv8*** de autoria de Stefano Tomei, aprovada pela banca examinadora constituída pelos seguintes professores:

Prof. Dr. Thaís Gaudencio do Rêgo
Departamento de Informática - Centro de Informática - UFPB

Prof. Dr. Yuri de Almeida Malheiros Barbosa
Departamento de Computação Científica - Centro de Informática - UFPB

Me. Caio Marcelo Campoy Guedes

João Pessoa, 14 de novembro de 2024

Centro de Informática, Universidade Federal da Paraíba
Rua dos Escoteiros, Mangabeira VII, João Pessoa, Paraíba, Brasil CEP: 58058-600
Fone: +55 (83) 3216 7093 / Fax: +55 (83) 3216 7117

AGRADECIMENTOS

Sou eternamente grato aos meus pais, que me forneceram amor, apoio e encorajamento inabaláveis desde o início desta jornada. Sem eles, enfrentar os desafios teria sido imensuravelmente mais difícil.

Agradeço também de coração aos meus amigos, que estiveram ao meu lado em todos os momentos, compartilhando tanto os desafios quanto as alegrias dessa etapa. Anderson José da Silva Frazão, Victor Batista Maia, Amon Lima de Sousa, Caio Marcelo Campoy Guedes, Yure Galdino Lopes, Deborah de Jesus Pereira Viana, Edson Alves Pereira Filho, João Vinícius, Flávia Passos, Nycholas de Sousa, Ana Carolina Simões Silva, Lucas Rincon, Márcio Welben Montenegro Mariani Alves,Yuri da Costa Gouveia, Altair Jussadir da Silva Pinto, Wellton Souza, Emmanuel De Miranda Viana Pinto, Luiz Henrique Rodrigues de Oliveira, Joanacelle Caldas de Melo, Suzane Gomes dos Santos e José Henrique de Carmago Januário, a amizade e o apoio de vocês foram essenciais.

Sou extremamente grato à minha professora e orientadora, Thaís Gaudencio do Rêgo, pela paciência e orientação ao longo deste projeto. Juntamente com Yuri de Almeida Malheiros Barbosa por seu suporte inestimável durante este percurso. A dedicação e apoio de vocês foram fundamentais para o meu desenvolvimento.

Em especial, quero reconhecer o apoio recebido durante os momentos mais sombrios, quando as dúvidas e incertezas pareciam intransponíveis. Vocês foram um farol de esperança e coragem, iluminando meu caminho quando mais precisei. Cada palavra de incentivo e cada gesto de apoio contribuíram imensamente para que eu perseverasse.

Cada um de vocês desempenhou um papel vital em minha trajetória, e sou profundamente grato por ter tido o privilégio de contar com o apoio inquestionável de todos. Obrigado por acreditarem em mim e por estarem sempre presentes.

RESUMO

A crescente demanda por sistemas automatizados capazes de realizar análises precisas em vídeos e imagens impulsiona o desenvolvimento de soluções avançadas em visão computacional. Nesse contexto, surge a necessidade de ferramentas eficientes para detecção e rastreamento de objetos que possam ser aplicadas em diversas áreas, como segurança, transporte, automação industrial, e muito mais. Este trabalho apresenta o desenvolvimento de uma aplicação web para detecção e rastreamento de objetos em vídeos e imagens, utilizando o algoritmo YOLOv8. A aplicação processa arquivos fornecidos pelos usuários, aplicando técnicas de visão computacional para identificar e rastrear objetos de forma eficiente. A interface da aplicação permite ao usuário selecionar entre diferentes modelos do YOLO (*nano, small, medium, large e huge*), cada um otimizado para diferentes níveis de precisão e velocidade. Além disso, a aplicação utiliza 12 das 80 classes disponibilizadas pelo YOLO. Essas classes foram priorizadas por sua relevância em cenários reais, nos quais a detecção e o rastreamento de pessoas, veículos, animais e objetos comuns são essenciais. Testes comparativos foram realizados utilizando os diferentes modelos do YOLO e os algoritmos de rastreamento BoT-SORT e ByteTrack para avaliar o desempenho em termos de acurácia e tempo de processamento. Os resultados incluem um arquivo processado com a identificação dos objetos detectados e dados detalhados sobre a quantidade de objetos, classes detectadas, média de acurácia das detecções e tempos de processamento para cada configuração testada. A conclusão destaca a eficácia do sistema desenvolvido e propõe aprimoramentos futuros, como a atualização dos modelos utilizados, integração com links externos para processamento de vídeos em tempo real, e melhorias na interface do usuário para facilitar a visualização dos resultados.

Palavras-chave: <Detecção Automática>, <Redes Neurais Convolucionais>, <Detecção de Objetos>, <YOLO>, <YOLOv8>, <Rastreamento de Objetos>, <BoT-SORT>,<ByteTrack>.

ABSTRACT

The growing demand for automated systems capable of performing accurate analyses on videos and images drives the development of advanced computer vision solutions. In this context, there is a need for efficient tools for object detection and tracking that can be applied in various fields such as security, transportation, industrial automation, and more. This study presents the development of a web application for object detection and tracking in videos and images using the YOLOv8 algorithm. The application processes files provided by users, applying computer vision techniques to efficiently identify and track objects. The application's interface allows users to select from different YOLO models (*nano, small, medium, large, and huge*), each optimized for varying levels of precision and speed. Additionally, the application uses 12 of the 80 classes available in YOLO. These classes were prioritized based on their relevance in real-world scenarios where the detection and tracking of people, vehicles, animals, and common objects are essential. Comparative tests were conducted using the different YOLO models and the tracking algorithms BoT-SORT and ByteTrack to evaluate performance in terms of accuracy and processing time. The results include a processed file with the identification of detected objects and detailed data on the number of objects, detected classes, average detection accuracy, and processing times for each configuration tested. The conclusion highlights the effectiveness of the developed system and proposes future improvements, such as model updates, integration with external links for real-time video processing, and enhancements to the user interface to facilitate result visualization.

Keywords: <Automatic Detection>, <Convolutional Neural Networks>, <Object Detection>, <YOLO>, <YOLOv8>, <Object Tracking>, <BotSort>, <ByteTrack>.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AM – Aprendizagem de Máquina

AP – Aprendizagem Profunda

API – *Application Programming Interface* (Interface de Programação de Aplicação)

CNN – *Convolutional Neural Network* (Rede Neural Convolucional)

CPU – *Central Processing Unit* (Unidade Central de Processamento)

CSS – *Cascading Style Sheets*

CUDA – *Compute Unified Device Architecture*

DL – *Deep Learning* (Aprendizagem Profunda)

GB – *Gigabyte*

GHz – *Gigahertz*

GPU – *Graphics Processing Unit* (Unidade de Processamento Gráfico)

IBM – *International Business Machines* (Máquinas de Negócios Internacional)

IA – Inteligência Artificial

MB - *Megabyte*

RAM – *Random Access Memory* (Memória de Acesso Randômico)

ReLU – *Rectified Linear Unit* (Unidade Linear Retificada)

RNA – Rede Neural Artificial

RNC – Rede Neural Convolucional

TB – *Terabyte*

YOLO – *You Only Look Once*

Sumário

1 INTRODUÇÃO

A visão computacional pode transformar diversas áreas críticas, como a segurança pública, a saúde, a automação industrial e o setor comercial. Essa tecnologia permite a análise de grandes volumes de dados visuais, extraindo informações valiosas que podem ser utilizadas para melhorar processos e tomar decisões mais informadas [7]. A visão computacional envolve técnicas de aprendizado de máquina, redes neurais convolucionais e algoritmos avançados que possibilitam a detecção, reconhecimento e rastreamento de objetos em imagens e vídeos [15].

O aumento da criminalidade e a necessidade de monitoramento constante em áreas urbanas densamente povoadas são questões críticas. Em muitas grandes cidades, sistemas de vigilância com visão computacional são utilizados para detectar automaticamente comportamentos suspeitos e identificar indivíduos em tempo real [24]. Esses sistemas analisam dados visuais de câmeras de segurança, identificando rapidamente ameaças potenciais e alertando as autoridades competentes, o que pode reduzir significativamente o tempo de resposta e aumentar a eficiência das operações de segurança [34].

Na área da saúde, a demanda por diagnósticos precisos e rápidos está em constante crescimento. A visão computacional pode revolucionar a análise de imagens médicas, permitindo a detecção precoce de doenças como câncer, doenças cardiovasculares e outras condições críticas. Hospitais têm utilizado algoritmos de aprendizado profundo treinados em grandes conjuntos de dados para identificar padrões sutis em radiografias, tomografias e ressonâncias magnéticas, que podem ser facilmente perdidos pelo olho humano [41]. Isso não apenas melhora a precisão dos diagnósticos, mas também acelera o processo, permitindo que os médicos se concentrem mais no tratamento e menos na detecção [16].

A automação industrial é outra área onde a visão computacional pode oferecer soluções significativas. A manufatura moderna exige altos níveis de precisão e controle de qualidade. Sistemas de visão computacional podem ser integrados em linhas de produção para inspeção automática de produtos, identificação de defeitos e garantia de conformidade com os padrões de qualidade [32]. Isso reduz a dependência de inspeções manuais, diminui o desperdício de materiais e aumenta a eficiência geral da produção [9].

No setor comercial, a visão computacional pode ser utilizada para melhorar a eficiência e a eficácia das operações. Sistemas de análise de vídeo podem contar o número de pessoas que transitam em frente a uma loja, permitindo que os comerciantes entendam melhor os padrões de tráfego e ajustem suas estratégias de marketing e alocação de pessoal. Além disso, a visão computacional pode ser usada para determinar os melhores locais para colocar publicidade física, como *outdoors*, analisando dados de fluxo de pedestres e veículos para maximizar a visibilidade e o impacto das campanhas publicitárias [33].

Além dessas áreas, a agricultura enfrenta desafios com a necessidade de alimentar uma população mundial crescente. A visão computacional pode ser utilizada em drones e robôs agrícolas para monitorar o crescimento das plantas, identificar pragas e doenças, e otimizar a aplicação de fertilizantes e pesticidas [2]. Isso resulta em uma agricultura mais sustentável e eficiente, com melhor uso dos recursos e menores impactos ambientais [8].

O transporte é outro setor beneficiado pela visão computacional. Sistemas de transporte inteligentes podem usar câmeras e sensores para monitorar o tráfego, detectar congestionamentos e acidentes, e otimizar rotas para reduzir o tempo de viagem e o consumo de combustível. Além disso, essas tecnologias podem melhorar a logística de policiamento do trânsito, respondendo rapidamente a emergências e melhorando a segurança nas vias [1].

No contexto ambiental, a visão computacional pode ajudar a monitorar e preservar ecossistemas. Sistemas de monitoramento baseados em visão computacional podem ser utilizados para rastrear espécies ameaçadas, monitorar desmatamento e detectar poluição em corpos d'água [23]. Essas aplicações ajudam a proteger a biodiversidade e a garantir a sustentabilidade dos recursos naturais para as futuras gerações. Por exemplo, na Amazônia, tecnologias de visão computacional, juntamente com aprendizado de máquina e processamento em nuvem, são empregadas para monitorar o desmatamento ilegal e proteger a floresta tropical [5].

Apesar de seu potencial transformador, a implementação de sistemas de visão computacional enfrenta desafios técnicos e éticos. A precisão dos algoritmos, a necessidade de grandes volumes de dados para treinamento e as preocupações com a privacidade e a segurança dos dados são questões que precisam ser abordadas [10]. A pesquisa contínua e a colaboração entre acadêmicos, indústria e formuladores de políticas são essenciais para superar esses desafios e maximizar os benefícios da visão computacional.

Em resumo, a visão computacional oferece soluções promissoras para muitos dos problemas mais urgentes do mundo atual. Desde a melhoria da segurança pública, até a revolução nos cuidados de saúde, passando pela automação industrial, comércio, agricultura sustentável, transporte inteligente e monitoramento ambiental, as aplicações são vastas e variadas. Com investimentos contínuos em pesquisa e desenvolvimento, e uma abordagem cuidadosa para resolver os desafios técnicos e éticos, a visão computacional pode desempenhar um papel crucial na construção de um futuro mais seguro, saudável e sustentável.

1.1 Tema

A aplicação da visão computacional, particularmente através do modelo de rede neural YOLOv8 (You Only Look Once versão 8) desenvolvido pela Ultralytics, é uma tecnologia essencial para a análise de grandes volumes de dados visuais. O YOLOv8 destaca-se pela eficiência e precisão na detecção de objetos, sendo ideal para contextos onde a análise manual pode ser custosa [36].

A aplicação *Web* proposta neste trabalho utiliza o YOLOv8, considerado um dos modelos mais avançados em detecção de objetos, com o objetivo de fornecer uma plataforma capaz de receber arquivos de vídeo e imagem nos formatos 'png', 'jpg', 'jpeg', 'gif', 'bmp', 'mp4', 'mov', 'avi' e 'webm', processá-los e retornar os resultados em arquivos de vídeo, conforme os parâmetros definidos pelo usuário. Com parâmetros customizados, esta abordagem visa facilitar a análise automatizada e personalizada de vídeos, promovendo eficiência e precisão na detecção de objetos.

Em uma aplicação prática, o projeto poderá ser utilizado em diversas situações, como na contabilização do fluxo de pessoas e veículos, fornecendo informações relevantes do ponto de vista comercial e publicitário. Outra aplicação possível seria a identificação e contagem de carros em estacionamentos, além do monitoramento de pessoas em diferentes seções de supermercados, com o intuito de otimizar a disposição de produtos e melhorar a experiência do cliente.

1.2 Problema

Com o aumento do uso de tecnologia, a análise manual de vídeos, que já era considerada um processo custoso e ineficiente, tornou-se ainda mais inadequada para atender às demandas atuais. Áreas como segurança, saúde, indústria e comércio precisam analisar grandes volumes de dados visuais de forma precisa. No entanto, detectar e rastrear objetos em vídeos é um grande desafio devido à complexidade das cenas.

As soluções atuais muitas vezes não são precisas ou personalizáveis o suficiente, para atender às necessidades variadas dos usuários. Além disso, faltam plataformas que permitam uma fácil integração e adaptação aos parâmetros específicos definidos pelos usuários, limitando a aplicação prática dessas tecnologias.

A aplicação *web* proposta neste trabalho visa abordar esses problemas ao usar o modelo de rede neural YOLO, conhecido por sua eficiência e precisão na detecção de objetos. A plataforma permitirá o processamento automatizado de vídeos, fornecendo resultados personalizados, conforme os parâmetros definidos pelo usuário. Assim, busca-se oferecer uma solução prática e eficaz para a análise de vídeos em diversos contextos.

1.2.1 Objetivo geral

O objetivo deste trabalho é desenvolver uma aplicação web que utilize a tecnologia YOLOv8 da Ultralytics para a identificação e rastreamento de objetos em vídeos, com base nos parâmetros de entrada fornecidos pelos usuários. Ao final do processamento, a aplicação fornecerá, caso requisitado, um relatório detalhado sobre o desempenho do sistema, incluindo tempos de processamento, classes e número de detecções realizadas, juntamente com o vídeo resultante das detecções.

1.2.2 Objetivos específicos

- **Desenvolver a arquitetura do servidor *backend*:** Criar uma estrutura robusta utilizando Python e tecnologias modernas, visando a eficiência e escalabilidade da aplicação.
- **Implementar um sistema de filas no servidor:** Utilizar processamento em paralelo para reduzir o tempo de espera, melhorando a experiência do usuário.
- **Integrar YOLOv8 para detecção de objetos:** Permitir ao usuário a escolha do modelo YOLOv8 a ser utilizado para o processamento.
- **Integrar algoritmos de rastreamento de objetos:** Permitir a escolha entre BoT-SORT e ByteTrack, atendendo às necessidades específicas dos usuários em relação ao rastreamento.
- **Desenvolver a interface do *frontend*:** Criar uma interface intuitiva que permita aos usuários interagir facilmente com a aplicação, enviando arquivos para processamento e visualizando os resultados.
- **Realizar testes comparativos de desempenho:** Avaliar a acurácia e o tempo de processamento entre diferentes modelos YOLOv8 e algoritmos de rastreamento, fornecendo dados para otimização do sistema.

1.3 Estrutura da monografia

No Capítulo 2 serão apresentados conceitos importantes e necessários para o entendimento do presente trabalho. Neste capítulo, serão abordados temas como Visão Computacional, Redes Neurais Artificiais, Aprendizagem Profunda, Redes Neurais Convolucionais, YOLO, rastreamento de objetos e as demais tecnologias aplicadas no desenvolvimento.

No Capítulo 3 será apresentada toda a metodologia aplicada no desenvolvimento da aplicação, incluindo a arquitetura do sistema e a escolha das tecnologias utilizadas.

No Capítulo 4 serão apresentados e discutidos os resultados obtidos baseados nos parâmetros de entrada do sistema, mostrando os dados referentes ao uso dos diferentes modelos do YOLOV8.

No Capítulo 5, serão apresentadas a conclusão do trabalho, as melhorias implementadas e sugestões para trabalhos futuros.

2 Conceitos Gerais

Neste capítulo, serão abordados os conceitos fundamentais para a compreensão do trabalho. Inicialmente, serão apresentados os conceitos, de forma breve, sobre Visão Computacional, Redes Neurais Artificiais (RNA), Aprendizado Profundo (DL, do inglês *Deep Learning*) e Redes Neurais Convolucionais (CNNs, do inglês *Convolutional Neural Network*). Em seguida, será explorada a detecção de objetos, com um foco especial no algoritmo YOLO (You Only Look Once) e sua versão mais recente, YOLOv8. Também serão discutidas as técnicas ByteTrack e BoT-SORT para rastreamento de objetos. Por fim, serão abordados componentes de *backend*, como a linguagem Python, o *framework* Flask, a biblioteca Celery e o RabbitMQ, o gerenciador de filas, bem como os elementos de *frontend*, incluindo HTML, CSS, Materialize, Javascript e jQuery.

2.1 Visão Computacional

A visão computacional é um campo de IA, que busca fazer a capacitação de máquinas a interpretar, reconhecer e entender o mundo de uma forma mais abrangente. Essa área da IA, permite que sistemas automatizados realizem tarefas como reconhecimento de objetos, identificação de padrões e interpretação de cenas, a partir de um conjunto dados. Para atingir esses objetivos, a visão computacional utiliza algoritmos avançados, como CNNs, que são especialmente projetadas para processar informações e extrair dados relevantes de diferentes tipos de entrada [7][15].

Com o avanço das técnicas de DL, a visão computacional tem se tornado cada vez mais precisa e eficiente, possibilitando sua aplicação em uma ampla gama de setores. Indústrias utilizam essa tecnologia para inspeção de qualidade e controle de processos, automatizando a detecção de defeitos em produtos com precisão que supera a capacidade do olho humano [32]. No comércio, sistemas de identificação de produtos automatizados reconhecem os itens por meio de câmeras, eliminando a necessidade de escanear códigos de barras manualmente, o que resulta em um atendimento mais ágil e eficiente [33]. Além disso, a visão computacional também está presente no monitoramento de segurança pública, na análise de tráfego em sistemas de transporte e até na agricultura, onde pode monitorar o crescimento de plantações e identificar padrões gerados pela presença de pragas de forma automatizada [32].

O avanço contínuo no desenvolvimento de soluções e aprimoramento da visão computacional permite que soluções automatizadas sejam cada vez mais precisas e eficientes no tratamento de dados. A combinação de algoritmos de detecção e rastreamento com CNNs e técnicas de DL tem sido fundamental para alcançar esses resultados [7][15]. Atualmente, sistemas modernos utilizam arquiteturas complexas e avançadas, como o YOLO (You Only Look Once).

Esses progressos refletem a importância crescente da visão computacional, que vem se expandindo consideravelmente e sendo aplicada em diferentes setores, possibilitando o desenvolvimento de ferramentas que automatizam processos complexos e oferecem informações valiosas. Para impulsionar ainda mais essa revolução, as RNAs desempenham um papel crucial, fornecendo as bases teóricas e práticas que permitem o funcionamento eficiente de muitos dos algoritmos utilizados em visão computacional.

2.2 Redes Neurais Artificiais

As RNAs são, em resumo, uma máquina que foi modelada para performar como o cérebro humano para uma função ou tarefa particular de interesse. Assim como o cérebro, seus sinais são processados através de neurônios e organizadas em forma de camadas. Cada neurônio recebe uma entrada, processa a entrada com base em seus pesos ajustáveis e uma função de ativação e transmite a saída para os neurônios nas próximas camadas [14].

2.3 Aprendizado Profundo

DL é uma subárea do aprendizado de máquina, baseada na estrutura e funcionamento do cérebro humano, utilizando RNAs. Consiste em múltiplas camadas de neurônios entre a camada de entrada e a camada de saída [13]. É com frequência utilizado em aplicações como detecção de objetos, tradução automática, reconhecimento de fala e processamento de imagens [31].

2.4 Redes Neurais Convolucionais

As CNNs são um tipo de rede neural especializada em processar dados que possuem uma estrutura de grade bidimensional, como imagens. O nome CNN se deve à operação matemática chamada convolução. Representada por um asterisco ([$*$]), a convolução é uma operação que ajuda a reduzir o ruído nas amostras e a extrair características relevantes dos dados de entrada [13].

$$s(t) = (x * w) \tag{1}$$

Na equação (1), x representa os dados de entrada e w são os pesos (ou *filters*). O resultado, $s(t)$, é o mapa de características gerado pela convolução dos dados de entrada com os pesos [13].

Na Figura 1, é ilustrada a aplicação da convolução 2D em uma imagem. A figura apresenta a entrada (Input) como uma matriz de pixels e o *Kernel* como a matriz de pesos que atua como um filtro.

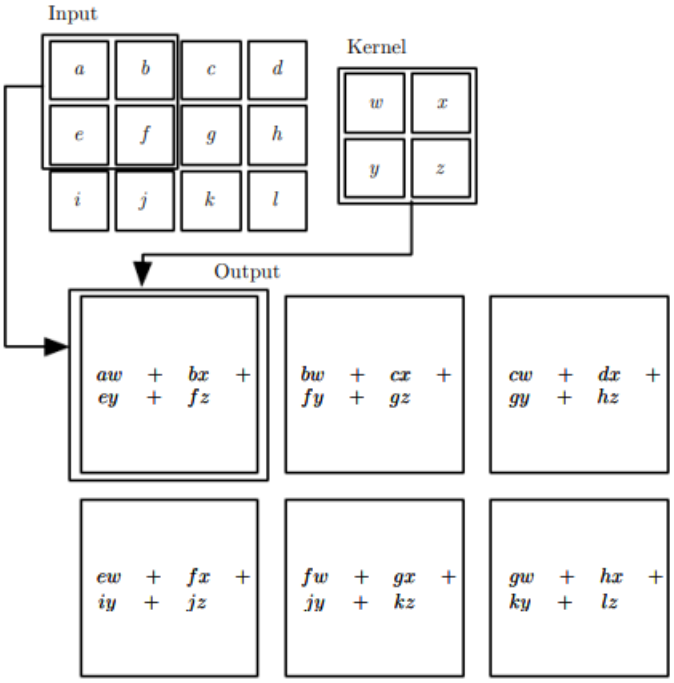

Figura 1: *Exemplo de aplicação de uma convolução 2D em uma matriz de imagem. A imagem de entrada (Input) é representada como uma matriz de pixels, enquanto o Kernel (filtro) é uma matriz de pesos que é aplicada sobre a imagem. A convolução é realizada por meio da multiplicação do Kernel por cada submatriz da imagem, resultando em uma nova matriz que captura as características extraídas. Cada elemento da matriz resultante reflete a intensidade da característica detectada pelo Kernel na região correspondente da imagem de entrada. Fonte: [13].*

Uma vez entendido o funcionamento das CNNs e suas capacidades de extração de características, é importante considerar que essas redes são compostas por várias camadas específicas, cada uma desempenhando funções críticas para o processamento de dados visuais. Segundo [13], as camadas principais incluem:

- **Camadas Convolucionais**: Responsáveis por aplicar filtros sobre a entrada, extraindo características locais, como bordas, texturas e padrões.
- **Camadas de Pooling (ou Subamostragem)**: Reduzem as dimensões espaciais da saída da camada convolucional, diminuindo a quantidade de parâmetros e cálculos na rede, o que contribui para a eficiência do modelo. Isso é realizado geralmente através de operações como *max pooling*, que seleciona o valor máximo em uma região específica da matriz de entrada, ou *average pooling*, que calcula a média dos valores em uma região específica. Essas operações ajudam a preservar as características mais relevantes enquanto reduzem a complexidade computacional.
- **Camadas de Ativação**: Frequentemente, após uma operação de convolução, aplica-

se uma função de ativação não linear, como $ReLU$ (Rectified Linear Unit), que introduz não linearidade no modelo e ajuda a capturar padrões complexos. A introdução de não linearidade é essencial, pois permite que a rede aprenda transformações mais complexas dos dados, superando a limitação das transformações lineares, que não seriam capazes de representar relações mais sofisticadas entre os dados. Sem essa não linearidade, a rede não seria capaz de aprender a partir de dados complexos, como imagens, de forma eficaz.

- **Camadas Fully Connected (ou Densas)**: Essas camadas conectam todos os neurônios de uma camada à próxima e são utilizadas para combinar as características extraídas nas camadas anteriores e realizar a classificação ou a regressão, dependendo da tarefa.

A combinação dessas camadas permite que as CNNs identifiquem padrões complexos em imagens, aprendendo de maneira hierárquica — das características mais simples (como bordas e texturas) às mais abstratas (como formas e objetos completos). Essa capacidade de extrair e combinar características possibilita a identificação e classificação de múltiplos objetos em uma única imagem, o que é essencial para tarefas como detecção de objetos em tempo real, reconhecimento de faces, e diversas aplicações na visão computacional. A habilidade das CNNs em processar informações visuais de forma eficiente abre caminhos para soluções inovadoras e cada vez mais sofisticadas no campo da análise visual [13].

2.5 YOLO

YOLO, é considerado o estado da arte em detecção de objetos desde suas primeiras versões, conseguindo se destacar em relação aos demais algoritmos de detecção de objetos [36]. YOLO encara o problema da detecção de objetos como um problema de regressão, associando valores contínuos para cada objeto, gerando assim caixas delimitadoras, a classificação e suas respectivas probabilidades. Todo o fluxograma de detecção é feito em uma única rede e em uma única avaliação [28].

Em sua versão inicial, o YOLO, considerado mais rápido que a maioria dos algoritmos existentes [28], apresentava problemas na localização de objetos pequenos, mas, em contrapartida, gerava menos falsos positivos. Isso acontecia porque o YOLO dividia a imagem em uma grade de células, e cada célula podia gerar apenas um objeto. Nesse caso, se um objeto pequeno aparecesse em uma parte pequena de uma célula, acabaria não sendo detectado com precisão [28]. A resolução deste problema resultou em novas versões do YOLO, entre essas versões, temos a versão 8, que implementa melhorias significativas em precisão e desempenho [36].

Segue abaixo a tabela de versões do YOLO com seus principais aprimoramentos no algoritmo.

Versão	Melhorias
YOLOv2	Âncoras (*Anchor Boxes*): São caixas de diferentes proporções e tamanhos pré-definidas. Isso permitiu ao modelo prever múltiplas caixas para cada célula da grade, melhorando assim a capacidade de detectar objetos pequenos [27].
	Normalização em lote (*Batch Normalization*): A inclusão da normalização em lote ajudou a estabilizar o treinamento e melhorar a precisão geral do modelo, incluindo a detecção de objetos pequenos [27].
	Camada de Passagem (*Passthrough Layer*): Esta camada permitiu a combinação de características de camadas mais profundas com as de camadas mais rasas, preservando detalhes finos da imagem que são cruciais para detectar objetos pequenos [27].
YOLOv3	Arquitetura de Rede Residual: Utilizou uma rede mais profunda e complexa, baseada na arquitetura *Darknet-53* com conexões residuais, que ajudou a modelar melhor os detalhes da imagem [29].

Versão	Melhorias
	Previsões em Múltiplas Escalas: A previsão é feita em escalas diferentes, permitindo que o objeto seja detectado em diversos tamanhos. Isso é realizado ao gerar previsões a partir de três camadas de características, cada uma com uma resolução diferente, o que melhora significativamente a detecção de objetos pequenos [29].
	Melhoria nas Âncoras: O método de âncoras foi refinado, resultando em uma melhoria na correspondência das caixas preditivas com os objetos em diferentes escalas [29].
	Rede de Pirâmide de Características (*Feature Pyramid Network*, ou FPN): A FPN foi utilizada para combinar características de diferentes camadas da rede, permitindo que informações de resoluções variadas sejam usadas para detectar objetos de diferentes tamanhos. A FPN melhora a capacidade de detecção de objetos pequenos, ao usar características extraídas em várias escalas [29].
YOLOv4	CSPDarknet53: Até então utilizava-se a arquitetura Darknet-53 como *backbone*, nesta versão foi introduzido a *CSPDarknet53* como a nova rede *backbone*, que aprimora a eficiência e a precisão da rede [4].
	Data Augmentation: Implementou técnicas avançadas de *Data Augmentation*, como *Mosaic* e *CutMix*, que ajudam a rede a generalizar melhor e detectar objetos pequenos em diferentes contextos[4].
	PANet: Utilizou a *Path Aggregation Network* (PANet) para aprimorar a fusão de características, melhorando a propagação de informações úteis sobre objetos pequenos nas camadas da rede [4].
YOLOv5	Modelo de escala automatizado: Automatizou o escalonamento do modelo, ajustando automaticamente o tamanho da rede e a resolução da imagem para otimizar a detecção de objetos pequenos [17].
	Data Augmentation com *Mosaic*: Continuou a utilizar técnicas avançadas de *Data Augmentation*, como o *Mosaic*, que combina quatro imagens de treinamento em uma única, aumentando a variabilidade e melhorando a detecção de objetos pequenos [17].
	Aprimoramento no PANet: Melhorou a arquitetura PANet para uma fusão de características ainda mais eficiente, aumentando a precisão na detecção de objetos de várias escalas, incluindo pequenos [17].

Versão	Melhorias
YOLOv6	Arquitetura *Backbone* YOLOv6: Desenvolveu uma nova arquitetura de *backbone* mais eficiente, que equilibra a profundidade e a largura da rede para melhorar a capacidade de detecção [38].
	Caminhos de Agregação Eficientes: Incorporou a *EfficientRep Backbone*, uma variação otimizada da rede *CSPDarknet*, para melhorar a eficiência computacional e a precisão [38].
	Otimizações de Treinamento: Implementou novas estratégias de treinamento, como técnicas avançadas de *Data Augmentation* e reguladores que ajudam a evitar sobreajuste, melhorando a detecção de objetos pequenos [38].
	Enhanced Detection Head: Melhorou a *head* de detecção com uma arquitetura *PANet* mais robusta e eficiente, aumentando a precisão e a capacidade de detecção de objetos pequenos [38].
YOLOv7	Arquitetura *Backbone* Melhorada: YOLOv7 introduziu uma nova arquitetura de *backbone*, que utiliza uma combinação de camadas CSP *(Cross Stage Partial)* e camadas convolucionais, otimizando o balanceamento entre profundidade e largura da rede para aumentar a precisão e a eficiência [37].
	Agregação de Caminhos Melhorada: Incorporou uma nova abordagem para a agregação de caminhos, melhorando a fusão de características de diferentes camadas e permitindo uma melhor detecção de objetos pequenos [37].
	Treinamento e Otimização Avançados: Aplicou técnicas avançadas de treinamento, como a otimização baseada em técnicas de DL mais recentes, resultando em uma melhor generalização do modelo e maior precisão na detecção de objetos de várias escalas [37].
	Eficiência Computacional: Fez melhorias significativas na eficiência computacional, permitindo um equilíbrio melhor entre a velocidade de inferência e a precisão do modelo, especialmente importante para aplicações em tempo real [37].
YOLOv8	CSPDarknet53: Implementada uma nova atualização na arquitetura *backbone*, que incorpora camadas convolucionais avançadas e novos princípios de design, resultando na combinação da CSPNet e a Darknet, melhorando assim a capacidade de extração de dado, permitindo que seja obtido detalhes mais específicos nas imagens [35].

Versão	Melhorias
	Aprimoramento do PANet: Melhorias para facilitar a integração de informações de diferentes escalas, melhorando a previsão geral da detecção [35].
	Atribuição Dinâmica de Âncoras: Melhorias na adaptação das âncoras durante o treinamento, otimizando o modelo para tamanho e formas diversas dos objetos [35].
	Treinamento Aprimorado: Melhorias no fluxograma de treinamento para convergência de resultados mais rápida e melhoria do modelo. Além de ter tornado mais eficiente, obteve melhorias de usabilidade [35].
	Técnicas Avançadas de *Data Augmentation*: Melhorias no *Mosaic*, melhorando assim a capacidade do modelo de generalizar diferentes cenários, melhorando a robustez do modelo [35].

Tabela 1: *Versões do YOLO e suas melhorias*

Segundo [28], a detecção de objetos é realizada utilizando uma única rede neural. A rede usa características da imagem inteira para prever as caixas delimitadoras dos objetos detectados para todas as classes simultaneamente. Isso significa que a rede neural trabalha na imagem como um todo e detecta todos os objetos de uma vez.

A imagem é dividida em uma grade SxS. Se o centro de um objeto cair em uma célula da grade, essa célula será responsável pela detecção do objeto. Cada célula prevê B caixas delimitadoras e suas respectivas pontuações de confiança. Essas pontuações determinam o quão confiante o modelo está de que a caixa delimitadora contém um objeto, bem como a precisão dessa caixa. Caso nenhum objeto exista naquela célula da grade, a pontuação de confiança é considerada zero. Caso contrário, a pontuação será igual à interseção sobre a união (IoU, do inglês *Intersection over Union*) entre a caixa prevista e a caixa verdadeira (*Ground Truth*) [28].

Cada caixa delimitadora consiste em um vetor que contém 5 dados de predição: x, y, w, h e a pontuação de confiança. As coordenadas (x,y) representam o centro da caixa em relação à célula da grade. A largura (w) e a altura (h) são previstas em relação à imagem inteira. Por fim, a previsão de confiança indica o IoU entre a caixa prevista e a caixa verdadeira de base [28].

Cada célula da grade também prevê C probabilidades condicionais de classe, $Pr(Classe_i|Objeto)$. Essas probabilidades são condicionadas à célula da grade conter um objeto. É prevista apenas um conjunto de probabilidades de classe por célula da grade, independentemente do número de caixas B [28].

No momento do teste, são multiplicadas as probabilidades condicionais de classe e as previsões de confiança individuais das caixas, resultando na fórmula:

$$Pr(\text{Classe}_i|\text{Objeto}) \times Pr(\text{Objeto}) \times \text{IoU}_{\text{truth pred}} = Pr(\text{Classe}_i) \times \text{IoU}_{\text{truth pred}} \quad (2)$$

onde:

- $Pr(\text{Classe}_i|\text{Objeto})$ é a probabilidade da classe dado que temos um objeto presente.
- $Pr(\text{Objeto})$ é a probabilidade de que realmente exista um objeto na caixa delimitadora.
- $\text{IoU}_{\text{truth pred}}$ é a métrica que mede a sobreposição entre a caixa delimitadora prevista e a caixa delimitadora da verdade (*ground truth*). A IoU é calculada como a área de interseção dividida pela área de união das duas caixas. Uma IoU mais alta significa que a previsão será mais precisa em relação à posição e ao tamanho do objeto.

[28]

Após a geração das caixas delimitadoras e das respectivas pontuações de confiança, é aplicado o algoritmo de Supressão de Máximos Não-Máximos (NMS, do inglês Non-Maximum Suppression). O NMS é uma técnica utilizada para eliminar caixas redundantes e sobrepostas que se referem ao mesmo objeto. Durante esta etapa, é definido um limiar que determina a sobreposição aceitável entre as caixas delimitadoras previstas, ou seja, o nível de coincidência que é considerado suficiente para que uma caixa seja mantida em vez de eliminada. Um valor padrão comumente utilizado para esse limiar é de 0,5, o que significa que se a interseção entre duas caixas for superior a 50%, uma delas será descartada. Se a IoU entre duas caixas for maior que esse limiar, a caixa com a menor pontuação de confiança é descartada. Esse processo garante que apenas a caixa mais confiável para cada objeto detectado seja mantida [28].

A Figura 2, mostra o fluxo da detecção de objetos.

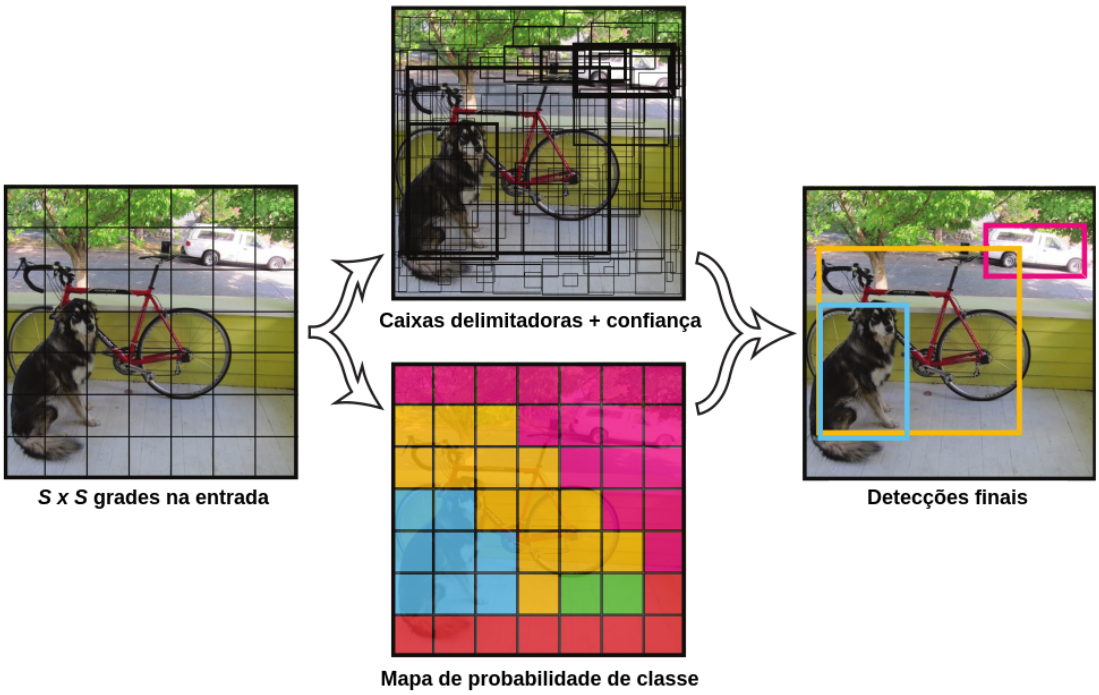

Figura 2: *Fluxo de detecção de objetos - A imagem é dividida em uma grade SxS e, para cada célula, são criadas B caixas delimitadoras, juntamente com a confiança de cada caixa e as suas probabilidades. Após a aplicação do NMS, as caixas redundantes são removidas, resultando em uma detecção final mais precisa e confiável.*
Fonte: Imagem adaptada de [28][1].

Como resultado do processamento, utilizando o exemplo da Figura 2, é obtido uma visualização dos objetos detectados, demarcados por suas respectivas caixas delimitadoras. O YOLO, além de retornar a imagem com as marcações, fornece informações adicionais, como a taxa de confiança (probabilidade de certeza do objeto detectado), as classes detectadas e as coordenadas da caixa delimitadora detectada. Com esses dados, é possível não apenas identificar objetos semelhantes, mas também rastreá-los ao longo do tempo.

2.6 Rastreamento de objetos

O rastreamento de objetos, também conhecido como *object tracking*, é a técnica utilizada na visão computacional para monitorar o movimento e a trajetória de objetos detectados em uma sequência de *frames* de um vídeo [39]. Essa técnica é amplamente aplicada em cenários onde é necessário acompanhar objetos ao longo do tempo, sendo crucial em sistemas de segurança, análise de tráfego e aplicações autônomas que demandam a análise contínua de múltiplos objetos [22]. Realizar a tarefa de rastrear um ou mais objetos ainda é algo desafiador. Um dos principais problemas está relacionado a oclusões, onde um objeto pode ser parcialmente, ou completamente coberto por outro, dificultando sua identificação e acompanhamento contínuo. Além disso, existem outros desafios, como mudanças de aparência, variações de iluminação, posição, ou até mesmo a escala, que podem comprometer a eficácia do rastreamento [39].

Para superar esses obstáculos, novos algoritmos de rastreamento são constantemente desenvolvidos. Segundo [39], entre as técnicas mais avançadas, destacam-se o *ByteTrack* e o *BoT-SORT*, ambos considerados eficientes e robustos para lidar com os desafios mencionados. Estes algoritmos têm se mostrado competentes por abordarem de maneira mais eficaz os problemas de oclusões e mudanças nas condições visuais.

O ByteTrack, por exemplo, utiliza um método de rastreamento por associação, onde tanto as detecções de alta confiança, quanto as de baixa confiança, são associadas aos objetos rastreados, aumentando a precisão, mesmo em situações de oclusão parcial. Esse algoritmo se destaca pelo seu desempenho em cenários onde há múltiplos objetos em movimento e mudanças constantes de contexto [40]. Já o BoT-SORT combina técnicas de otimização de associação, com o uso de re-identificação visual (Re-ID), o que possibilita rastrear objetos de maneira mais precisa, mesmo quando eles saem de cena e retornam. Sua abordagem inovadora permite lidar melhor com objetos que têm aparência similar, ou que mudam de visual durante o rastreamento, minimizando erros causados por essas situações [42].

Essas soluções representam avanços significativos no campo do rastreamento de objetos, resolvendo parte dos desafios mencionados e contribuindo para aplicações práticas mais robustas e eficientes.

2.6.1 ByteTrack

O ByteTrack é um algoritmo de rastreamento de múltiplos objetos (MOT), que se destaca por sua capacidade de associar detecções de alta e baixa confiança, em um processo de rastreamento. Em muitos algoritmos tradicionais de rastreamento, apenas as detecções com alta confiança são consideradas, o que pode resultar na perda de objetos quando eles

são parcialmente ocluídos, ou quando a confiança na detecção é temporariamente baixa. O ByteTrack resolve esse problema utilizando uma abordagem dupla: ele processa tanto as detecções com alta confiança quanto as de baixa confiança, permitindo uma maior robustez em cenários complexos[40].

O algoritmo segue o princípio de dividir as detecções em duas categorias: detecções de alta e baixa confiança. Primeiramente, as detecções de alta confiança são associadas aos objetos já existentes. Em seguida, as detecções de baixa confiança são associadas às detecções não utilizadas no primeiro estágio. Esse método possibilita a recuperação de objetos que poderiam ter sido ignorados em outros sistemas, devido à variação na confiança das detecções. Como resultado, o ByteTrack demonstra um desempenho notável, especialmente em cenários com múltiplos objetos e condições de oclusão parcial [40].

A Figura 3 ilustra o funcionamento do ByteTrack, destacando sua capacidade de rastrear objetos mesmo em situações de oclusão.

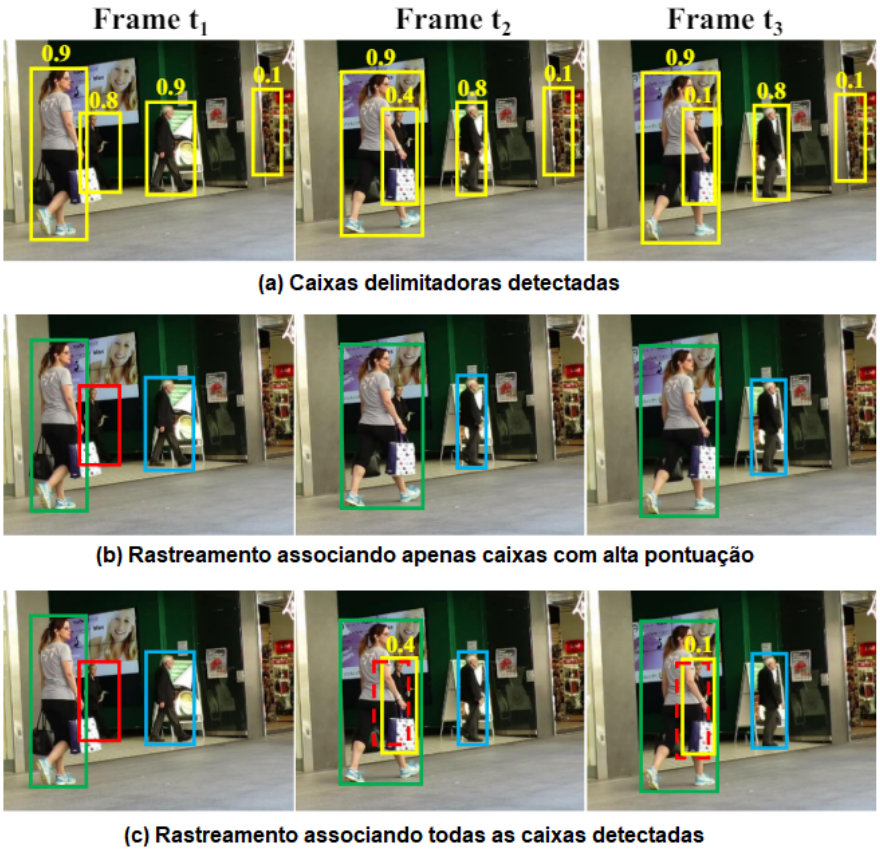

Figura 3: *Exemplo de como o ByteTrack funciona: (a) caixas delimitadoras e suas pontuações, (b) rastreamento feito por outros métodos, onde objetos com pontuações inferiores são descartados, (c) o ByteTrack mostrando que mesmo um objeto em oclusão ainda mantém sua informação, indicado pela caixa tracejada.*
Fonte: Imagem adaptada de [40][2].

Além disso, o ByteTrack implementa um mecanismo de associação de objetos,

que se baseia em algoritmos de associação por maximização de fluxo, como o algoritmo Hungarian. Esse tipo de algoritmo é utilizado para resolver problemas de alocação e otimização, permitindo que o sistema associe corretamente as detecções de objetos, mesmo quando eles estão temporariamente fora da cena ou ocluídos [40]. Essa abordagem é essencial para manter a continuidade do rastreamento ao longo do tempo.

2.6.2 BoT-SORT

O BoT-SORT (*Boosted SORT with Re-Identification*) é uma evolução do algoritmo SORT (*Simple Online and Realtime Tracking*), que integra técnicas de re-identificação visual (Re-ID) para melhorar a precisão e robustez no rastreamento de múltiplos objetos. Enquanto o SORT original se baseia apenas na associação de detecções consecutivas usando o algoritmo de *Kalman Filter* e a métrica de distância Euclidiana para a correspondência de detecções, o BoT-SORT incorpora uma rede neural de Re-ID, que permite identificar e rastrear um objeto com mais precisão, mesmo quando ele sai de cena temporariamente, ou muda de aparência [42][3].

Uma das principais inovações do BoT-SORT é a utilização de Re-ID para associar objetos com aparências semelhantes, ou temporariamente colocados em posição de oclusão. Essa técnica reduz significativamente o erro de associação em cenários com múltiplos objetos visualmente semelhantes. Além disso, o algoritmo aprimora a associação de objetos ao combinar a detecção baseada em movimento, utilizando o *Kalman Filter*, com a correspondência visual proporcionada pelo modelo de Re-ID. Ele também considera a compensação de movimento da câmera, que é crucial em ambientes dinâmicos, onde o rastreamento é impactado tanto pelo movimento do objeto, quanto pelo da própria câmera. Isso permite ao BoT-SORT manter a continuidade do rastreamento, mesmo em situações de alta complexidade visual. [42].

Outro ponto que diferencia o BoT-SORT, de outros algoritmos, é sua capacidade de combinar as informações temporais (trajetória dos objetos) e visuais (características dos objetos) de forma eficiente. Essa integração permite um rastreamento mais robusto, principalmente em ambientes onde objetos podem entrar e sair de cena, ou sofrer alterações na aparência, devido a mudanças de iluminação, ângulo ou posição. Por isso, o BoT-SORT é amplamente utilizado em cenários como vigilância urbana e monitoramento de multidões, onde a robustez contra mudanças visuais é essencial [42].

2.7 Estrutura de um Projeto Web

Um projeto *web* é tipicamente dividido em duas camadas: o *backend* e o *frontend*. O *frontend* é a interface com a qual os usuários interagem, enquanto o *backend* lida com o

processamento dos dados e a lógica de negócio. Essas camadas interagem continuamente, onde as ações do usuário no *frontend* geram requisições que são processadas no *backend*, permitindo uma experiência de usuário dinâmica e responsiva [26].

2.7.1 *Backend*

O *backend* é a parte de um sistema responsável por receber as solicitações, processar os dados e tratá-los, de acordo com a lógica de negócio. É a camada do software que interage diretamente com o servidor e é responsável pelas requisições do cliente, executando as operações necessárias para responder de forma esperada [19].

2.7.1.1 Python

Python é uma linguagem de programação de alto nível, conhecida por sua simplicidade e legibilidade. Ela é amplamente utilizada no desenvolvimento *backend*, devido à sua vasta biblioteca de módulos e *frameworks*, como Django e Flask, que facilitam a criação de aplicações web robustas e escaláveis. Python também é popular para processamento de dados, aprendizado de máquina e automação de tarefas [20].

2.7.1.2 Celery

Celery é uma biblioteca em Python utilizada para gerenciar tarefas assíncronas e enfileiramento de tarefas. Ela permite a execução de tarefas em segundo plano, fora do fluxo principal da aplicação, melhorando a performance e a responsividade. Celery é frequentemente utilizado em conjunto com RabbitMQ ou Redis, como sistemas de mensageria para coordenar e distribuir tarefas [6].

2.7.1.3 RabbitMQ

RabbitMQ é um sistema de mensageria e filas robusto que preza pela confiabilidade. Com suporte a diversos protocolos, incluindo AMQP *(Advanced Message Queuing Protocol)*, ele é classificado como interoperável. Sua flexibilidade se deve à variedade de opções de protocolos de mensagens suportados para a comunicação entre sistemas. RabbitMQ também oferece alta disponibilidade, sendo uma escolha popular para sistemas que necessitam de comunicação confiável e eficiente [25].

2.7.2 *Frontend*

O *frontend* é a camada da aplicação mais próxima do usuário final. Essa camada engloba o design, a interface do usuário (UI, do inglês *User Interface*) e a experiência do usuário (UX, do inglês *User Experience*). O *frontend* é responsável por exibir os dados ao

usuário da melhor forma possível, utilizando frequentemente tecnologias como *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e JavaScript [30].

2.7.2.1 HTML

HTML é a linguagem padrão para criação de páginas *web*. Ele define a estrutura e o conteúdo de um documento *web*, utilizando uma série de elementos e *tags*. HTML é essencial para o desenvolvimento *frontend*, pois serve como a base sobre a qual CSS e JavaScript são aplicados para estilizar e adicionar interatividade às páginas [12].

2.7.2.2 Javascript

JavaScript é uma linguagem de programação, que permite adicionar interatividade e comportamentos dinâmicos às páginas *web*. Ele é executado no navegador do usuário, possibilitando a criação de interfaces responsivas e interativas. JavaScript é essencial para o desenvolvimento *frontend* moderno, sendo a base para *frameworks* e bibliotecas como React, Angular e Vue.js [11].

2.7.2.3 Materialize

Materialize é um *framework frontend* baseado no design de materiais do Google. Ele fornece uma coleção de componentes CSS e JavaScript pré-estilizados, que facilitam a criação de interfaces de usuário modernas e responsivas. Materialize é conhecido por sua estética limpa e funcionalidade intuitiva, permitindo que os desenvolvedores criem rapidamente protótipos e aplicações completas [21].

2.7.2.4 JQUERY

jQuery é uma biblioteca de JavaScript que simplifica a manipulação do Modelo de Objeto de Documento (DOM, do inglês *Document Object Model*), o tratamento de eventos, a animação e a comunicação *Asynchronous JavaScript and XML* (AJAX). AJAX é uma técnica que permite que páginas *web* se atualizem de maneira assíncrona, possibilitando a troca de dados com o servidor sem a necessidade de recarregar a página inteira. Isso resulta em uma experiência de usuário mais dinâmica e responsiva para o usuário. O jQuery fornece uma sintaxe simplificada para realizar requisições AJAX, permitindo que desenvolvedores enviem e recebam dados de forma eficaz. Ele oferece uma sintaxe mais fácil e poderosa para realizar tarefas comuns em JavaScript, reduzindo a quantidade de código necessária. jQuery foi amplamente adotado devido à sua capacidade de resolver inconsistências entre navegadores e facilitar o desenvolvimento *frontend* [18].

3 METODOLOGIA

A metodologia utilizada neste trabalho foi estruturada para proporcionar uma análise detalhada e eficiente da aplicação, desenvolvida para a detecção e rastreamento de objetos em vídeos utilizando YOLOv8 e técnicas de rastreamento, como ByteTrack e BoT-SORT. Este capítulo descreve os métodos e técnicas adotadas em cada fase do desenvolvimento e implementação do sistema, bem como as ferramentas e bibliotecas utilizadas.

3.1 Desenvolvimento do Sistema

O desenvolvimento do sistema foi dividido em duas partes principais: *backend* e *frontend*. O *backend* é responsável por toda a lógica de negócios, processamento de dados e interação com o servidor, enquanto o *frontend* se encarrega da interface do usuário e da experiência visual da aplicação. Essa separação permite uma melhor organização do código e facilita a manutenção e a escalabilidade do sistema. A seguir, são apresentadas as descrições detalhadas de cada uma dessas partes.

3.1.1 *Backend*

O *backend* da aplicação foi desenvolvido utilizando a linguagem Python, devido à sua robustez e vasta gama de bibliotecas voltadas para inteligência artificial (IA) e processamento de imagens. O *framework* Flask foi escolhido para desenvolvimento da aplicação web na parte do *backend*, devido à sua simplicidade e eficiência. O Flask facilita a definição e gerenciamento de rotas, permitindo uma comunicação eficiente entre as diferentes partes da aplicação.

Esta seção descreve os métodos e técnicas adotadas no desenvolvimento e implementação de uma aplicação web para detecção e rastreamento de objetos em vídeos. O foco principal foi a utilização de YOLOv8 para detecção e das técnicas ByteTrack e BoT-SORT para rastreamento.

- **YOLOv8:** A detecção de objetos foi realizada utilizando o algoritmo YOLOv8, uma versão aprimorada do YOLO, conhecido por sua alta precisão e velocidade. YOLOv8 apresenta melhorias significativas em relação às versões anteriores, tornando-o adequado para aplicações em tempo real e em vídeos de alta resolução. As 12 classes utilizadas pela aplicação são resultantes de uma filtragem de classes mais utilizadas dentre aplicações e demonstrações do algoritmo, já existentes, feitas pelo próprio autor.

- **ByteTrack e BoT-SORT:** Para o rastreamento dos objetos detectados, foram utilizados os algoritmos ByteTrack e BoT-SORT. Ambos são conhecidos por sua eficácia em manter a identidade dos objetos ao longo dos *frames* do vídeo. A aplicação dispõe da opção para escolha de qual algoritmo será aplicado. Ambos recebem como parâmetros *frames* processados pela YOLO, contendo a informação da localização dos objetos detectados e aplicam seu algoritmo para fazer o rastreamento.

3.1.1.1 Processamento Assíncrono

Para lidar com o processamento dos vídeos, foi empregado um modelo de processamento assíncrono, utilizando as seguintes ferramentas:

- **Celery:** Biblioteca de gerenciamento de tarefas, que permite a execução de processos em segundo plano e o agendamento de tarefas periódicas. Na versão 5.2.7, o Celery foi utilizado para gerenciar o processamento dos vídeos, garantindo que cada etapa do processo fosse executada de forma eficiente. Para garantir a concorrência de tarefas, o Celery recebe a aplicação como parâmetros, com o objetivo de gerenciar todas as tarefas e processos que ocorrem em segundo plano.
- **RabbitMQ:** Na versão 3.11.13, foi utilizado como *broker* (intermediário de comunicação) de mensagens para o Celery. O RabbitMQ gerencia as filas de tarefas, garantindo a distribuição e o balanceamento de carga entre os *workers* (processos assíncronos). O RabbitMQ foi executado em um *container* Docker, visando a modularização e facilitando o gerenciamento e a escalabilidade do sistema. Devido a sua flexibilidade, o RabbitMQ não requer nenhum parâmetro específico. Ele possui a função de receber os processos e armazená-los em filas.

3.1.2 *Frontend*

O *frontend* foi desenvolvido com HTML, CSS e JavaScript, utilizando a biblioteca Materialize para criar uma interface de usuário responsiva e intuitiva.

- **HTML/CSS:** Estrutura e estilização da interface.
- **Materialize:** *Framework* CSS que facilita a criação de *layouts* modernos e responsivos.
- **jQuery:** Biblioteca JavaScript utilizada para manipulação do *DOM* e requisições assíncronas ao *backend*, melhorando a interatividade da aplicação.

3.2 Tecnologias e Ferramentas Utilizadas

- **Linguagem de Programação:** Python, escolhido por sua robustez e extenso suporte a bibliotecas de Inteligência Artificial (IA).
- **Bibliotecas de Detecção e Rastreamento:** YOLOv8 para detecção, ByteTrack e BoT-SORT para rastreamento.
- **Gerenciamento de Tarefas:** Celery, para execução assíncrona de tarefas.
- **Broker de Mensagens:** RabbitMQ, para gerenciamento de filas de tarefas.
- **Frontend:** HTML, CSS, Materialize para design responsivo, e jQuery para interatividade.

3.3 Fluxo de Funcionamento

1. **Recepção do Vídeo:** O usuário faz o *upload* do vídeo através da interface web, juntamente com os parâmetros escolhidos. A requisição é aceita pelo *frontend* que realiza a trasnferência do arquivo e parâmetros para o backend.A interface foi projetada para ser intuitiva, permitindo que usuários com pouca experiência técnica possam facilmente enviar seus vídeos.
2. **Processamento no Backend:** No *backend*, o vídeo é adicionado a fila do processamento. Uma vez que o vídeo é selecionado para ser processado, ele é passado para o YOLOv8 para que seja feito o processo de detecção. Cada *frame* do vídeo é analisado, identificando e classificando objetos de interesse.
3. **Rastreamento dos Objetos:** Com os objetos detectados, o sistema aplica as técnicas dos algoritmos de rastreamento (ByteTrack ou BoT-SORT), associando os objetos entre os *frames* e mantendo suas identidades ao longo do vídeo, com o máximo de precisão possível.
4. **Geração do Relatório:** Concluído o processamento, o sistema gera um vídeo processado, onde os objetos detectados e rastreados são destacados, e que fica disponível ao usuário para *download*. Além disso, é criado um relatório detalhado, contendo informações como o número total de detecções, total de classes detectadas, total de cada classe detectada, média de confiança das classes detectadas e outras medidas relativas ao tempo de processamento.
5. **Entrega ao Usuário:** O relatório fica disponível na página do resultado do processamento e o vídeo processado fica disponibilizado para *download*, permitindo ao usuário acessar e analisar os resultados de forma prática.

Abaixo na Figura 5, temos um diagrama demonstrando o fluxo de funcionamento da aplicação e também o *frontend* da aplicação.

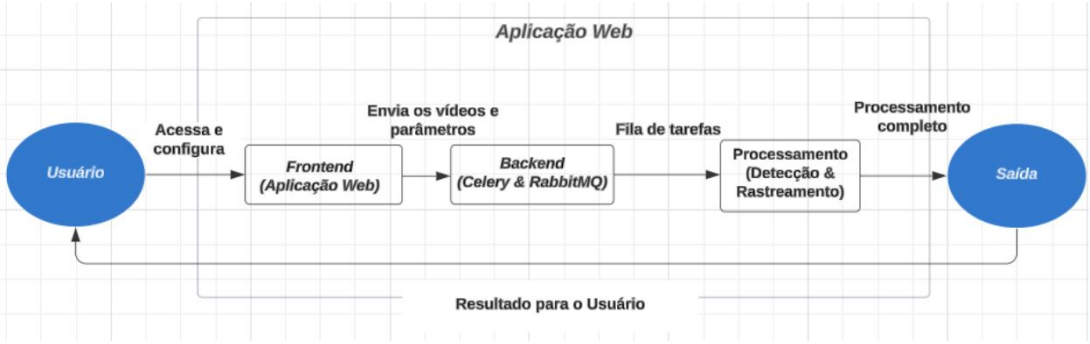

Figura 4: *Diagrama de fluxo de funcionamento da aplicação. Fonte: Autor.*

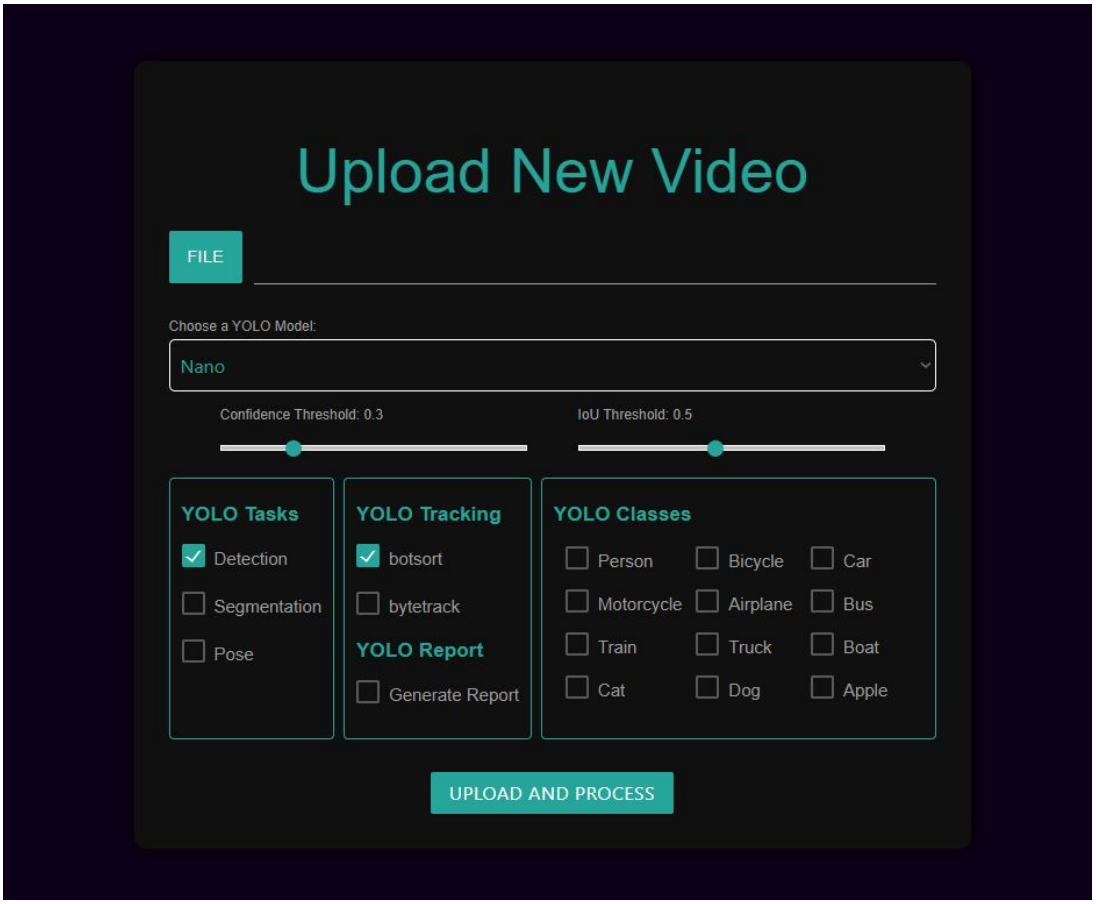

Figura 5: *Frontend da aplicação. Fonte: Autor.*

Com o fluxo descrito, podemos explorar o frontend da aplicação. Na Figura X, é exibida a interface da aplicação, contendo os seguintes elementos:

- **Campo de Upload de Arquivos**: permite o envio do arquivo de vídeo para processamento.

- **Campo de Seleção do Modelo YOLOv8**: possibilita a escolha do modelo YOLOv8 desejado para a detecção de objetos.
- **Seletores de Intervalo (Range)**: controlam os valores de Confiança e IoU, ajustando o grau de precisão da detecção e a sobreposição dos objetos.
- **Campo de Checagem para Tipo de Tarefa**: define a tarefa específica a ser executada pela aplicação, como detecção, segmentação e estimativa de pose.
- **Campo de Checagem para Algoritmo de Rastreamento**: permite selecionar entre os algoritmos de rastreamento disponíveis, como BotSort ou ByteTrack.
- **Campo de Checagem para Geração de Relatório**: oferece a opção de gerar ou não um relatório detalhado ao término do processamento.
- **Campo de Múltipla Escolha para Classes**: possibilita a seleção das classes de objetos que serão detectadas pelo algoritmo.

3.4 Hardware Utilizado

Todo o desenvolvimento, execução da aplicação, processamento dos dados e testes realizados foram feitos utilizando o seguinte hardware:

- **Processador:** Intel Core i7 8700 @ 3.20GHz
- **Placa Mãe:** Gigabyte Technology Co. Ltd. Z370M AORUS Gaming-CF (U3E1)
- **Placa de Vídeo:** GTX 1060 6GB
- **Memória RAM:** 32GB Dual-Channel DDR4 @ 1595MHz (18-22-22-42)

3.5 Testes e Validação

Para garantir a eficácia do sistema, foram realizados testes exaustivos em diversos cenários, aplicando diferentes parâmetros que utilizavam a mesma entrada. Os testes incluíram desde a verificação da precisão das detecções e rastreamentos, o tempo de processamento e a aplicação de diferentes níveis do modelo de detecção.

3.5.1 Comparação de Parâmetros

Um mesmo vídeo foi utilizado com diferentes parâmetros de detecção e rastreamento, para comparar os resultados e avaliar o impacto das alterações no desempenho do sistema. Diversos ajustes foram feitos nos parâmetros do YOLOv8, alternando entre as

variantes *nano, small, medium, large e huge*, além de alternar o algoritmo de rastreamento entre ByteTrack e BoT-SORT, para analisar como essas variações afetam a precisão das detecções, a continuidade do rastreamento e o tempo total de processamento. Esse processo permitiu identificar a configuração mais adequada para diferentes tipos de vídeos, garantindo a robustez e a adaptabilidade da aplicação.

3.5.2 Cenários de Teste

Os testes do sistema foram organizados em dois grupos: o primeiro grupo de testes foi focado em avaliar as funcionalidades do sistema (detecção, segmentação e estimativa de pose), enquanto o segundo grupo teve como objetivo avaliar o desempenho geral do sistema.

As funcionalidades implementadas do YOLOv8 incluem:

- **Detecção:** Identifica e localiza objetos em imagens ou vídeos, fornecendo informações sobre a classe de cada objeto detectado e suas posições na forma de caixas delimitadoras.
- **Segmentação:** Permite a delimitação precisa dos objetos dentro da imagem, diferenciando os *pixels* que pertencem a cada objeto. Essa funcionalidade é essencial para aplicações que exigem uma análise mais detalhada das formas dos objetos.
- **Estimativa de Pose:** Determina a orientação e a posição de seres humanos ou objetos em uma cena, identificando a localização de articulações e a postura, o que é útil em análises de movimento e interação.

Para os testes de funcionalidade, foram utilizados vídeos de diferentes contextos. O primeiro vídeo focou no monitoramento de tráfego, envolvendo classes de objetos como carros, motos, ônibus e caminhões. O segundo grupo de vídeos foi utilizado para avaliar a identificação e a função de segmentação de animais: o primeiro vídeo continha gatos e cachorros para testar a detecção e classificação das espécies, enquanto o segundo vídeo testou a função de segmentação precisa dos objetos detectados. Por fim, o terceiro arquivo, uma imagem foi utilizada para testar a estimativa de pose humana, avaliando a capacidade do sistema de detectar e rastrear múltiplas pessoas em movimento.

Os testes de desempenho foram realizados especificamente com o vídeo de tráfego de veículos, com foco na comparação das variantes do modelo YOLOv8 (*nano, small, medium, large e huge*) e na alternância entre os algoritmos de rastreamento ByteTrack e BoT-SORT. Esses testes permitiram medir o impacto de diferentes parâmetros ajustáveis, como taxa de confiança e limiar de IoU, avaliando a precisão das detecções, a continuidade do rastreamento e o tempo de processamento. O objetivo foi otimizar o desempenho do

sistema, garantindo que ele mantivesse um equilíbrio adequado entre precisão e eficiência em diferentes cenários.

O sistema disponibiliza 12 opções de classes de objetos, incluindo: pessoa, bicicleta, carro, motocicleta, avião, ônibus, trem, caminhão, barco, gato, cachorro e maçã. O usuário tem a flexibilidade de escolher quais classes deseja identificar nos vídeos ou imagens, personalizando os testes conforme o cenário de aplicação.

3.5.3 Metodologia dos Testes

Nesta seção, é detalhada a metodologia utilizada para realizar os testes de desempenho e precisão. O sistema está apto para aceitar arquivos de imagem e vídeos, limitado ao tamanho de 64MB e aos tipos que utilizam as seguintes extensões de arquivo: *"png", "jpg", "jpeg", "gif", "bmp", "mp4", "mov", "avi", "webm"*.

3.5.4 Padrão de Teste

- **Procedimento de Teste:** Foi utilizado o mesmo arquivo de vídeo como entrada, variando os parâmetros para observar os impactos das mudanças, tanto no tempo de processamento, quanto no desempenho de cada propriedade aplicada.
- **Modelos YOLO:** Os testes foram executados para todos os níveis de modelo da versão da YOLOV8 (*nano, small, medium, large, huge*). Esses modelos são baseados no nível de processamento exigido, ou seja, quanto maior o nível, mais processamento é necessário para sua execução, demandando assim mais tempo.
- **Parâmetros de Detecção:** A aplicação dispõe para o usuário a escolha de parâmetros como a Taxa de Confiança e a Taxa do IoU, onde o usuário vai escolher um valor entre 0 e 1,0. O valor do intervalo vai indicar a porcentagem das respectivas taxas. A Taxa de Confiança representa o valor de certeza mínimo necessário, em porcentagem, para detecção de um objeto e é iniciada com valor padrão de 0,3. A Taxa de IoU representa a porcentagem de sobreposição entre a área delimitada pela detecção do objeto e a área real do objeto, e é iniciada com valor padrão de 0,5.
- **Algoritmo de Rastreamento:** Em conjunto com a detecção, é executado o algoritmo de rastreamento selecionado pelo usuário (BoT-SORT ou ByteTrack).
- **Métricas Avaliadas:** Ao fim do processo, é gerado um relatório com dados obtidos com o processamento do arquivo.

4 APRESENTAÇÃO E ANÁLISE DOS RESULTADOS

Neste capítulo, serão apresentados os resultados obtidos, a partir dos experimentos realizados com a aplicação web desenvolvida. Primeiramente, serão detalhados os testes de funcionalidade, que incluem a detecção, segmentação e estimativa de pose. Em seguida, abordaremos o teste de desempenho da YOLO, que envolve o teste das taxas de acurácia na detecção e rastreamento de objetos, além de uma análise comparativa entre os diferentes parâmetros utilizados.

4.1 Teste de Funcionalidade

O teste de funcionalidade foi dividido em três partes, cada uma focada em uma característica específica da aplicação: detecção, segmentação e estimativa de pose. Na Tabela 2 são apresentadas informações descritivas dos arquivos utilizados.

Nome	**Tipo**	**Resolução**	**Tamanho**
Traffic2.mp4	vídeo	3840 × 2160	32MB
Cat-Dog-Eating.webm	vídeo	426 × 240	1,4MB
Pose-estimation.jpg	imagem	744 × 580	121KB

Tabela 2: *Dados referentes aos arquivos utilizados para o teste de funcionalidade*

- **Detecção de Objetos:** O primeiro vídeo utilizado foi de tráfego em uma via, com o objetivo de identificar objetos selecionados, como carros e caminhões. Para ilustrar este teste, a Figura 6 mostra um *frame* retirado do vídeo, no qual os objetos detectados estão marcados com suas caixas delimitadoras. Cada objeto possui um rótulo com um identificador, sua classe e a taxa de confiança associada. Esse teste teve como objetivo verificar a eficácia da aplicação na detecção de diferentes classes de veículos em um ambiente dinâmico.

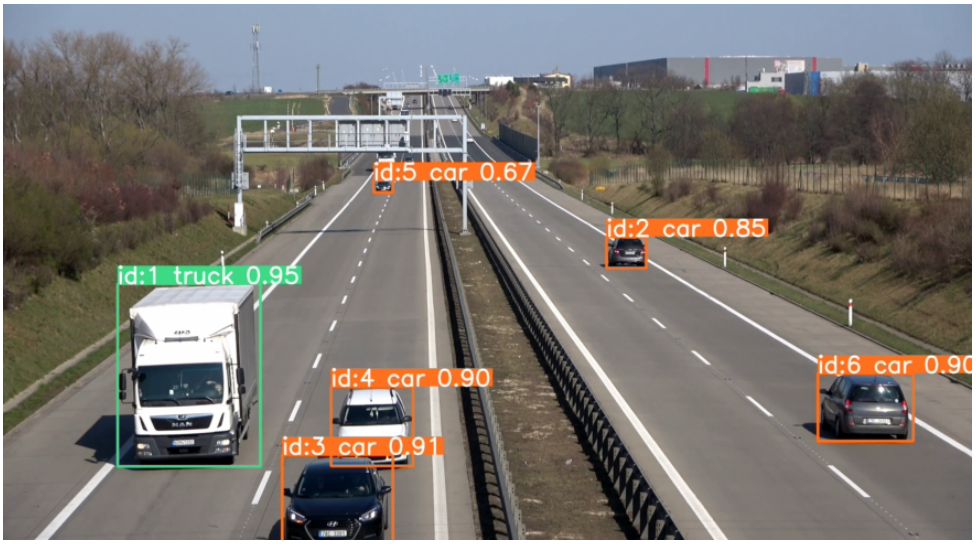

Figura 6: *Imagem do vídeo de tráfego com objetos detectados. Cada objeto detectado recebe um identificador, sua classe e a sua confiança. Fonte: Imagem adaptada de [36].*

- **Segmentação de Objetos:** No segundo vídeo, foi utilizada a funcionalidade de segmentação do YOLOv8, para demarcar as classes de objetos detectados com cores distintas. A Figura 7 a seguir ilustra um gato e um cachorro, cada um segmentado em uma cor diferente. Este teste teve como objetivo avaliar a capacidade do sistema de segmentar visualmente os objetos, facilitando a identificação e a análise de cada classe presente na cena.

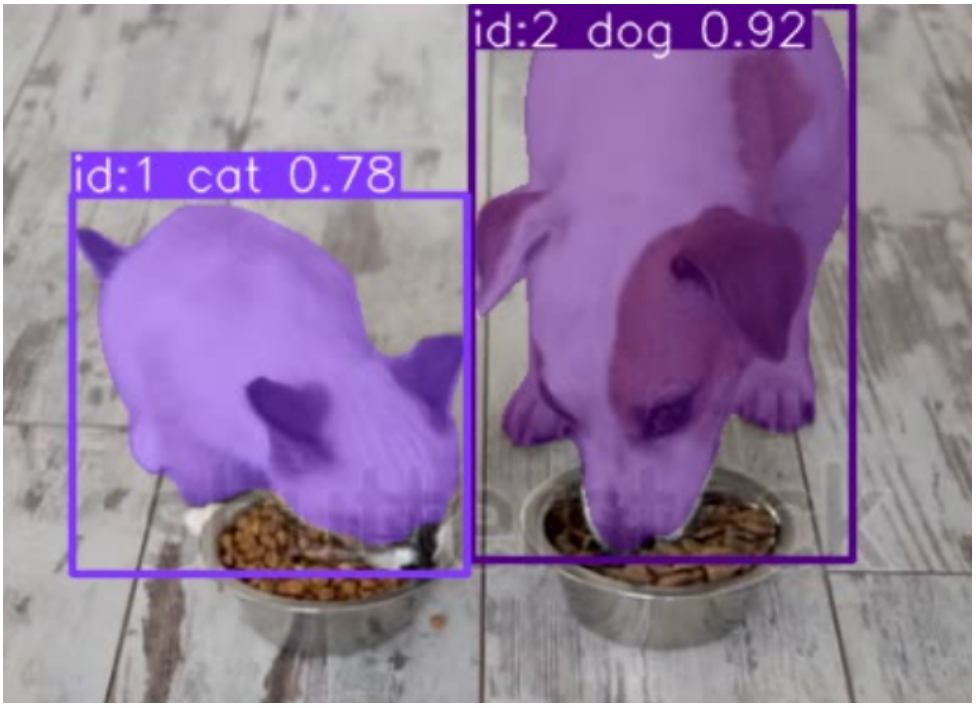

Figura 7: *Imagem do vídeo mostrando objetos das classes "cat" e "dog" detectados, aos quais foi aplicado o filtro de segmentação. Cada objeto detectado é rotulado com um identificador, sua classe — representada por uma cor exclusiva — e a taxa de confiança da detecção. Fonte: Shutterstock (Adaptada por Autor).*

- **Estimativa de Pose:** Por fim, foi utilizada uma imagem contendo pessoas para testar a funcionalidade de estimar a pose. Este teste buscou verificar a precisão da aplicação em identificar a posição e a orientação dos indivíduos na imagem. A Figura 8 ilustra a estimativa de pose realizada a partir de uma imagem.

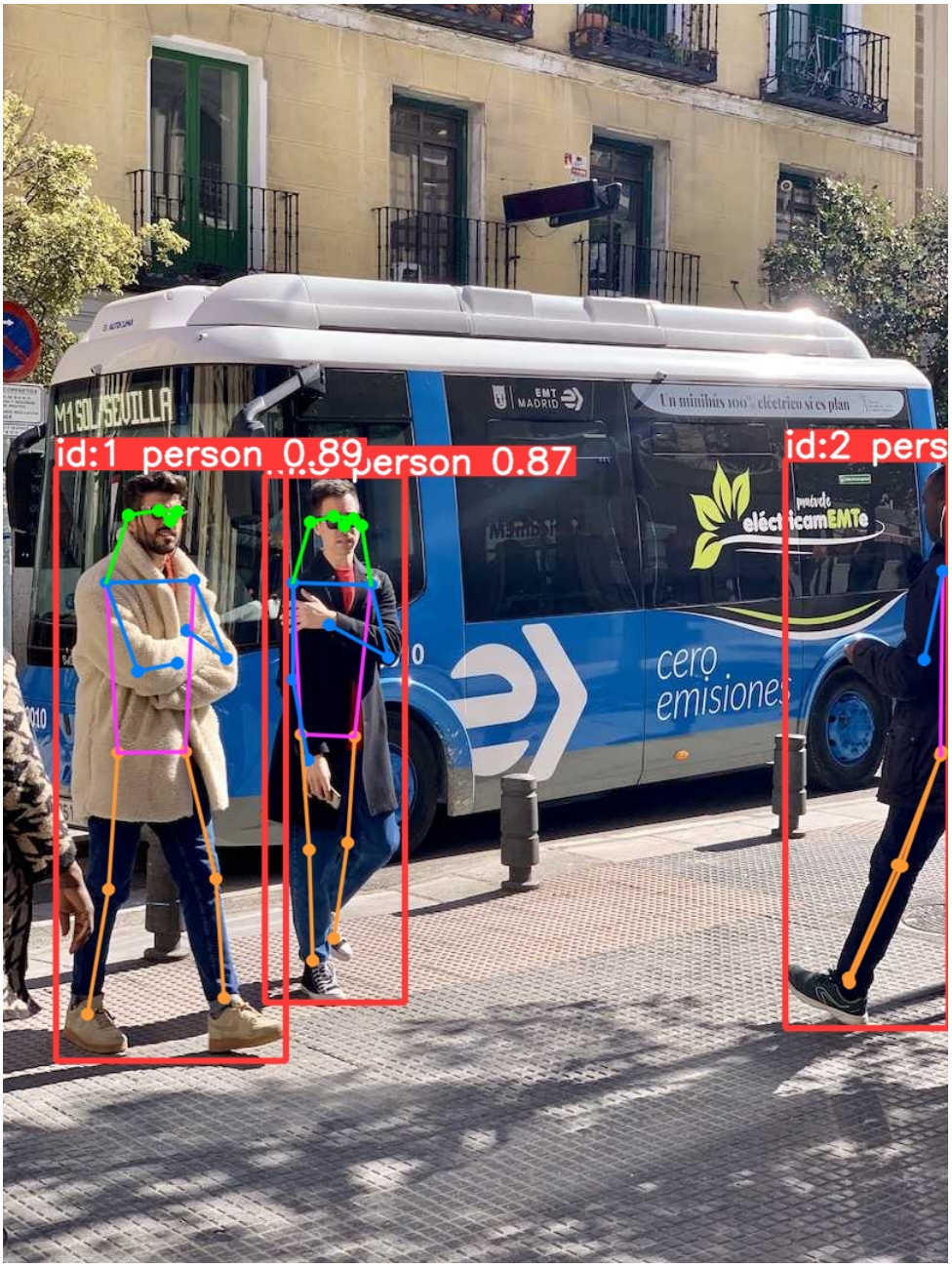

Figura 8: *Imagem mostrando objetos da classe 'person' com a estimativa de pose de cada indivíduo detectado. Fonte: Imagem adaptada de [36].*

Os testes de funcionalidade realizados demonstraram que a aplicação atende de forma satisfatória os requisitos propostos. A detecção de objetos no vídeo de tráfego foi eficaz, identificando com precisão os veículos selecionados. A funcionalidade de segmentação provou ser útil na visualização das classes de objetos, permitindo uma análise clara e distinta através do uso de cores exclusivas. Por fim, a estimativa de pose das pessoas foi realizada com sucesso, evidenciando a robustez do sistema em reconhecer e analisar diferentes posturas.

4.2 Teste de desempenho do YOLOV8

O teste de desempenho tem como objetivo avaliar a eficiência do modelo YOLOv8, em conjunto com os algoritmos de rastreamento BoT-SORT e ByteTrack. O foco principal desse teste não é apenas a detecção de objetos, mas também a medição dos tempos de processamento e o impacto dos algoritmos de rastreamento. Para realizar o teste, foi utilizado um arquivo de vídeo como entrada, e cada parâmetro dos modelos da YOLOv8 foi testado em conjunto com cada um dos algoritmos de rastreamento.

A Tabela 3 disponibiliza as informações referentes ao arquivo utilizado para os testes.

Nome	**Tipo**	**Resolução**	**Tamanho**
Traffic.mp4	vídeo	1280 × 720	4.86MB

Tabela 3: *Dados referentes ao arquivo utilizado para o teste de desempenho*

Após a execução dos testes, fazendo uso dos dados obtidos, pode-se observar a variação de desempenho do sistema com a variação dos níveis de processamento. Neste ponto está sendo medido apenas o processamento dos dados em *frames* por segundo (FPS). Nas Tabelas 4 e 5, são descritos os dados obtidos utilizando o BoT-Sort e o ByteTrack.

YOLOv8 Model	**Processamento Total(s)**	**FPS**
Nano	8,32	61
Small	10,09	50
Medium	15,19	33
Large	22,28	22
Huge	30,71	16

Tabela 4: *Dados de processamento relacionados a Taxa de Frames do arquivo, totalizando 508 quadros, utilizando BoT-SORT com YOLOv8, extraídos do relatório gerado*

YOLOv8 Model	**Processamento Total(s)**	**FPS**
Nano	7,82	64
Small	9,23	55
Medium	15,17	33
Large	21,99	23
Huge	30,46	16

Tabela 5: *Dados de processamento relacionados a Taxa de Frames, totalizando 508 quadros, utilizando ByteTrack com YOLOv8, extraídos do relatório gerado*

A comparação entre os dados das Tabelas 4 e 5 mostra que o nível do modelo YOLOv8 tem um impacto direto no desempenho da aplicação. Modelos mais complexos

resultam em uma redução significativa na taxa de quadros. Além disso, o tipo de algoritmo de rastreamento utilizado também influencia o desempenho da aplicação. Neste caso, o ByteTrack se mostrou levemente superior, especialmente nos modelos menores, com um ganho de desempenho de até 9,35% no modelo *Small*.

4.3 Acurácia na Detecção e Rastreamento

Nesta seção, são apresentados os resultados relacionados à acurácia na detecção e rastreamento de objetos, utilizando os diferentes modelos da YOLOv8 (*Nano, Small, Medium, Large e Huge*) e aplicando os dois algoritmos de rastreamento disponbilizados pela aplicação. O vídeo utilizado para o teste é de uma rodovia onde carros e caminhões trafegam, possui 26 carros e 7 caminhões ao longo do vídeo.

Os dados numéricos apresentados nas Tabelas 6 e 7 são relativos a quantidade de cada item.

Model	**Objetos**	**Classes**	**Classe Individual**	**Confiança**	**Acurácia**
Nano	32	3	*Car*: 28 *Train*: 1 *Truck*: 3	65%	96.97%
Small	32	2	*Car*: 25 *Truck*: 7	68%	93.94%
Medium	30	2	*Car*: 24 *Truck*: 6	68%	90.91%
Large	35	2	*Car*: 26 *Truck*: 9	70%	94.29%
Huge	36	2	*Car*: 26 *Truck*: 10	67%	91.67%

Tabela 6: *Dados de processamento relacionados a Acurácia e Detecção de objetos utilizando BoT-SORT com YOLOv8, extraídos do relatório gerado*

Model	Objetos	Classes	Classe Individual	Confiança	Acurácia
Nano	35	3	*Car*: 31 *Train*: 1 *Truck*: 3	65%	94.29%
Small	36	2	*Car*: 29 *Truck*: 7	68%	91.67%
Medium	38	2	*Car*: 31 *Truck*: 7	72%	86.84%
Large	42	2	*Car*: 32 *Truck*: 10	73%	78.57%
Huge	43	2	*Car*: 31 *Truck*: 12	69%	76.74%

Tabela 7: *Dados de processamento relacionados a Acurácia e Detecção de objetos utilizando ByteTrack com YOLOv8, extraídos do relatório gerado*

A acurácia é uma métrica utilizada para avaliar a qualidade da detecção de objetos, indicando o quão precisa é a identificação dos objetos em relação ao número total de objetos presentes na cena. No contexto deste trabalho, a acurácia foi calculada considerando o número total de objetos presentes no vídeo de entrada e o número de objetos detectados por cada modelo YOLOv8.

Para calcular a acurácia de cada modelo, utilizamos a seguinte fórmula:

$$\text{Acurácia} = \left(\frac{\text{Objetos Reais}}{\text{Objetos Reais} + \text{Objetos Excedentes}} \right) \times 100\% \tag{3}$$

- **Objetos Reais**: número total de objetos que deveriam ser detectados (neste caso, 33 objetos).
- **Objetos Excedentes**: diferença entre o número total de objetos detectados pelo modelo e o número de objetos reais. Esta métrica indica a quantidade de detecções incorretas ou duplicadas feitas pelo modelo.

O objetivo de calcular a acurácia desta forma é entender o quão preciso é o modelo ao detectar os objetos e avaliar o impacto das detecções erradas. Uma acurácia maior mostra que o modelo identifica corretamente a maioria dos objetos reais, enquanto valores menores indicam mais erros ou detecções desnecessárias. Este cálculo ajuda a comparar o desempenho dos modelos de forma clara e objetiva.

Os dados apresentados nas Tabelas 6 e 7 mostram que, à medida que o modelo da YOLOv8 se torna mais complexo (do *Nano* ao *Huge*), há um aumento no número de objetos detectados. Embora o número de classes detectadas permaneça constante, com a inferência dos algoritmos de rastreamento em ambos os casos, o número de objetos

detectados foi maior no caso do ByteTrack, assim como a Taxa de Confiança dos objetos identificados. Os resultados esperados indicavam a presença de 26 carros e 7 caminhões no vídeo utilizado. No entanto, alguns carros estavam acoplados a reboques, o que pode ter gerado a identificação incorreta de caminhões e carros adicionais em certos ângulos. Essa configuração, juntamente com a dinâmica da detecção, contribuiu para a ocorrência do aumento de falsos positivos.

Houve também um caso comum em ambos os testes, na execução do modelo *Nano*, foi observada a presença de um falso positivo para a classe 'Train'. Essa situação pode ser atribuída ao fato de que o *Nano* prioriza a velocidade, o que pode influenciar sua capacidade de identificação.

4.4 Intervalo de Processamento

Esta seção apresenta os tempos totais de processamento, conforme mostrados nas Tabelas 8 e 9, complementando as informações dos testes realizados.

YOLOv8	**Pré-processamento (s)**	**Inferência (s)**	**Pós-processamento (s)**
Nano	0,99	5,93	1,14
Small	0,99	6,32	1,12
Medium	1	15,87	1,29
Large	0,97	18,40	1,22
Huge	0,97	26,71	1,22

Tabela 8: *Tempos totais de processamento de pré-processamento, inferência e pós-processamento utilizando diferentes modelos YOLOv8 com BoT-SORT, extraídos do relatório gerado*

YOLOv8	**Pré-processamento (s)**	**Inferência (s)**	**Pós-processamento (s)**
Nano	0,96	5,47	1,10
Small	0,98	6,86	1,16
Medium	0,97	11,87	1,22
Large	0,99	18,26	1,24
Huge	0,95	26,52	1,26

Tabela 9: *Tempos totais de processamento de pré-processamento, inferência e pós-processamento utilizando diferentes modelos YOLOv8 com Bytetrack, extraídos do relatório gerado*

4.5 Relatório

O relatório gerado após o processamento, caso o usuário selecione a opção *Generate Report*, fornece uma visão detalhada dos resultados obtidos. O relatório apresenta a

quantidade de objetos detectados, as classes identificadas, o total de cada classe e a média de acurácia. Além disso, inclui os tempos de processamento de cada modelo YOLOv8 utilizado, abrangendo desde o pré-processamento, a inferência do modelo utilizado e o pós-processamento, facilitando a análise de desempenho da aplicação.

Como mostrado na Figura 9, o relatório disponibiliza ao usuário dois tipos de informação, a primeira parte é sobre detecções e a segunda parte é sobre informações do processamento do video.

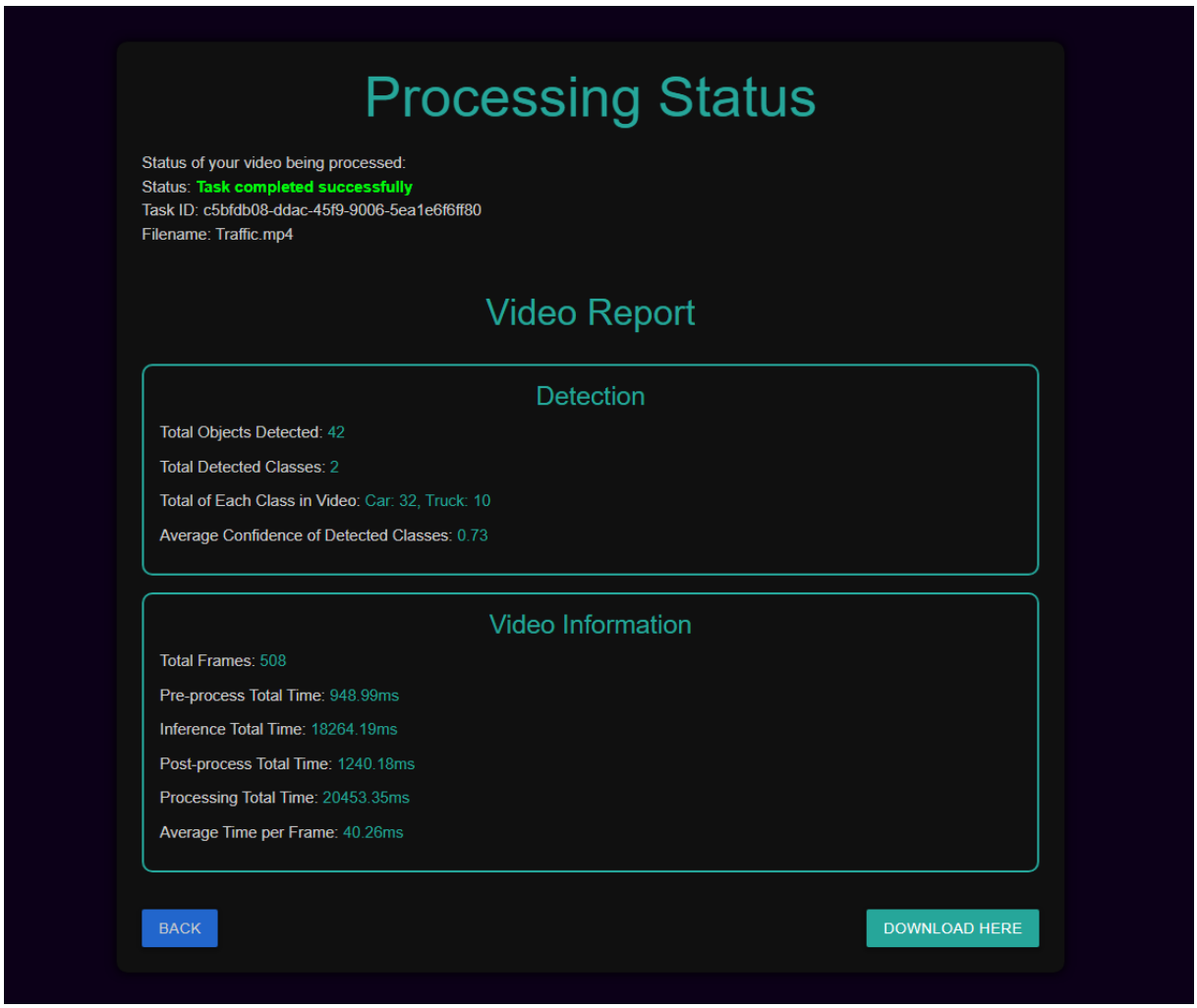

Figura 9: *Relatório do vídeo do caso YOLOv8 Large com rastreamento ByteTrack*

4.6 Conclusão dos Resultados

A análise dos resultados obtidos com a aplicação mostra a eficácia da YOLOv8 na detecção de objetos em conjunto com os algoritmos de rastreamento BoT-SORT e ByteTrack. É notável que o uso de modelos mais complexos da YOLOv8 (*Large* e *Huge*) traz melhores resultados, em termos de quantidade de objetos detectados e acurácia, em ambos os algoritmos de rastreamento.

Os testes demonstram que, embora o tempo de processamento aumente com a complexidade do modelo, a precisão dos objetos detectados justifica o uso de modelos mais robustos e complexos, em cenários que demandam maior acurácia. Os testes foram realizados diversas vezes com o objetivo de assegurar a consistência dos dados obtidos. Com base nesses dados, é possível observar que o ByteTrack traz uma resposta levemente mais rápida em modelos com um pouco menos de tempo de procesamento. Contudo, o ByteTrack também tende a detectar mais objetos e gerar mais falsos positivos.

Dessa forma, os dados gerados pelos relatórios fornecem uma base sólida para avaliar e comparar o desempenho dos diferentes modelos e configurações, auxiliando na escolha da melhor abordagem para o uso da aplicação.

5 CONCLUSÕES E TRABALHOS FUTUROS

Os resultados obtidos nos testes realizados mostram que a aplicação desenvolvida para detecção e rastreamento de objetos apresentou um bom desempenho. A combinação do YOLOv8 com os algoritmos de rastreamento BoT-SORT e ByteTrack se mostrou promissora, ao alcançar níveis satisfatórios de acurácia e precisão no rastreamento de objetos no cenário testado. Observou-se que os modelos mais complexos da YOLOv8 (*Large* e *Huge*), embora exijam maior capacidade de processamento, devido ao número maior de camadas na rede neural, entregaram os melhores resultados em termos de quantidade de objetos detectados e precisão das detecções.

O desempenho da aplicação foi influenciado pelas limitações do hardware utilizado, já que os testes foram conduzidos em um computador pessoal. Em um ambiente especializado, com maior poder de processamento, espera-se que a aplicação apresente resultados ainda mais eficientes, sendo recomendada a adoção de plataformas na nuvem, projetadas para lidar com processamento intensivo de dados e IA.

Conclui-se que o projeto atingiu os objetivos propostos, demonstrando a viabilidade e a eficiência de utilizar o YOLOv8 para detecção de objetos, em arquivos de imagens ou vídeos. No entanto, é importante notar que foram realizados apenas poucos testes, e a inclusão de uma maior diversidade de vídeos poderá oferecer uma análise mais robusta. Assim, ainda existem oportunidades de melhorias e atualizações futuras que podem aumentar a funcionalidade e a precisão da aplicação.

Para trabalhos futuros, algumas melhorias podem ser implementadas na aplicação. Entre elas, destaca-se manter a aplicação sempre atualizada com o modelo YOLO de versões mais recentes. A plataforma também pode ser aprimorada para aceitar não apenas arquivos de imagem ou vídeo, mas também *links* externos de vídeos, como vídeos do YouTube ou *streams* de vídeo em tempo real. Implementar a exibição do vídeo resultante gerado diretamente na página da aplicação. Outra possível melhoria seria a inclusão de *logs* de processamento em tempo real, permitindo que os usuários visualizem os resultados na própria página e façam *download* do *log* de atividades em formato PDF.

REFERÊNCIAS

[1] Alireza Adineh. A survey of different approaches for real-time traffic sign detection and recognition. *International Journal of Research Publication and Reviews*, 4(5):1938–1943, 2023. Available at: `https://ijrpr.com/uploads/V4ISSUE5/IJRPR12846.pdf`.

[2] Manya Afonso and Valerio Giufrida. Synthetic data for computer vision in agriculture. *Frontiers in Plant Science*, 14, 2023.

[3] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Sort: A simple, online and real-time tracker. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468. IEEE, 2016.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[5] M. A. Brovelli, Y. Sun, and V. Yordanov. Monitoring forest change in the amazon using multi-temporal remote sensing data and machine learning classification on google earth engine. *ISPRS International Journal of Geo-Information*, 9(10):580, 2020.

[6] Celery Project. *Celery Documentation*, 2023.

[7] Deloitte. Ai computer vision solutions architecture, 2023. Disponível em:: `https://www2.deloitte.com/content/dam/Deloitte/us/Documents/deloitte-analytics/us-da-ai-computer-vision-solutions-architecture.pdf`.

[8] S. Oleiro Araújo et al. Machine learning applications in agriculture: Current trends, challenges, and future perspectives. *Agronomy*, 13(12):2976, 2023.

[9] Abdelfatah Ettalibi, Abdelmajid Elouadi, and Abdeljebar Mansour. Ai and computer vision-based real-time quality control: A review of industrial applications. *Procedia Computer Science*, 2023. Disponível em: `https://www.sciencedirect.com/science/article/pii/S187705092300491X`.

[10] L. C. A. Filho and G. C. da Conceição. Impactos da inteligÊcia artificial na sociedade. *Revista Interface Tecnológica*, 20(2):134–145, 2023.

[11] David Flanagan. *JavaScript: The Definitive Guide: Master the World's Most-Used Programming Language*. O'Reilly Media, 7th edition, 2020.

[12] Elisabeth Freeman and Eric Robson. *Head First HTML and CSS*. O'Reilly Media, 2nd edition, 2014.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[14] Simon Haykin. *Neural Networks and Learning Machines*. Pearson, 3 edition, 2009.

[15] IBM. What is computer vision?, 2023. Disponível em:: `https://www.ibm.com/topics/computer-vision`.

[16] Muhammad Fazal Ijaz and Marcin Woźniak. Recent advances in deep learning and medical imaging for cancer treatment. *Cancers*, 16(4):0700, 2024.

[17] Glenn Jocher. Yolov5. GitHub repository, 2020.

[18] jQuery Foundation. jquery documentation. `https://api.jquery.com/`, 2021. Accessed: 2024-06-08.

[19] Kenneth C. Laudon and Jane P. Laudon. *Sistemas de Informação Gerenciais: administrando a empresa digital*. Pearson Prentice Hall, São Paulo, 10 edition, 2007.

[20] Mark Lutz. *Learning Python*. O'Reilly Media, 5th edition, 2013.

[21] Materialize. Materialize: A modern responsive front-end framework based on material design. `https://materializecss.com/`, 2021. Accessed: 2024-06-08.

[22] K. Murali and G. Varghese. A survey on object detection in autonomous vehicles. *Journal of Intelligent Transportation Systems*, 26(1):1–21, 2022.

[23] Sangam B. Neupane, Kazuhiko Sato, and Bishnu P. Gautam. A literature review of computer vision techniques in wildlife monitoring. *International Journal of Scientific and Research Publications*, 16:282–292, 2022. Available at: `https://www.researchgate.net/publication/366005635`.

[24] National Institute of Justice. Using artificial intelligence to address criminal justice needs, 2023. Disponível em:: `https://nij.ojp.gov/library/publications/using-artificial-intelligence-address-criminal-justice-needs`.

[25] Pivotal Software, Inc. *RabbitMQ Documentation*, 2023.

[26] Filipe Portela and Ricardo Queirós. *Introdução ao Desenvolvimento Moderno Para a Web: Do Front-End ao Back-End: uma visão global*. FCA, Lisboa, 2018.

[27] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[28] Joseph Redmon and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[29] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[30] Elisabeth Robson and Eric Freeman. *Use a Cabeça! HTML e CSS*. Alta Books, Rio de Janeiro, 1 edition, 2013.

[31] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2020.

[32] Katsuhiko Sakamoto, Fumihiro Kimura, and Kenji Fujimura. Real-time visual inspection system for detecting defects in products on the production line, 2023.

[33] Applied Sciences. Investigating the effect of outdoor advertising on consumer decisions: An eye-tracking and a/b testing study of car drivers' perception. *Applied Sciences*, 13(11):6808, 2023. Disponível em: `https://www.mdpi.com/2076-3417/13/11/6808`.

[34] Sensors. Real-time threat detection in surveillance systems using ai. *Sensors*, 2023. Disponível em: `https://www.mdpi.com/journal/sensors`.

[35] Ultralytics. Yolov8: The next evolution of yolo. `https://github.com/ultralytics/yolov8`, 2023.

[36] Ultralytics. Yolo by ultralytics, 2024. Available at: `https://docs.ultralytics.com/`, urldate=2024-05-26.

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

[38] Chien-Yao Wang, Yang Li, Hsieh-Yang Mark Liao, et al. Yolov6: A single-stage object detection framework for real-time applications, 2022.

[39] Papers with Code. Multi-object tracking. `https://paperswithcode.com/task/multi-object-tracking`, 2024. Accessed: 2024-06-08.

[40] Yifu Wu and Zhaoxiang Zhang. Bytetrack: Multi-object tracking by associating every detection box. *arXiv preprint arXiv:2110.06864*, 2022. Accessed: 2024-06-08.

[41] Huanhuan Zhang and Yufei Qie. Applying deep learning to medical imaging: A review. *Applied Sciences*, 13(18):10521, 2023.

[42] Yu Zhang and Xinggang Wang. Botsort: Robust association multi-pedestrian tracking. *arXiv preprint arXiv:2206.14651*, 2022. Accessed: 2024-06-08.