UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

BEATRIZ FRÉCCIA AMANTE

RASTREAMENTO DE PESSOAS POR TÉCNICAS DE APRENDIZAGEM DE MÁQUINA

GUARAPUAVA

2025

BEATRIZ FRÉCCIA AMANTE

RASTREAMENTO DE PESSOAS POR TÉCNICAS DE APRENDIZAGEM DE MÁQUINA

Person tracking using machine learning techniques

Projeto de Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Tecnólogo em Tecnologia em Sistemas para Internet do Curso Superior de Tecnologia em Sistemas para Internet da Universidade Tecnológica Federal do Paraná.

Orientadora: Drª Kelly Lais Wiggers

GUARAPUAVA

2025

AGRADECIMENTOS

Agradeço, primeiramente, a minha orientadora Profa. Dra. Kelly Lais Wiggers, por sua orientação, paciência e apoio incondicional. Há momentos em que o mundo parece assustador e muito grande para nós. Sua amizade e humanidade fizeram toda a diferença, me dando esperanças para seguir em frente de cabeça erguida.

Aos professores, também. Se você que está lendo isso, também teve a sorte de estudar no curso de Sistemas para Internet no Campus da UTFPR de Guarapuava, saiba que você está entre mestres e doutores incrivelmente qualificados e humanos. Minha mais eterna gratidão.

Incluo aqui também um *spotlight* especial ao Professor Doutor Eleandro Maschio, o qual eu admiro grandemente. Saiba, professor, se você ler essas linhas, lhe admiro profundamente e o vejo como inspiração, assim como minha orientadora.

À minha família, mesmo distante fisicamente, ainda assim me apoiou nesses anos de forma que apenas aqueles que nos conhecem como a palma da mão são capazes. Aos meus pais, Maria Fréccia Amante e Vilmar Amante Filho, que mesmo com todos os erros e acertos, sempre quiseram o melhor pra mim e torceram para que eu chegasse onde cheguei, e aonde ainda chegarei. Ao meus irmãos, também, Samuel e George. Eu não sou uma pessoa religiosa, mas se há um Deus, eu agradeço todos os dias que tive o prazer de crescer ao lado deles, pois algumas pessoas são como um presente divino.

Ao grupo de TI da Cooperativa Agraria, por moldar meu início profissional. À Codeminer42, que me fez, e faz, tão realizada profissionalmente. Vocês são incríveis, e eu serei eternamente grata a tudo que aprendi, e ainda aprendo, lá dentro.

Queria agradecer, também, à família que escolhemos, aos laços que não são de sangue, mas continuam tão significativos quanto. Esses foram os três anos mais carregados da minha vida. Aconteceram tantas coisas, e constantemente eu me sentia sobrecarregada, numa cidade que não é minha, cuidando de problemas que não eram meus, preenchendo minha cabeça apenas de coisas sombrias, sem ver uma luz no final do túnel. Perdi a conta de quantas vezes pensei em desistir, sem família para apoiar nas proximidades, com a vida pessoal conturbada por anos de problemas acumulados e, quando as coisas alcançaram um nível insustentável, quem me segurou foram vocês. Bruno, Roberta, Patrícia, Carolina, Tiago, Benjamin, Santiago... São tantos nomes, mas vocês sabem que estão todos aqui.

Saibam que independente de onde a vida nos leve, vocês sempre terão um lugar ao meu lado.

Obrigada por existirem.

Por ultimo, ao leitor:

As vezes a vida parece insustentável.

As vezes nós nos diminuímos para caber em espaços que não nos condizem, ou nos calamos para mantermos a paz, pois crescemos em ambientes conturbados que nos fazem acreditar que afeto e dor andam de mãos dadas.

Vocês não estão sozinhos.
Vocês são o suficiente.
Eu prometo que isso passará.
Não desistam de si mesmos. Dos seus sonhos.
O sol volta, mesmo depois da noite mais longa

RESUMO

Este trabalho propõe o desenvolvimento de uma API *RESTful* acessível e personalizável para rastreamento e reidentificação de pessoas em vídeos, utilizando técnicas de aprendizado de máquina e visão computacional. A proposta busca democratizar o uso de sistemas baseados em inteligência artificial (IA) ao integrar modelos de detecção, rastreamento e extração de características visuais em uma interface funcional e intuitiva. Tais tecnologias, embora amplamente estudadas, ainda enfrentam barreiras de adoção devido à complexidade técnica e à escassez de ferramentas públicas e bem documentadas. A solução desenvolvida permitirá identificar indivíduos em transmissões ao vivo ou vídeos enviados, com suporte à adição de identificadores únicos e proteção da privacidade por meio de técnicas de criptografia. A iniciativa visa preencher lacunas existentes entre bibliotecas de pesquisa e aplicações práticas, oferecendo uma plataforma modular aplicável em contextos como segurança urbana, automação, controle de acesso e projetos sociais. O projeto utiliza bibliotecas consolidadas como YOLO e OSnet aliadas a *frameworks web* como *FastAPI*, para construir um *pipeline* de rastreamento *end-to-end* com foco em desempenho e acessibilidade por desenvolvedores e pesquisadores.

Palavras-chave: inteligencia artificial; deep learning; people tracking; re-identification; real-time video.

ABSTRACT

This project proposes the development of an accessible and customizable RESTful API for person tracking and re-identification in videos, using machine learning and computer vision techniques. The goal is to democratize the use of AI-based systems by integrating object detection, tracking, and feature extraction into a functional and user-friendly interface. Despite significant advances in this area, adoption remains limited due to technical complexity and the lack of public, well-documented tools. The proposed solution identifies individuals in both live video streams and uploaded recordings, supports unique identifiers, and includes data privacy measures such as encryption. The system bridges the gap between research libraries and practical applications, offering a modular platform suitable for urban security, automation, access control, and social projects. Built with established libraries such as YOLO, OSnet, and web frameworks like FastAPI, this end-to-end pipeline emphasizes performance and accessibility for developers and researchers.

Keywords: artificial intelligence; deep learning; people tracking; re-identification; real-time video.

LISTA DE ABREVIATURAS E SIGLAS

Siglas

API	Interface de Programação de Aplicações (API, do inglês *Application Programming Interface*)
DDD	Design Direcionado Domínio (DDD do inglês *Domain Driven Design*)
GB	Gigabyte (GB)
GPU	Unidade de Processamento Gráfico (GPU do inglês *Graphic Process Unit*)
HTTP	Protocolo de Transferência de Hipertexto (HTTP, do inglês Hypertext Transfer Protocol)
IA	Inteligência Artificial
JSON	Notação de Objetos JavaScript (JSON, do inglês JavaScript Object Notation)
RAM	Memória de Acesso Aleatório (RAM, do inglês Random Access Memory)
RBAC	Controle de Acesso Baseado em Papéis (RBAC do inglês *Role-Based Access Control*)
RNC	Redes neurais convolucionais, do inglês *Convolutional Neural Network*)
ROI	Área de interesse (ROI do inglês *Region of Interest*)
TCP	Contexto de Protocolo de Texto (TPC do inglês *Text Context Protocol*)
TI	*Tecnologia da Informação*
XSS	*Cross-site scripting*)

SUMÁRIO

1 INTRODUÇÃO

Com o avanço da era da informação, a sociedade tem se adaptado de forma contínua às inovações tecnológicas. Essas, por sua vez, permeiam desde aspectos cotidianos, como a substituição de linhas telefônicas por dispositivos móveis, até aplicações complexas de Inteligência Artificial voltadas à automação e otimização de processos em escala.

Assistentes virtuais, *chatbots* e sistemas inteligentes tornaram-se cada vez mais comuns no cotidiano da população, contribuindo para automatizar tarefas, reduzir a necessidade de intervenção humana e minimizar falhas operacionais. Estes sistemas são baseados em arquiteturas de redes neurais artificiais inspiradas no funcionamento do cérebro humano, compostas por múltiplas camadas ocultas que simulam sinapses neurais e realizam cálculos matemáticos em paralelo para transformar entradas (como imagens ou textos) em saídas interpretáveis (GOODFELLOW; BENGIO; COURVILLE, 2016).

Entre os maiores desafios da Inteligência Artificial (IA), está o campo da visão computacional, mais especificamente o reconhecimento e o rastreamento de objetos em vídeos. Essa subárea da IA busca capacitar máquinas a interpretarem o conteúdo visual de imagens e vídeos, com o objetivo de detectar e acompanhar, ao longo do tempo, objetos ou indivíduos em movimento. Suas aplicações abrangem áreas como segurança pública, monitoramento urbano, controle de acesso (WEI *et al.*, 2020), varejo e marketing comportamental (JUNIOR; MARTINI, 2019), sistemas de transporte inteligentes e saúde pública (SILVA, 2022), por exemplo, para análise de fluxo em hospitais ou acompanhamento de pacientes com distúrbios de mobilidade (YADAV *et al.*, 2022).

Modelos modernos permitem, além da detecção de indivíduos, seu reconhecimento com base em características visuais, mesmo diante de ambientes com múltiplas câmeras, iluminação variável ou alta densidade populacional. Técnicas como detecção de objetos (por exemplo, o algoritmo *YOLO — You Only Look Once*, que divide imagens em grades e identifica objetos em tempo real) (REDMON *et al.*, 2016), rastreamento multi-objetos (como o DeepSORT, que combina informações espaciais com *features* visuais para acompanhar indivíduos ao longo de sequências (WOJKE; BEWLEY; PAULUS, 2017)) e reidentificação de pessoas, que utiliza vetores numéricos chamados *embeddings* para reconhecer a mesma pessoa em diferentes contextos (como o modelo OSNet em (ZHOU *et al.*, 2019)), são frequentemente utilizadas em conjunto para garantir que uma mesma pessoa seja reconhecida ao longo de diferentes momentos e cenas (MCLAUGHLIN; RINCON; MILLER, 2016).

Apesar do avanço dessas soluções, sua adoção ainda é limitada pela complexidade técnica envolvida em sua implementação. Muitos dos *frameworks* existentes exigem conhecimentos especializados em aprendizado profundo, engenharia de software e manipulação de vídeo em tempo real, o que restringe sua aplicação a empresas com equipes altamente capacitadas. Além disso, o processo de treinamento de modelos próprios demanda poder compu-

tacional significativo e acesso a grandes bases de dados anotadas, o que inviabiliza o uso por desenvolvedores independentes ou instituições de pequeno porte.

O sistema proposto nesse Trabalho de Conclusão de Curso utilizará modelos previamente treinados, com o intuito de facilitar sua aplicação em contextos diversos, como segurança urbana, empresas e ambientes domésticos. A proposta visa democratizar o acesso a essa tecnologia, unindo conceitos de processamento de vídeo, aprendizado de máquina e desenvolvimento *web* em uma solução funcional e intuitiva.

Adicionalmente, nota-se uma escassez de API públicas que ofereçam suporte eficiente para reconhecimento de pessoas por imagem ou vídeo. Quando existentes, essas ferramentas costumam ser mal documentadas, desatualizadas ou de difícil integração com sistemas modernos. Este projeto busca preencher essa lacuna, utilizando modelos pré-treinados e *gsplframework* de código aberto, como YOLOv11 (REDMON *et al.*, 2016), OSNet (ZHOU *et al.*, 2019) e FastAPI (RAMíREZ, 2023), para simplificar a adoção da tecnologia mesmo por usuários sem formação em IA.

Cabe destacar que, por lidar com dados sensíveis como imagens de pessoas, questões como desempenho, acurácia e segurança são cruciais. A aplicação proposta incluirá criptografia de identificadores (por meio de *Advanced Encryption Standard* a fim de proteger a privacidade dos usuários, manter a estrutura de dados e mitigar riscos de ataques ou uso indevido das informações. A interface será concebida de forma clara e segura, promovendo facilidade de uso sem comprometer a proteção dos dados.

Ao final, espera-se obter um protótipo funcional de API que permita rastrear indivíduos em vídeos com o uso de IA, com possibilidade de extensão futura para funcionalidades como alertas em tempo real, reconhecimento corporal, múltiplos alvos e *dashboards* analíticos.

1.1 Objetivos

1.1.1 Objetivo geral

Desenvolver uma API funcional para rastreamento de pessoas em vídeos, com suporte tanto para transmissões em tempo real quanto para *uploads*, utilizando técnicas de aprendizado de máquina e visão computacional.

1.1.2 Objetivos específicos

- Investigar e selecionar bases de dados e um modelo de detecção, rastreamento e re-identificação de pessoas que ofereça bom desempenho em tempo real;
- Integrar o modelo selecionado em uma aplicação acessível via API;

- Permitir a inserção de identificadores personalizados para reconhecimento individual;
- Criar uma interface mínima para interação com o sistema, com foco em usabilidade e segurança;
- Avaliar o desempenho do sistema em diferentes contextos (qualidade de vídeo, iluminação, número de indivíduos, métricas estatísticas);
- Implementar mecanismos de criptografia para garantir a privacidade dos dados dos usuários;
- Documentar a API de forma clara, visando sua reutilização por outros pesquisadores e desenvolvedores.

1.2 Justificativa

O tema proposto é de relevância no cenário contemporâneo, especialmente no contexto da crescente urbanização, da digitalização de serviços e do aumento da demanda por soluções tecnológicas voltadas à segurança, automação e gestão inteligente de espaços. O uso de sistemas de rastreamento e reconhecimento de pessoas por vídeo tem potencial para impactar diretamente áreas críticas como segurança pública, cidades inteligentes, controle de acesso, logística, varejo, saúde e ambientes corporativos (MELO; SERRA, 2022).

Em meio a esse panorama, destaca-se a dificuldade de acesso a ferramentas que possibilitem a implementação de tais sistemas de forma prática e acessível. As soluções atualmente disponíveis para rastreamento e identificação de pessoas por vídeo são, em sua maioria, restritas a grandes empresas ou instituições com equipes altamente especializadas, uma vez que exigem domínio técnico em aprendizado profundo, Redes neurais convolucionais, do inglês *Convolutional Neural Network*) (RNC), visão computacional e manipulação de fluxos de vídeo em tempo real. Essa barreira tecnológica restringe a adoção mais ampla da IA, impedindo que pesquisadores, pequenas empresas e até mesmo desenvolvedores experientes — mas não especializados em IA — consigam criar suas próprias soluções personalizadas (ALMASAWA; ELREFAEI; MORIA, 2019).

Além disso, falta no mercado um ecossistema sólido de APIs públicas bem documentadas que permitam integração rápida com sistemas já existentes, o que limita o uso prático de IAs em vídeo em contextos fora do meio corporativo. Este projeto responde diretamente a essa lacuna, propondo uma API robusta, adaptável e fácil de usar, que permita a qualquer pessoa — com ou sem conhecimento profundo em IA — implementar sistemas de rastreamento com segurança e eficiência.

Ao desenvolver uma API acessível, customizável e de fácil integração, este trabalho visa democratizar o acesso à tecnologia de rastreamento por vídeo baseada em IA. A proposta é permitir que diferentes perfis de usuários — incluindo pesquisadores, desenvolvedores *web* e

profissionais de *Tecnologia da Informação* (TI) — possam aplicar essas técnicas sem a necessidade de treinar modelos do zero ou compreender toda a complexidade dos algoritmos subjacentes. O projeto também se destaca por abordar um ponto sensível no desenvolvimento de sistemas de monitoramento: a proteção da privacidade e a segurança da informação. Por isso, o sistema proposto incorpora práticas como criptografia de *embeddings* e uma interface clara e funcional, com foco em confiabilidade e proteção dos dados sensíveis dos usuários.

Adicionalmente, o projeto está alinhado aos princípios de viabilidade técnica e aplicabilidade prática. Existem bibliotecas e *frameworks* consolidados no ecossistema de código aberto — como OpenCV, YOLO, OSNet e Flask/FastAPI — que podem ser aproveitados para estruturar uma solução funcional dentro do tempo e dos recursos disponíveis no desenvolvimento de um trabalho de conclusão de curso.

Por fim, o projeto é também motivado por um interesse pessoal da autora nas áreas de aprendizado de máquina e desenvolvimento *web*. Trata-se, portanto, de uma oportunidade de aplicar conhecimentos adquiridos ao longo da formação acadêmica em um desafio real, que une teoria e prática e contribui tanto para o avanço pessoal quanto para o debate e a produção de conhecimento dentro da área de computação aplicada.

2 APRENDIZADO DE MÁQUINA

A Inteligência Artificial é um campo da ciência da computação que busca desenvolver sistemas capazes de executar tarefas que normalmente exigiriam inteligência humana, como percepção visual, reconhecimento de padrões, tomada de decisões e processamento de linguagem natural. Entre os principais subcampos da IA, destaca-se o aprendizado de máquina, que permite que os sistemas aprendam padrões e tomem decisões com base em dados, sem serem explicitamente programados para cada tarefa específica (GOODFELLOW; BENGIO; COURVILLE, 2016).

Nesse contexto, uma subárea da Inteligência Artificial é o aprendizado de máquina aplicado a problemas de visão computacional. Este, por sua vez, possui uma vertente ainda mais especializada, o aprendizado profundo, descrito com mais detalhes na Seção 2.1, que se baseia em arquiteturas conhecidas como RNC. Essas redes são compostas por múltiplas camadas interconectadas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para isso, as redes neurais convolucionais são as arquiteturas mais utilizadas. Elas foram projetadas para processar dados que possuem uma estrutura de matriz, como imagens, e são compostas por filtros capazes de capturar padrões locais, como bordas, texturas e formas (GOODFELLOW; BENGIO; COURVILLE, 2016). À medida que a informação percorre as camadas da rede, essas representações locais são combinadas em padrões mais complexos, permitindo que o modelo reconheça objetos inteiros ou identifique características específicas de uma pessoa.

Ao final da construção de uma rede neural convolucional, tem-se como resultado um conjunto de números que representam características extraídas das imagens. A qualidade dessa representação é diretamente dependente do modelo escolhido, da diversidade do conjunto de dados utilizado no treinamento e das técnicas aplicadas, como *Deep Metric Learning*, *Local Feature Learning* e o uso de GANs para aumento de dados.

2.1 Aprendizado Profundo

O *Deep Learning*, ou aprendizado profundo, é uma subárea do aprendizado de máquina que tem como objetivo construir modelos computacionais capazes de aprender representações hierárquicas de dados, utilizando arquiteturas compostas por múltiplas camadas de processamento não linear, conhecidas como redes neurais profundas (GOODFELLOW; BENGIO; COURVILLE, 2016).

O termo profundo deve-se à presença de várias camadas sucessivas de transformação dos dados, onde cada camada aprende uma representação progressivamente mais abstrata de informação.

Historicamente, o conceito de redes neurais existe desde a década de 1950, com o modelo de *perceptron* desenvolvido por Frank Rosenblatt. Porém, as primeiras arquiteturas de re-

des neurais convolucionais apresentavam várias limitações práticas, como, por exemplo, a falta de recursos computacionais precedentes à época (GOODFELLOW; BENGIO; COURVILLE, 2016). Foi apenas a partir de 2006, com o desenvolvimento de técnicas como o pré-treinamento não supervisionado e, posteriormente, com a popularização de Unidade de Processamento Gráfico (GPU do inglês *Graphic Process Unit*) (GPU) e de algoritmos como a retropropagação, que o aprendizado profundo passou a alcançar resultados expressivos em problemas reais.

O aprendizado profundo se tornou uma abordagem central em diversos ramos da Inteligência Artificial, especialmente em visão computacional, processamento de linguagem natural, reconhecimento de fala e jogos, devido à sua capacidade de aprender diretamente dos dados, dispensando engenharia manual de *features*, aplicando suas próprias regras para medir o peso dessas *features*.

Arquitetura de uma Rede Neural Profunda

Uma rede neural profunda é composta por um conjunto de unidades computacionais chamadas de neurônios artificiais ou nós, organizados em múltiplas camadas. Essas camadas podem ser divididas em três tipos principais: camada de entrada (*input*), camadas ocultas (*hidden layers*) e camada de saída (*output*).

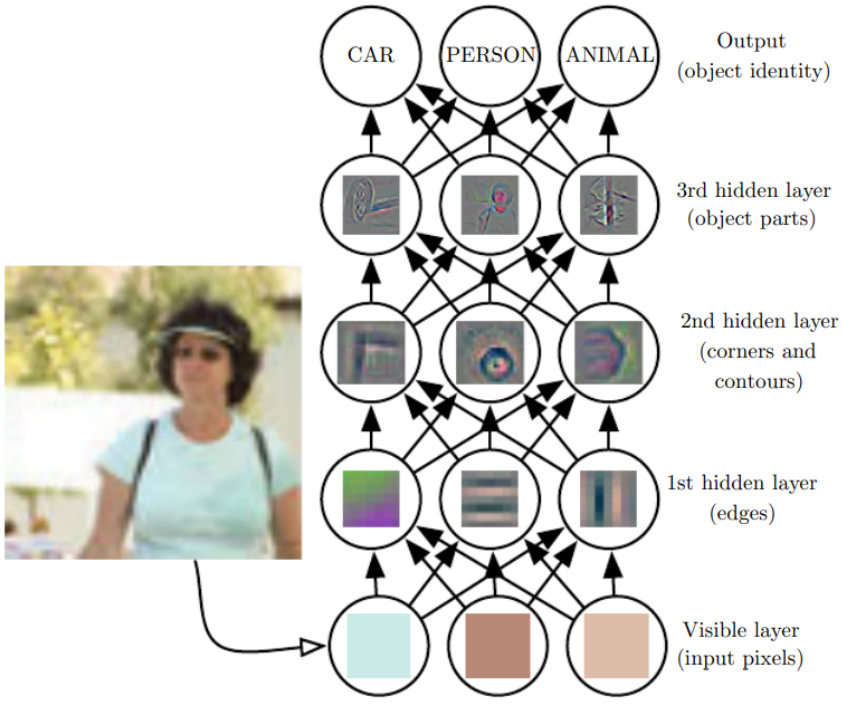

Figura 1 – Exemplo de arquitetura de rede neural profunda com múltiplas camadas ocultas.
Fonte: *Deep Learning* (GOODFELLOW; BENGIO; COURVILLE, 2016).

A Figura 1 ilustra uma arquitetura típica de rede neural profunda com múltiplas camadas ocultas. Cada neurônio artificial recebe um conjunto de entradas durante o treinamento, as quais são multiplicadas por pesos associados a cada conexão. Em seguida, essas entradas

ponderadas são somadas a um viés e passam por uma função de ativação, como a ReLU (*Rectified Linear Unit*), Sigmoid ou Tanh, que introduz não-linearidade ao sistema (GOODFELLOW; BENGIO; COURVILLE, 2016). Essa característica permite que a rede seja capaz de modelar relações complexas entre os dados.

Parâmetros e Hiperparâmetros

O desempenho e o comportamento de uma rede neural profunda durante o treinamento e a inferência dependem de diversos parâmetros e hiperparâmetros que influenciam desde o ajuste dos pesos até a estabilidade do aprendizado. A seguir, são descritos os principais hiperparâmetros utilizados no desenvolvimento de modelos de aprendizado profundo.

- **Número de épocas:** Quantidade de vezes que o modelo percorrerá o conjunto de treinamento completo. Durante cada época, o modelo realiza sucessivas atualizações dos seus parâmetros (pesos e viés), com o objetivo de reduzir o erro em relação aos valores esperados. Normalmente, o treinamento de redes profundas requer múltiplas épocas para que o modelo consiga generalizar bem os padrões aprendidos (GOODFELLOW; BENGIO; COURVILLE, 2016);

- **Viés:** Parâmetro adicional presente em cada neurônio de uma rede neural, que tem como função ajustar a saída do neurônio mesmo quando todas as entradas são zero. O viés também é ajustado na função de gradiente descendente, e é comumente inicializado com valores pequenos e aleatórios (GOODFELLOW; BENGIO; COURVILLE, 2016);

- **Taxa de aprendizado:** A taxa de aprendizado é um hiperparâmetro fundamental que controla a magnitude da atualização dos pesos da rede a cada iteração do processo de treinamento. Ela determina o quão rápido ou devagar o modelo ajusta os seus parâmetros em resposta ao gradiente da função de perda. Uma taxa de aprendizado muito alta pode fazer com que o modelo oscile ou até mesmo divirja, nunca alcançando um mínimo da função de perda. Por outro lado, uma taxa muito baixa pode tornar o treinamento extremamente lento, além de aumentar o risco de o modelo ficar preso em mínimos locais (GOODFELLOW; BENGIO; COURVILLE, 2016);

- ***Batch size*:** O *batch size* refere-se ao número de amostras de dados utilizadas para calcular o gradiente e atualizar os pesos da rede a cada iteração de treinamento. Em vez de atualizar os pesos após cada amostra individual (*stochastic gradient descent*) ou apenas ao final de uma época (*batch gradient descent*), utiliza-se o *mini-batch gradient descent*, que realiza as atualizações após processar pequenos lotes de amostras. Essa abordagem equilibra o custo computacional e a estabilidade do gradiente, sendo uma prática comum o uso de tamanhos de lote entre 16, 32, 64 ou 128, dependendo

dos recursos computacionais disponíveis e do tamanho do *dataset* (GOODFELLOW; BENGIO; COURVILLE, 2016);

- **Função de perda:** A função de perda é responsável por quantificar a discrepância entre a saída prevista pelo modelo e os valores reais. Durante o treinamento, o objetivo da rede neural é minimizar essa discrepância. Existem diferentes funções de perda, e a escolha depende do tipo de tarefa. Para problemas de classificação, por exemplo, é comum utilizar a *Cross-Entropy Loss*, enquanto para regressão, funções como *Mean Squared Error* (MSE) são mais adequadas. A função de perda orienta o cálculo dos gradientes durante o processo de retropropagação, influenciando diretamente na atualização dos parâmetros do modelo (GOODFELLOW; BENGIO; COURVILLE, 2016);

- **Otimização:** O algoritmo de otimização é o método utilizado para minimizar a função de perda ajustando os pesos e o viés da rede de forma eficiente. O mais tradicional é o *Gradient Descent*, mas versões mais avançadas são amplamente utilizadas devido à sua capacidade de acelerar a convergência e lidar melhor com problemas como platôs de gradiente e ravinas. Entre os otimizadores mais populares destacam-se:

  - **SGD *Stochastic Gradient Descent***: Realiza atualizações com base em lotes pequenos, sendo simples, mas eficiente;
  - **Adam (*Adaptive Moment Estimation*)**: Ajusta automaticamente a taxa de aprendizado de cada parâmetro com base em momentos do gradiente, proporcionando uma convergência mais rápida e estável;
  - **RMSprop**: Uma variação que normaliza os gradientes pelo quadrado da média das magnitudes recentes, sendo eficaz em problemas com gradientes esparsos (GOODFELLOW; BENGIO; COURVILLE, 2016).

  A escolha do otimizador influencia diretamente a eficiência do treinamento e a qualidade da solução final.

- **Função de ativação:** A função de ativação é responsável por introduzir não-linearidades no modelo, permitindo que a rede neural seja capaz de aprender relações complexas entre as entradas e as saídas. Sem uma função de ativação não-linear, a rede seria apenas uma combinação linear de suas entradas, independentemente da quantidade de camadas, impedindo o aprendizado de padrões complexos, como para reconhecimento de imagens.

  As funções de ativação mais comuns incluem:

  - **ReLU (*Rectified Linear Unit*):** Define a saída como zero para valores negativos e linear para positivos, sendo a mais utilizada em redes profundas devido à sua simplicidade e eficácia em evitar o problema do gradiente desvanecente;

- **Sigmoid:** Comprime os valores de saída para um intervalo entre 0 e 1, muito usada em problemas de classificação binária, mas com risco de saturação para entradas muito altas ou muito baixas;
- **Tanh (Tangente Hiperbólica)**: Similar à sigmoid, mas com saída entre -1 e 1, o que pode proporcionar melhor centralização dos dados em algumas situações (GOODFELLOW; BENGIO; COURVILLE, 2016).

Esses conceitos serão explorados em detalhes na Seção 4, onde serão definidos os valores específicos utilizados em cada etapa do desenvolvimento da solução proposta.

Processo de Treinamento

O treinamento de uma rede neural profunda consiste em dois ciclos principais: a passagem direta e a retropropagação.

Durante a passagem direta, os dados de entrada são processados camada por camada, da entrada até a saída da rede. Cada neurônio artificial realiza uma combinação linear das entradas recebidas, ponderadas pelos seus respectivos pesos, e adiciona um termo de viés. O resultado dessa combinação é então passado por uma função de ativação, como a ReLU ou a Sigmoid, que introduz não-linearidade ao modelo, permitindo que a rede aprenda representações complexas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Matematicamente, o funcionamento de cada camada pode ser descrito pela Equação 1:

$$a^{(l)} = f\left(W^{(l)} \cdot a^{(l-1)} + b^{(l)}\right) \tag{1}$$

Onde:

- $a^{(l)}$ representa a saída da camada l;
- $W^{(l)}$ são os pesos da camada l;
- $b^{(l)}$ é o vetor de vieses;
- f é a função de ativação escolhida para a camada.

Ao final da última camada, a rede produz uma predição ou saída $(\hat{y})$, que é comparada ao valor real (y) por meio de uma função de perda, como *Cross-Entropy Loss* para tarefas de classificação ou *Mean Squared Error* para regressão. Essa função calcula o erro entre a predição e o valor esperado.

Em seguida, inicia-se a retropropagação, em que o erro calculado é propagado de volta através das camadas da rede. O algoritmo de retropropagação, proposto originalmente por Rumelhart, Hinton e Williams (1986), calcula os gradientes parciais da função de perda em relação

aos pesos de cada camada, utilizando o método de diferenciação conhecido como regra da cadeia do cálculo diferencial (GOODFELLOW; BENGIO; COURVILLE, 2016).

Esses gradientes são então utilizados por um algoritmo de otimização, como o SGD, Adam ou RMSprop, para ajustar os pesos da rede, reduzindo gradualmente o erro nas próximas iterações (BERLYAND; JABIN, 2023).

O processo de treinamento repete a passagem direta e a retropropagação para múltiplos *mini-batches* até completar uma época, que corresponde a uma varredura completa por todo o conjunto de treinamento.

O ciclo completo é repetido ao longo de múltiplas épocas, enquanto o desempenho da rede é monitorado por meio de métricas como acurácia, precisão, *recall* e perda de validação. Durante o treinamento, hiperparâmetros como a taxa de aprendizado, o *batch size*, o número de épocas e o tipo de otimizador têm influência direta na velocidade de convergência e na qualidade do modelo final.

Durante o processo de treinamento, técnicas de regularização também são essenciais para evitar o fenômeno conhecido como sobreajuste, no qual a rede aprende padrões muito específicos dos dados de treinamento e apresenta baixo desempenho em dados não vistos anteriormente.

Uma das estratégias mais utilizadas para mitigar esse problema é o *Dropout*, proposto por Srivastava *et al.* (2014). O *Dropout* consiste em desativar aleatoriamente uma porcentagem dos neurônios durante cada iteração de treinamento, forçando a rede a não depender excessivamente de nenhum caminho específico para a propagação de informação. Isso incentiva a formação de representações mais robustas e generalizáveis. O valor típico para o parâmetro de *Dropout* varia entre 0,2 e 0,5, significando que entre 20% e 50% dos neurônios podem ser temporariamente desativados por iteração (SRIVASTAVA *et al.*, 2014).

Além do *Dropout*, outra prática comum durante o treinamento de redes neurais profundas é o uso de *Data Augmentation*. Esta técnica visa aumentar a diversidade dos dados de entrada por meio de transformações artificiais aplicadas ao conjunto de dados original. Entre as formas mais comuns de *Data Augmentation* em visão computacional estão: rotações, inversões horizontais, variações de brilho, recortes aleatórios e alterações de escala (GOODFELLOW; BENGIO; COURVILLE, 2016). O objetivo é tornar o modelo mais robusto a variações que podem ocorrer nas condições reais de uso, como mudanças de iluminação, perspectiva ou oclusões parciais.

Combinadas, as estratégias de otimização, regularização (como *Dropout*) e *Data Augmentation* contribuem para o desenvolvimento de um modelo mais preciso e com melhor capacidade de generalização ao lidar com novas amostras de dados.

2.2 Detecção de pessoas via aprendizado profundo

Um dos elementos necessários para alguns problemas de visão computacional é a geração automática de Área de interesse (ROI do inglês *Region of Interest*) (ROI)s, também conhecidas como *bounding boxes*. Essas *bounding boxes* são áreas específicas dentro de uma imagem ou vídeo onde se presume haver informação relevante — neste caso, a presença de uma pessoa. Para que sistemas automatizados funcionem corretamente, é necessário que consigam identificar, sem intervenção humana, onde essas pessoas estão no vídeo. Essa tarefa pode ser realizada por modelos de detecção, como por exemplo, o *You Only Look Once* (REDMON *et al.*, 2016), que segmentam os quadros do vídeo em tempo real, indicando com precisão onde os objetos — pessoas, neste caso — estão localizados.

Arquitetura YOLO para Detecção de Objetos

O algoritmo YOLO é uma das arquiteturas mais conhecidas e utilizadas na tarefa de detecção de objetos em tempo real. Proposto por Redmon *et al.* (2016), o YOLO surgiu com o objetivo de resolver limitações de modelos anteriores que, embora apresentassem alta acurácia, sofriam com elevado tempo de processamento, tornando-os inviáveis para aplicações com restrições de latência, como vídeos ao vivo.

A Figura 2 ilustra uma arquitetura da rede neural introduzida no YOLO.

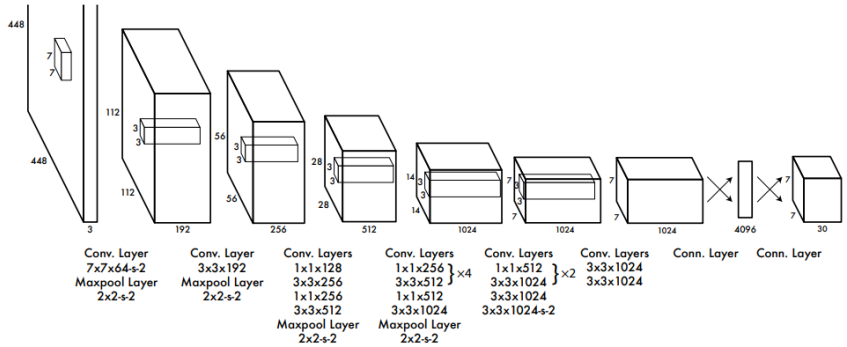

Figura 2 – Arquitetura da rede convolucional usada no modelo YOLO.
Fonte: You Only Look Once: Unified, Real-Time Object Detection (REDMON *et al.*, 2016)

Enquanto arquiteturas tradicionais de detecção de objetos operam por meio de um *pipeline* de múltiplas etapas — geralmente incluindo a geração de regiões propostas, extração de características para cada uma dessas regiões e posterior classificação — o YOLO adota uma abordagem de regressão direta, tratando a detecção como um único problema de mapeamento de entrada para saída (REDMON *et al.*, 2016).

O funcionamento do YOLO pode ser descrito da seguinte forma: a imagem de entrada é dividida em uma grade de células de tamanho fixo. Para cada célula da grade, o modelo prevê simultaneamente:

- As coordenadas das *bounding boxes* — incluindo posição (x, y) e dimensões (largura e altura);
- A classe (ou tipo) do objeto contido naquela caixa (ex.: pessoa, carro, bicicleta);
- A confiança da detecção — um valor numérico que expressa o grau de certeza da rede de que a caixa contém um objeto e que esse objeto pertence à classe prevista.

Essa estrutura permite que o YOLO realize a detecção de todos os objetos de uma imagem em uma única execução da rede neural, sem a necessidade de percorrer múltiplas regiões de forma sequencial. Em termos práticos, isso resulta em uma redução significativa do tempo de processamento, permitindo a detecção em tempo real, mesmo em vídeos com altas taxas de quadros por segundo.

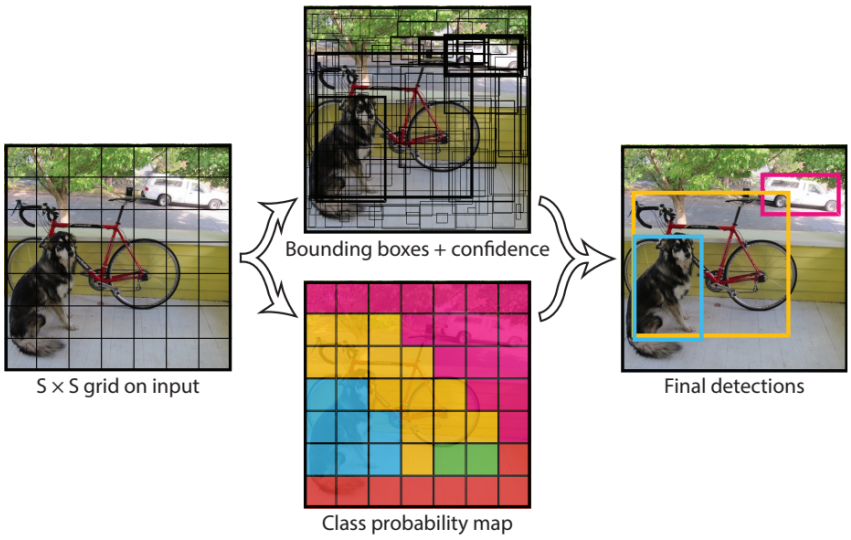

Figura 3 – Formação de *Bounding Boxes* e classificação de imagens com YOLO.
Fonte: You Only Look Once: Unified, Real-Time Object Detection (REDMON *et al.*, 2016)

A saída final do YOLO consiste em uma série de caixas delimitadoras com suas respectivas classes e níveis de confiança, permitindo ao sistema identificar múltiplos objetos em diferentes posições de uma única imagem ou quadro de vídeo. Essa característica faz com que o YOLO seja uma das principais escolhas em aplicações que exigem rapidez, como vigilância por vídeo, direção autônoma e análise de fluxo de pessoas em tempo real (REDMON *et al.*, 2016).

Desde a sua primeira versão, o YOLO passou por diversas atualizações significativas. Versões como o YOLOv5, YOLOv8 e YOLOv11 apresentam melhorias tanto em desempenho quanto em acurácia, incorporando avanços como camadas de atenção, arquiteturas mais profundas e técnicas modernas de otimização (REDMON *et al.*, 2016).

Enquanto o YOLO é amplamente utilizado para detecção de objetos em tempo real, incluindo pessoas, ele não é projetado para reidentificação. Então, para reidentificar indivíduos, é necessário extrair *embeddings* que capturem características únicas além da simples detecção.

Portanto, após a detecção inicial com o YOLO, é comum empregar modelos especializados em reidentificação para extrair *embeddings* e realizar a correspondência entre indivíduos ao longo do tempo e em diferentes câmeras.

2.3 Reidentificação de Pessoas (*Person Re-Identification*)

A Reidentificação de Pessoas é uma tarefa dentro da visão computacional que consiste em reconhecer o mesmo indivíduo em diferentes cenas, câmeras ou momentos, mesmo que a pessoa esteja com variações de ângulo, iluminação ou postura (ALMASAWA; ELREFAEI; MORIA, 2019).

Após a detecção, é necessário o processamento com Inteligência Artificial para a extração de características únicas dessas pessoas detectadas. Essas características, chamadas de *features* ou *embeddings*, são vetores numéricos que codificam informações visuais relevantes de cada indivíduo — como tipo de roupa, cor, estrutura corporal e outros padrões que, juntos, permitem distinguir uma pessoa de outra. Esses vetores são fundamentais para reconhecer o mesmo indivíduo em câmeras diferentes, em momentos distintos, mesmo que ele não esteja sempre na mesma posição ou iluminação. Diversas abordagens têm sido desenvolvidas para gerar *embeddings* eficazes para tal objetivo, e dentro elas, incluem-se:

- Aprendizado Métrico Profundo:
  - Utiliza redes neurais treinadas com funções de perda específicas, como a *triplet loss*, para mapear imagens de pessoas em um espaço vetorial onde indivíduos semelhantes estão próximos e diferentes estão distantes;
  - É eficaz na captura de relações de similaridades e diferenças entre indivíduos (LI *et al.*, 2023).
- Aprendizado de Características Locais:
  - Foca na extração de partes específicas do corpo (por exemplo, cabeça, ombro, joelho e pé) para gerar *embeddings* mais robustos e variações de pose e oclusões;
  - Modelos como o PCB (*Part-based Convolutional Baseline*) dividem a imagem em segmentos horizontais e extraem características de cada parte (WU *et al.*, 2024).
- Redes Adversárias Generativas (GANs):
  - Treina duas redes neurais generativas para competirem entre si e gerar novos dados mais autênticos a partir de um determinado conjunto de dados de treinamento;

  - Utilizadas para gerar imagens sintéticas ou transformar imagens existentes, ajudando a modelar variações de aparência e melhorar a generalização dos *embeddings* (Amazon Web Services, 2023).

- Aprendizado de Características Sequenciais:

  - Explora informações temporais em sequências de vídeo, utiliza redes recorrentes ou mecanismos de atenção para capturar padrões dinâmicos de movimento e comportamento;
  - Essa abordagem é valiosa para reidentificação em vídeos, onde o movimento pode fornecer pistas adicionais (AL-JABERY *et al.*, 2020).

O processo de Re-ID geralmente é composto por três etapas principais:

1. **Extração de *Embeddings*:** Após a detecção da pessoa, a *bounding box* é processada por uma rede neural convolucional especializada, como a OSNet, para extrair um *embedding* que representa as características visuais únicas daquele indivíduo (ZHOU *et al.*, 2019);

2. **Comparação Vetorial:** Através de uma métrica de distância (como a Euclidiana ou *Cosine Distance*), o sistema compara o *embedding* da pessoa detectada com o *embedding* de referência fornecido pelo usuário. *Embeddings* com menor distância são considerados mais similares, permitindo a identificação da pessoa alvo (ALMASAWA; ELREFAEI; MORIA, 2019);

3. **Decisão de Identidade:** Caso a distância calculada esteja abaixo de um limiar previamente definido, o sistema considera que a detecção corresponde ao indivíduo de interesse.

Arquitetura OSNet para Re-ID

O OSNet é uma arquitetura de rede neural convolucional especializada na extração de *embeddings* de imagens de pessoas. Seu diferencial está na capacidade de capturar informações em múltiplas escalas espaciais de forma simultânea, por meio de blocos chamados *Omni-Scale Blocks*, que integram diferentes campos receptivos de convolução. Isso permite ao modelo ser eficiente tanto na captura de detalhes locais (como textura de roupa) quanto de contextos globais (como estrutura corporal) e temporais, no caso de capturas por conjuntos de *frames* em vídeos (ZHOU *et al.*, 2019). A Figura 4 ilustra a arquitetura típica do OSNet.

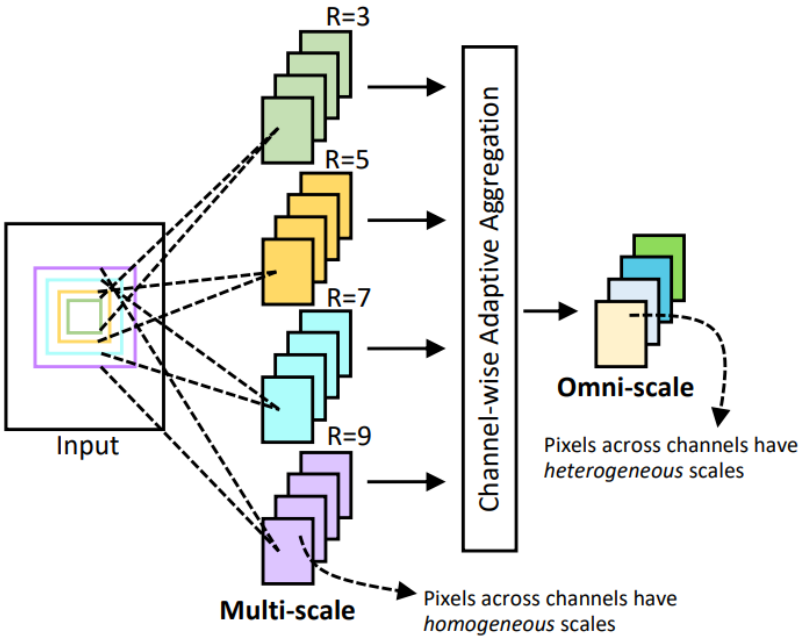

Figura 4 – Arquitetura convolucional usada no modelo Omni Scale.
Fonte: Omni-Scale Feature Learning for Person Re-Identification (ZHOU *et al.*, 2019)

2.4 Estudos Relacionados

Este trabalho se insere dentre múltiplas linhas de pesquisa consolidadas na área de ciência da computação. Está diretamente relacionado à área de reconhecimento facial e reidentificação de pessoas, que estuda formas de identificar indivíduos por meio de características visuais, com ou sem uso de biometria direta. Assim também com o campo da vigilância inteligente, onde algoritmos analisam vídeos em tempo real para tomar decisões ou alertar operadores sobre eventos de interesse.

Um exemplo significativo é o trabalho de Chen *et al.* (2018), que aplicou *Deep Metric Learning* com a função de perda *triplet loss* para gerar *embeddings* discriminativos em tarefas de reidentificação. Os autores demonstraram que esse método pode atingir alta acurácia ao separar vetorialmente indivíduos distintos com eficácia, mesmo em grandes conjuntos de dados.

Já o modelo PCB, proposto por Sun *et al.* (2021), adota uma abordagem de aprendizado local, segmentando a imagem em partes horizontais e extraindo características de cada uma delas. Essa estratégia mostrou-se particularmente eficaz em cenários com variações de pose e oclusão parcial, frequentemente encontradas em ambientes reais.

No contexto de melhoria de generalização e aumento da variabilidade visual, Karmakar e Mishra (2021) introduziram o uso de GANs para gerar amostras sintéticas de pessoas. Isso permitiu treinar modelos mais robustos mesmo com conjuntos de dados limitados, ampliando a capacidade dos *embeddings* de generalizar para novos indivíduos.

Quanto à reidentificação em vídeos, McLaughlin, Rincon e Miller (2016) propuseram uma rede recorrente convolucional (RRC) capaz de capturar padrões temporais para melhorar

a identificação de indivíduos em sequências contínuas. O modelo demonstrou que o uso de informações temporais melhora significativamente o desempenho em comparação com abordagens baseadas apenas em imagens estáticas.

Além disso, há outras pesquisas que merecem ser mencionadas em rastreamento de multiobjetos, uma vertente da Inteligência Artificial que busca acompanhar simultaneamente várias entidades em movimento ao longo do tempo, como o estudo de Bergmann, Meinhardt e Leal-Taixe (2019) com o *Tracktor++*, que combina detecção com rastreamento baseado em *tracking by regression*. Outro elo importante está com a detecção de anomalias em vídeo, como ações suspeitas ou comportamentos fora do padrão, que utiliza muitos dos mesmos fundamentos técnicos aqui abordados.

Por fim, estes estudos podem também tangenciar áreas mais aplicadas como engenharia de *software*, especialmente no que diz respeito à criação de infraestrutura moderna de APIs e interfaces *web* funcionais, que são essenciais para tornar as descobertas da Inteligência Artificial utilizáveis no mundo real, fora do ambiente restrito de laboratórios e grupos de pesquisa.

Além das contribuições acadêmicas em reidentificação e rastreamento, este trabalho também tangencia discussões recentes em engenharia de *software* para sistemas baseados em IA, especialmente no que se refere à produção de APIs modernas. Ramachandran (2024) propõe um modelo de *API IA-first*, que integra o processamento vetorial e infraestrutura, otimizando o uso de modelos de aprendizado de máquina em produção.

3 DESENVOLVIMENTO DE UMA APLICAÇÃO *WEB*

Atualmente, um dos maiores desafios enfrentados por profissionais e pesquisadores que desejam utilizar aprendizado de máquina para rastreamento de pessoas em vídeo é a ausência de soluções completas e acessíveis que conduzam todo o processo de ponta a ponta (ALMASAWA; ELREFAEI; MORIA, 2019). Desta forma, falta no mercado e na comunidade científica uma ferramenta integrada que vá desde a captura do vídeo até a entrega dos resultados processados de forma compreensível e utilizável.

Um aspecto crítico é a entrega dos resultados em um sistema *web* funcional, que permita que esses dados processados sejam utilizados por outros sistemas ou por usuários humanos sem conhecimento técnico em IA. Isso envolve não apenas a construção de uma API pública, que permita que sistemas externos enviem vídeos e recebam respostas com as identificações, mas também a criação de uma interface gráfica amigável, acessível por navegador, onde se possa testar e visualizar o funcionamento da solução. É nesse ponto que se torna clara a importância da arquitetura *end-to-end* — ou seja, que conecte todos os estágios do processo de forma integrada e fluida.

Tornar tecnologias avançadas de rastreamento de pessoas por vídeo acessíveis e reutilizáveis possui grande impacto social e tecnológico. A segurança pública e privada, por exemplo, frequentemente depende da análise de horas de gravações de câmeras, o que consome tempo e recursos humanos. Ferramentas automatizadas podem agilizar esse processo, reduzindo falhas humanas e otimizando respostas em tempo real. Em ambientes corporativos ou comerciais, como shoppings, supermercados ou escolas, o monitoramento de fluxo de pessoas permite, além da segurança, a análise de comportamento de clientes, otimização de rotas e melhoria da experiência do usuário. Em prédios públicos ou privados, o controle de acesso por meio de reidentificação permite sistemas mais inteligentes e adaptativos, promovendo automação predial e redução de custos com pessoal.

Nesse capítulo, será explicado um pouco sobre o funcionamento por trás de uma aplicação *web* e sobre estruturas necessárias para a implementação e integração de uma (ou mais) Inteligência Artificial no contexto geral desse projeto.

3.1 API RESTful

Uma API RESTful (*Representational State Transfer*) é um estilo arquitetural para desenvolvimento de aplicações *web* que seguem os princípios do Protocolo de Transferência de Hipertexto (HTTP, do inglês Hypertext Transfer Protocol) (HTTP), ou seja, é enviada uma solicitação de um cliente (ex.: um navegador) para um servidor, pedindo uma ação ou informação. O termo foi inicialmente definido por Roy Fielding em sua tese de doutorado em Fielding (2000). APIs baseadas em REST permitem que diferentes sistemas se comuniquem através de requisições HTTP, utilizando métodos como GET, POST, PUT e DELETE.

Entre as principais características de uma API RESTful estão:

- **Comunicação *Stateless*:** Cada requisição do cliente ao servidor é independente e não requer armazenamento de contexto entre as requisições;
- **Uso de Recursos:** Cada recurso (como imagens, vídeos, dados de usuário) é identificado por uma URL única;
- **Formato Padrão de Dados:** A troca de informações geralmente é realizada em formatos como Notação de Objetos JavaScript (JSON, do inglês JavaScript Object Notation) (JSON).

3.2 *WebSocket*

O protocolo *WebSocket*, padronizado pela RFC 6455 (FETTE; MELNIKOV, 2011), é uma tecnologia que permite comunicação bidirecional, persistente e em tempo real entre cliente e servidor através de uma única conexão de Contexto de Protocolo de Texto (TPC do inglês *Text Context Protocol*) (TCP). Diferentemente das requisições HTTP tradicionais, nas quais cada mensagem necessita abrir e fechar uma conexão, o *WebSocket* mantém uma conexão aberta, permitindo que o servidor envie dados ao cliente assim que novos eventos ocorrem.

Essa característica faz com que o *WebSocket* seja especialmente adequado para aplicações que necessitam de baixa latência e comunicação em tempo real, como sistemas de transmissão de vídeo ao vivo ou monitoramento de dados contínuos.

3.3 Integração entre API RESTful e *WebSocket* no Sistema Proposto

A combinação de uma API RESTful e de um canal *WebSocket* proporciona uma arquitetura flexível e escalável para o sistema de rastreamento de pessoas. Enquanto a API RESTful permitirá operações pontuais, como o envio de imagens de referência ou vídeos armazenados, o *WebSocket* garantirá a comunicação contínua e em tempo real para análise de transmissões ao vivo.

Essa arquitetura permitirá que usuários de diferentes perfis — desde desenvolvedores até operadores de segurança — interajam com o sistema de forma eficiente, acessível e com baixo tempo de resposta.

4 MATERIAIS E MÉTODOS

Nesse capítulo serão apresentados os materiais e métodos utilizados para o desenvolvimento de uma ferramenta *end-to-end* de rastreamento de pessoas baseada em Inteligência Artificial, com o intuito de possibilitar a detecção e identificação de indivíduos em vídeos transmitidos em tempo real ou por meio de arquivos enviados. A solução combina técnicas modernas de visão computacional, como detecção de *bounding boxes*, extração de *embeddings* e comparação vetorial, de forma a reconhecer um indivíduo específico a partir de uma imagem de referência fornecida pelo usuário. Uma visão geral do desenvolvimento pode ser observada na Figura 5.

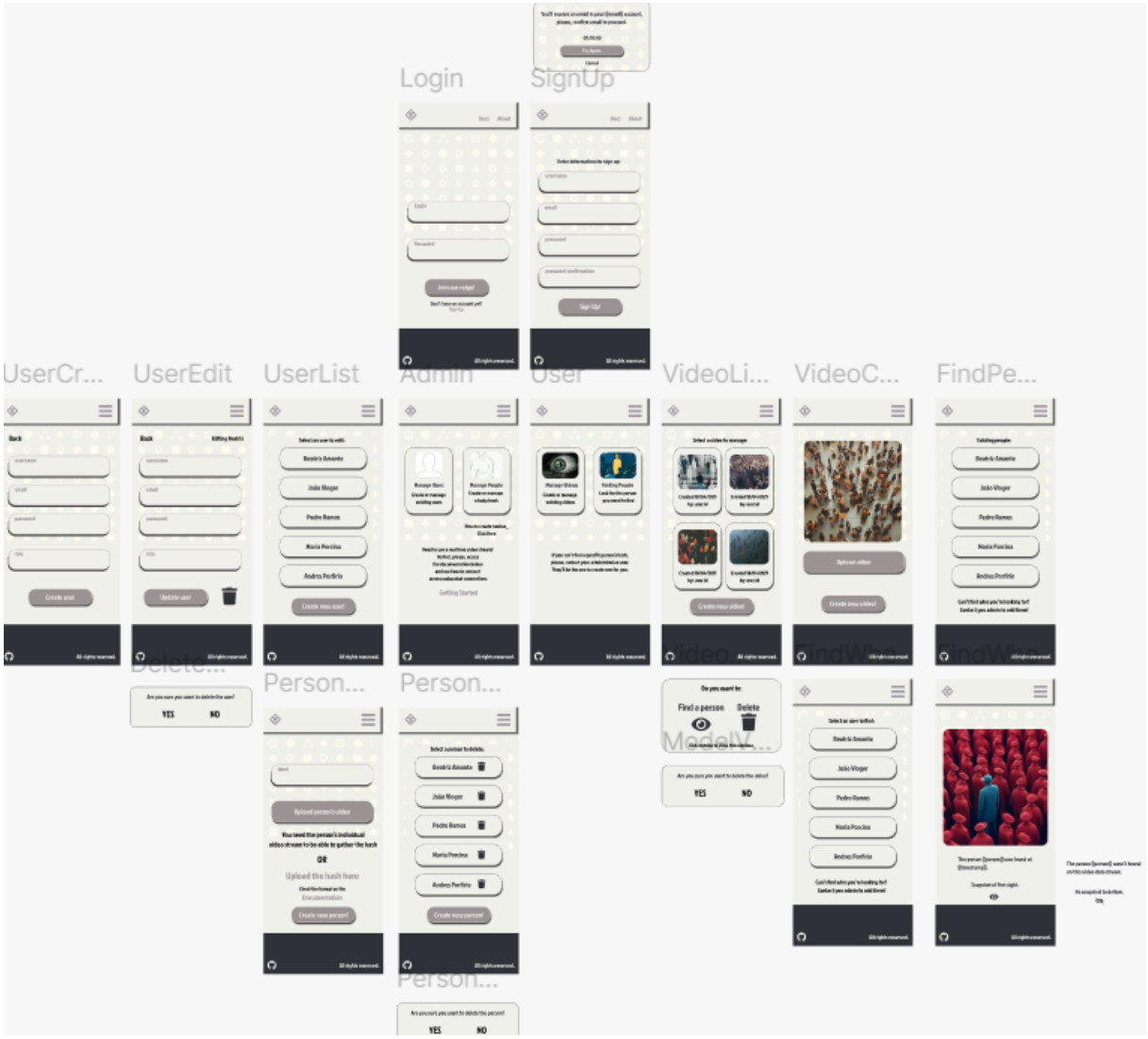

Figura 5 – Visão geral do sistema *Watch Me*.
Fonte: Autoria própria.

A ferramenta é estruturada como uma API acessível, com conexão via protocolo *websocket* e suporte a requisições RESTful, além de oferecer um *dashboard* interativo para visualização dos resultados. Por meio dessa interface, o usuário pode carregar uma imagem de uma

pessoa de interesse e conectar uma fonte de vídeo — seja por *upload* direto ou por *streaming*. A API é responsável por identificar se, onde e quando o indivíduo-alvo aparece nos vídeos processados, retornando as informações em formato JSON e visualizando os dados em tempo real por meio do painel *web*.

4.1 Materiais

O desenvolvimento da solução proposta neste Trabalho de Conclusão de Curso exige a utilização de um conjunto diversificado de ferramentas, *frameworks*, linguagens de programação e serviços de infraestrutura. A seguir, apresentam-se os principais materiais que foram empregados ao longo das etapas de desenvolvimento e implementação da API para rastreamento de pessoas com inteligência artificial.

- **Figma:** O Figma é uma ferramenta de prototipação e design de interfaces amplamente utilizada no desenvolvimento de aplicações *web* e *mobile*. Ele permite a criação de um *design system* estruturado, proporcionando melhor organização visual e facilitando a implementação da experiência do usuário (UX) e da interface do usuário (UI). O Figma é multiplataforma e pode ser acessado via navegador, além de possuir versões para Windows e MacOS (FIGMA, 2025);

- **DBDiagram:** Para a modelagem inicial do banco de dados relacional, foi utilizada a ferramenta online DBDiagram. Essa plataforma permite a criação rápida de diagramas entidade-relacionamento (ER) com sintaxe simplificada, possibilitando a exportação para SQL ou integração com outras ferramentas de desenvolvimento de banco de dados (DBDIAGRAM, 2025);

- **Git e GitHub:** O controle de versão do código foi realizado por meio do Git, utilizando o repositório remoto hospedado no GitHub. Essa combinação de ferramentas permitirá o versionamento eficiente do projeto, o trabalho colaborativo, e a implementação de fluxos de desenvolvimento como *branching* e *pull requests*, além de garantir o histórico de alterações e rastreabilidade das modificações (CHACON; STRAUB, 2014);

- **Notion:** Para o gerenciamento das *sprints* e organização das tarefas, foi utilizado o Notion. Essa ferramenta permite a criação de quadros Kanban, listas de tarefas e a documentação de decisões técnicas, promovendo a gestão ágil do desenvolvimento (NOTION, 2025);

- **Docker:** Para garantir a portabilidade e a reprodutibilidade do ambiente de desenvolvimento e produção, foi utilizada a tecnologia de *containers Docker*. O banco de dados será containerizado, possibilitando sua implantação em diferentes servidores ou máquinas virtuais de forma rápida e consistente (DOCKER, 2025);

- **Sphinx:** Para a documentação técnica da API e dos módulos de IA, foi utilizado o Sphinx, um gerador de documentação em formato estático, compatível com o padrão *reStructuredText*. O Sphinx permitirá a geração de documentos navegáveis, incluindo exemplos de uso, *endpoints* disponíveis e descrições detalhadas do funcionamento interno da aplicação (SPHINX, 2025);
- **PostgreSQL:** Armazenamento das informações de identificação, *embeddings* de referência, logs de processamento e dados de usuários da aplicação (POSTGRESQL, 2025);
- **OpenAPI (*Swagger*):** O *Swagger* permite que a estrutura de cada *endpoint* da API — incluindo parâmetros de entrada, formatos de resposta, códigos de status HTTP e exemplos de requisição — seja descrita de forma legível tanto por humanos quanto por máquinas. Ao adotar o *Swagger*, foi possível gerar automaticamente uma interface interativa via navegador, onde desenvolvedores podem testar os *endpoints* diretamente, enviar requisições e visualizar as respostas em tempo real, sem a necessidade de criar um cliente manual (OPENAPI, 2025).

Frontend

A interface visual da aplicação foi desenvolvida utilizando:

- **React Native**: Permite o desenvolvimento de aplicações móveis multiplataforma com código único (NATIVE, 2025);
- **Expo:** Simplifica o processo de construção, empacotamento e execução do aplicativo (EXPO, 2025);
- **Tailwind CSS:** Através do pacote *NativeWind*, que permite aplicar estilos utilitários de forma programática em *React Native* (NATIVEWIND, 2025).

4.1.1 Desenvolvimento da API RESTful para regra de negócio

Para o desenvolvimento da API RESTful, foram usados os seguintes *frameworks*, linguagens e ferramentas:

- ***TypeScript:*** Adiciona tipagem estática ao *JavaScript*, tornando o desenvolvimento mais seguro e reduzindo erros em tempo de compilação (TYPESCRIPT, 2025);
- ***Fastify:*** *Framework* principal para o desenvolvimento da API RESTful. Reconhecido por sua alta performance e baixo consumo de recursos (FRAMEWORK, 2025);

- **Objection.js e Knex:** Respectivamente, ORM baseado em Modelos e construtor de consultas SQL. Essa combinação facilitou a manipulação de dados no banco relacional escolhido, garantindo flexibilidade e escalabilidade (OBJECTION.JS, 2025; KNEX.JS, 2025);

- **Zod:** Ferramenta de validação de *schemas* baseada em TypeScript, que permite definir, de forma tipada e declarativa, as estruturas esperadas de entrada para cada *endpoint* (INFERENCE, 2025);

- **JWT:** Ou JSON Web Token, é um padrão aberto (RFC 7519) que define uma maneira compacta e autônoma de transmitir informações de forma segura entre duas partes. Ele é comumente usado para autenticação, pois permite que um servidor verifique a identidade de um usuário e retorne um token que este aplicativo pode usar para acessar recursos protegidos sem precisar enviar as credenciais novamente a cada solicitação.

4.1.2 Desenvolvimento das arquiteturas de aprendizado profundo

Bases de Dados para Treinamento e Testes

Para o treinamento e avaliação dos modelos, foram utilizadas bases de dados públicas amplamente reconhecidas na área de reidentificação de pessoas e detecção de objetos, como:

- **DukeMTMC-VidReID**: Para treinamento e avaliação dos modelos de reidentificação (ZHENG; ZHENG; YANG, 2017). Ela foi derivada do DukeMTMC, um *dataset* original de múltiplas câmeras para vigilância, com anotações adaptadas para tarefas de Re-ID por vídeo. O DukeMTMC-VidReID contém cerca de 1.812 identidades únicas, distribuídas em 4.832 imagens, capturadas por 8 câmeras diferentes. O conjunto de dados é dividido em 16.522 imagens para treinamento, 2.196 para consulta e treino e 2.636 para a galeria de teste. Uma das principais características do DukeMTMC-VidReID é a grande variação de ângulos de câmera, iluminação e poses, o que o torna um *benchmark* desafiador e amplamente adotado em pesquisas recentes de Re-ID. A base foi organizada em três conjuntos:

    1. 70% para Treinamento: Imagens usadas para atualização dos pesos do OSNet;
    2. 15% para Validação: Usadas para ajuste de hiperparâmetros e evitar sobreajuste;
    3. 15% para Teste: Exclusivas para avaliação final da acurácia da detecção.

- **COCO**: Para tarefas relacionadas à detecção de pessoas no vídeo (LIN *et al.*, 2014). O COCO (*Common Objects in Context*) é uma das bases de dados mais utilizadas no treinamento de modelos de detecção de objetos, incluindo pessoas. Ele contém mais de 330.000 imagens, das quais cerca de 200.000 são anotadas com mais de 1,5 milhão de instâncias de objetos. Ao todo, o *dataset* abrange 80 categorias de objetos. O COCO oferece anotações detalhadas em formato JSON, incluindo as coordenadas de *bounding boxes*, máscaras de segmentação e pontos-chave, sendo amplamente utilizado em *benchmarks* de detecção, segmentação e reconhecimento de pessoas. A base foi organizada em três conjuntos:

    1. 70% para Treinamento: Imagens usadas para atualização dos pesos do YOLOv11;
    2. 15% para Validação: Usadas para ajuste de hiperparâmetros e evitar sobreajuste;
    3. 15% para Teste: Exclusivas para avaliação final da acurácia da detecção.

Essas bases foram escolhidas por sua diversidade, qualidade e ampla adoção em *benchmarks* acadêmicos.

Ambas as proporções seguem recomendações da comunidade científica, garantindo volume suficiente de dados para ajuste de hiperparâmetros sem comprometer a avaliação final.

Bibliotecas e *Frameworks* de aprendizado de máquina

Foram utilizadas as seguintes bibliotecas e *frameworks* para o treinamento dos modelos:

- **YOLOv11:** Modelo para detecção de pessoas nos vídeos (ULTRALYTICS, 2024);
- **PyTorch**: *Framework* de aprendizado profundo para treinamento e inferência (PYTORCH, 2025);
- **Ultralytics YOLO**: Implementação popular e otimizada do YOLOv11, com suporte a Python (ULTRALYTICS, 2025);
- **CUDA e cuDNN (opcional)**: Para o treinamento realizado em GPU, foram necessárias as bibliotecas CUDA Toolkit e cuDNN para aceleração computacional (CORPORATION, 2025a; CORPORATION, 2025b);
- **OSNet (Omni-Scale Network):** Modelo para extração de *embeddings* e reidentificação de indivíduos (ZHOU *et al.*, 2019);
- **Torchreid**: Biblioteca *open-source* para experimentos de Re-ID, contendo implementações pré-treinadas de modelos como OSNet, PCB e AGW (TORCHREID, 2025);

- **NumPy e SciPy**: Bibliotecas auxiliares para manipulação de vetores, arrays e cálculo de distâncias (NUMPY, 2025; SCIPY, 2025).

Ferramentas para desenvolvimento da API para comunicação com os modelos treinados

- **Python:** Linguagem amplamente utilizada na área de Inteligência Artificial e aprendizado profundo, devido à sua extensa gama de bibliotecas especializadas para aprendizado de máquina (Python Software Foundation, 2025);
- **FastAPI:** *Framework* leve e de alta performance em Python, usado para a exposição da API pós comunicação com os modelos e para a conexão *Websocket* (RAMíREZ, 2023);
- **OpenCV:** Biblioteca para processamento de vídeo e manipulação de imagens em tempo real (OPENCV, 2025);
- ***Google Colab Research:*** Ferramenta fornecida pela *Google Cloud* para pesquisas baseadas em treinamento de modelos (GOOGLE, 2025);
- **AES**: Algoritmo *Advanced Encryption Standard* (AES) em modo EAX. Nesse processo, os *embeddings* de referência são convertidos em bytes e criptografados com uma chave definida em configuração. O uso do modo EAX assegura tanto a confidencialidade quanto a integridade dos dados, permitindo posteriormente a descriptografia segura dos *embeddings* originais quando necessário.

4.2 Métodos

O desenvolvimento da solução foi dividido em etapas técnicas bem definidas, contemplando desde o treinamento e configuração dos modelos de Inteligência Artificial até a implementação da API, interface *web* e medidas de segurança de dados.

4.2.1 Detecção de *Bounding Boxes*

Na primeira etapa, chamada de detecção, foi usado o modelo YOLOv11, que analisa cada quadro de um vídeo e localiza objetos de interesse, como pessoas, retornando *bounding boxes* com suas respectivas classificações.

O modelo é treinado para localizar e gerar *bounding boxes* ao redor das pessoas em cada *frame* do vídeo. Esses recortes são posteriormente processados para extração de características, constituindo o ponto de partida do *pipeline* de reidentificação. A base para o treinamento foi a COCO, já descrita na Seção 4.1.2.

Figura 6 – Exemplo de detecção de objetos com YOLO
Fonte: You Only Look Once: Unified, Real-Time Object Detection (REDMON *et al.*, 2016).

- Parâmetros de Treinamento do YOLOv11:

    1. **Taxa de aprendizado:** Definida inicialmente como 0,001. Este valor foi escolhido para permitir atualizações graduais dos pesos, evitando oscilações durante o treinamento;
    2. ***Batch Size*:** Um lote de 64 amostras foi utilizado, equilibrando entre velocidade de processamento e estabilidade da atualização dos gradientes;
    3. **Número de épocas:** Fora um ciclo de 100 a 150 épocas com um modelo parcialmente treinado, com monitoramento da perda de validação, evitando sobreajuste;
    4. **Função de perda:** Foram utilizadas as funções padrão da arquitetura YOLOv11: *bounding box regression loss*, *objectness loss* e *classification loss*;
    5. ***Data Augmentation*:** Técnicas como *random horizontal flip*, *scale jittering*, *color jitter* e *random crop* foram aplicadas para aumentar a acurácia do modelo diante de variações de iluminação, ângulo e posição (GONZALEZ; WOODS, 2008);
    6. ***Dropout:*** Embora a arquitetura YOLOv11 minimize o uso de *dropout*, técnicas de regularização como *early stopping* foram empregadas.

Durante o processo de treinamento do YOLOv11, observou-se que a configuração inicial resultava em flutuações no gradiente e consumo excessivo de memória. Foram, portanto, realizados ajustes progressivos com base nas métricas de validação e nos limites do ambiente de execução (GPU Tesla T4, 16 Gigabyte (GB) (GB)s de VMemória de Acesso Aleatório (RAM, do inglês Random Access Memory) (RAM)).

Estes foram computados em diferentes experimentos para delimitar os valores dos hiperparâmetros e posteriormente foram ajustados com base na análise dos resultados obtidos durante o processo de treinamento. Os principais ajustes foram:

- **Taxa de aprendizado (lr0):** ajustada de 0.001 para 0.0049, resultando em convergência mais rápida e estabilidade na perda de validação;

- ***Batch Size*:** reduzido de 64 para 32 amostras, mitigando erros de alocação de memória e estabilizando o gradiente;
- **Camadas congeladas:** adicionou-se o parâmetro `freeze=5` para preservar pesos de camadas iniciais e evitar sobreajuste;
- **Otimizador:** alterado para *AdamW*, permitindo melhor controle da regularização L2 e reduzindo a tendência de sobreajuste;
- **Funções de perda:** ajustadas conforme segue — `box`=8.0, `cls`=0.5, `dfl`=1.5, `kobj`=1.0 — resultando em melhor equilíbrio entre as penalizações de classificação e regressão espacial;
- **Dropout:** reduzido de 0.2 para 0.0, após verificar que a arquitetura já incorporava camadas implícitas de regularização (*batch normalization*);
- **Treinamento multi-escala:** habilitado (`multi_scale=True`), melhorando a robustez a diferentes resoluções de entrada;
- **Aprimoramento de precisão mista:** ativado (`amp=True`), otimizando uso de VRAM sem perda de desempenho.

O principal objetivo desses ajustes foi maximizar a métrica *mAP* na detecção de pessoas, mantendo o equilíbrio entre precisão e custo computacional.

Os valores foram definidos a partir das recomendações da documentação oficial do YOLOv11 e de experimentos reportados na literatura recente (ULTRALYTICS, 2024), buscando replicar práticas consolidadas de ajuste fino em modelos de detecção em tempo real.

Além disso, empregou-se o método nativo de afinação de hiperparâmetros disponibilizado pelo *framework* Ultralytics, que integra o *Ray Tune*, uma ferramenta de otimização distribuída voltada ao ajuste automático de hiperparâmetros em modelos de aprendizado profundo (LIANG *et al.*, 2018). Essa abordagem permitiu explorar de forma sistemática diferentes combinações de parâmetros, favorecendo a convergência estável do modelo e a obtenção de métricas de desempenho mais consistentes.

4.2.2 Extração de *Embeddings*

Na segunda etapa, chamada de extração de *embeddings*, o modelo de reidentificação aplica uma RNC sobre as imagens recortadas das pessoas detectadas. O resultado é um vetor numérico que representa matematicamente as características visuais do indivíduo, como cor da roupa, proporção corporal, padrões e texturas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Para cada pessoa detectada, foi aplicado um modelo leve de reidentificação (Re-ID), o OSNet (ZHOU *et al.*, 2019), que transforma a imagem recortada da pessoa em um *embedding*.

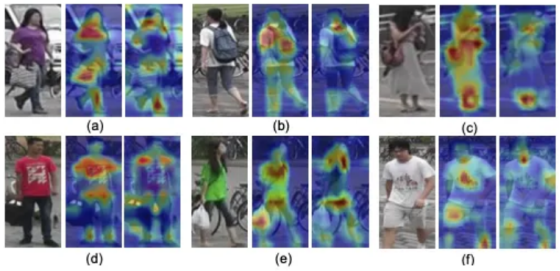

Figura 7 – Exemplo de mapeamento de *embeddings*.
Fonte: Omni-Scale Feature Learning for Person Re-Identification. (ZHOU *et al.*, 2019)

Como o sistema não parte de um banco de dados pré-existente, é possível fazer *upload* de um *embedding* de referência previamente extraído a partir de uma imagem fornecida pelo usuário final, por exemplo, uma captura de tela, ou uma foto da pessoa a ser rastreada, preferencialmente de corpo todo;

Para o treinamento e validação do OSNet, foi utilizada a base pública DukeMTMC-VidReID, já descrita na Seção 4.1.2.

- Parâmetros de Treinamento da OSNet

    1. **Taxa de aprendizado:** Inicialmente definida como 0,0003 com decaimento progressivo, utilizando o *scheduler* StepLR para reduzir a taxa em um fator de 0,1 a cada 40 épocas. (ZHOU *et al.*, 2019);
    2. **Otimizador:** Função Adam com β_1=0.9 de decaimento exponencial da media dos gradientes e β_2=0.999 de decaimento exponencial da média dos quadrados dos gradientes, proporcionando melhor convergência em problemas de classificação complexos;
    3. **Função de Perda:** A função *Triplet Loss* foi utilizada por ser amplamente empregada em tarefas de reidentificação de pessoas, nas quais o objetivo principal é aprender um espaço de representação discriminativo.
    4. ***Batch Size*:** Um lote de 64 amostras foi utilizado, para equilibrar velocidade de processamento e estabilidade da atualização dos gradientes, porém a memória RAM do GPU não conseguiu carregar os *batches* inteiros, logo o *batch* foi diminuído para 6 imagens;
    5. **Número de épocas:** Foram utilizado um ciclo de 250 épocas, com monitoramento da perda de validação para evitar sobreajuste;
    6. ***Data Augmentation*:** Técnicas como *random horizontal flip*, *random erasing* e *random crop* foram aplicadas para aumentar a acurácia do modelo diante de variações de iluminação, ângulo e posição.

A imagem de referência fornecida pelo usuário é processada pelo OSNet para gerar o *embedding*, que é salvo como *blob* e depois utilizado para comparação na análise do vídeo.

Estes foram escolhidos com base em recomendações da documentação oficial do OSNet e em experimentos descritos na literatura recente (ZHOU *et al.*, 2019).

Comparação Vetorial

O usuário da aplicação (*client*) é responsável por fornecer a imagem de referência do indivíduo que se deseja rastrear, como comentado anteriormente. A cada detecção de pessoa no vídeo, o sistema calcula a distância entre o vetor de referência (utilizando a métrica de distância euclidiana, mais recomendada para *embeddings*) e os vetores extraídos das pessoas detectadas no vídeo. As menores distâncias indicam maior similaridade, permitindo identificar recorrências do mesmo indivíduo ao longo do tempo e de diferentes cenas;

A etapa de correspondência (*matching*) entre a imagem de referência e as detecções no vídeo é realizada utilizando métricas de distância vetorial. A principal métrica adotada é a distância euclidiana, calculada conforme a Equação 2:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2}$$

Onde x representa o vetor de *embedding* da imagem de referência e y o vetor de uma detecção no vídeo.

4.2.3 Avaliação do desempenho

A avaliação da acurácia da detecção de pessoas e da reidentificação foram realizada com base nas seguintes métricas:

- Para Detecção (YOLO) (ULTRALYTICS, 2024):

    1. **mAP (*mean Average Precision*):** Média das precisões em múltiplos níveis de *Intersection over Union* (IoU). Mede a qualidade das caixas geradas;

$$\text{AP} = \int_0^1 p(r)\, dr \tag{3}$$

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_i \tag{4}$$

    Onde:

    - $p(r)$ é a precisão em função do *recall*;
    - N é o número total de classes.

2. **IoU *(Intersection over Union*):** Métrica que avalia a sobreposição entre a caixa prevista e a real;

$$\text{IoU} = \frac{\text{Área da Interseção}}{\text{Área da União}} \tag{5}$$

3. ***Precision* e *Recall*:** Avaliação da taxa de verdadeiros positivos e cobertura de detecção.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

Onde:

- TP são os verdadeiros Positivos;
- TN são os falsos positivos;
- FP são os falsos positivos;
- FN são os falsos negativos.

- Para Re-ID (OSNet) (ZHOU *et al.*, 2019):

1. ***Rank-k Accuracy***: Indica a frequência com que a pessoa correta aparece nas k primeiras posições da lista de resultados;

$$\text{Rank-k Accuracy} = \frac{\text{Nº de consultas positivas nas k primeiras posições}}{\text{Número total de consultas}} \tag{8}$$

2. **mINP (*mean Inverse Negative Penalty*):** Mede a robustez do modelo considerando variações nos resultados negativos;

$$\text{INP} = \frac{1}{P} \sum_{i=1}^{P} \frac{1}{r_i} \tag{9}$$

Onde:

- P é o número total de consultas;
- r_i é o Rank da primeira ocorrência correta para a consulta.

3. **CMC (*Cumulative Matching Characteristic*):** Avalia o desempenho cumulativo da reidentificação por Rank (ZHOU *et al.*, 2019).

As fórmulas dessas métricas estão descritas na literatura (ZHENG *et al.*, 2015; ZHOU *et al.*, 2019) e foram implementadas como *scripts* de validação paralela ao sistema quando não havia métodos pré-prontos provindos dos *frameworks* de treino.

4.2.4 Integração do modelo via API

A comunicação com o sistema foi viabilizada por meio de uma API desenvolvida com o *framework* FastAPI. Foram implementados *endpoints* específicos para as seguintes funções:

- /upload-hash: Recebimento da imagem de referência, que será convertida em *embedding*;
- /video-stream: Conexão com transmissões ao vivo, via protocolo *websocket*;
- /find: Requisições de busca por aparições da pessoa de interesse no vídeo.

4.2.5 Desenvolvimento da Interface Web

Nesta etapa, foi desenvolvida uma interface gráfica minimalista, conectada a uma API escalável, com o objetivo de viabilizar testes, demonstrações e a utilização prática do sistema por usuários finais. A interface permite a visualização dos vídeos processados com marcações visuais nas *bounding boxes*, indicação dos *timestamps* das ocorrências e identificação dos indivíduos reconhecidos no vídeo.

O desenvolvimento da API e da interface segue os princípios de Design Direcionado Domínio (DDD do inglês *Domain Driven Design*) (DDD), uma abordagem arquitetural proposta por Eric Evans (EVANS, 2003), cujo foco é alinhar a estrutura de *software* ao domínio de negócio, promovendo um código mais organizado, desacoplado e de fácil manutenção. O DDD tem como conceitos centrais os seguintes:

- **Interface (*Presentation Layer*):** Essa camada é responsável por receber as requisições dos usuários (via interface *web* ou chamadas de API RESTful) e entregar as respostas apropriadas. Inclui os controladores de rotas, validações iniciais de entrada (em conjunto com o Zod) e os adaptadores que traduzem as requisições externas em comandos compreendidos internamente pela aplicação;
- **Camada de Aplicação (*Application Layer*):** A camada de aplicação é responsável por coordenar o fluxo de execução das operações de negócio, sem implementar regras de negócio propriamente ditas. Ela orquestra os casos de uso, chamando os serviços do domínio, gerenciando transações e controlando o fluxo de dados entre as outras camadas;

- **Camada de Domínio (*Domain Layer*):** Esta camada contém o núcleo das regras de negócio da aplicação. Aqui estarão definidas as entidades, *value objects*, serviços de domínio, regras de validação específicas e as interfaces dos repositórios. No contexto deste projeto, conceitos como Pessoa, Vídeo e Usuário serão modelados como entidades ou objetos de valor;

- **Camada de Infraestrutura (*Infrastructure Layer*):** A camada de infraestrutura é responsável por implementar os detalhes técnicos da aplicação. Isso inclui o acesso ao banco de dados (via Knex e Objection), serviços de armazenamento de arquivos (como os vídeos e *embeddings*) e comunicação com a Inteligência Artificial (incluindo o disparo das detecções via Python). Essa camada também é responsável pela integração com bibliotecas externas, como o Sphinx para documentação.

Ao adotar o DDD, o projeto garante que suas regras de negócio sejam expressas de forma clara e independente da camada de infraestrutura, facilitando futuras expansões, como a adição de novas fontes de vídeo, suporte a múltiplos algoritmos de reidentificação ou a integração com sistemas externos.

4.2.6 Criação da documentação

A documentação técnica da API foi gerada utilizando o Sphinx, com formato de saída em HTML estático, integrando exemplos de uso, explicações de cada *endpoint* e detalhes sobre o funcionamento interno dos módulos de IA. Além disso, foi criado um *OpenAPI Specification* para facilitar o consumo da API por desenvolvedores externos.

5 RESULTADOS FINAIS

Este capítulo apresenta os resultados obtidos no desenvolvimento da solução de rastreamento de pessoas em vídeo baseada em Inteligência Artificial. São descritas as etapas concluídas, bem como uma breve análise dos dados coletados em conformidade com os objetivos estabelecidos nas Seções 1.1.1 e 1.1.2. Além disso, são apresentados os protótipos desenvolvidos para o sistema, considerando que este Trabalho de Conclusão de Curso consiste em uma proposta de caráter experimental, com foco na usabilidade, segurança de acesso e no uso ético de técnicas de Inteligência Artificial aplicadas ao rastreamento de pessoas em vídeo.

5.1 Telas

Nesta seção, são apresentados as telas de interface elaboradas para a aplicação *web* e o painel administrativo do sistema de rastreamento de pessoas. O foco principal foi criar um fluxo de interação simples, seguro e com baixo nível de complexidade operacional, considerando usuários com diferentes níveis de conhecimento técnico. As telas foram projetadas no Figma e desenvolvidas por componentização em React Native e Expo, seguindo princípios de usabilidade e *design* responsivo.

Autenticação e Controle de Acesso

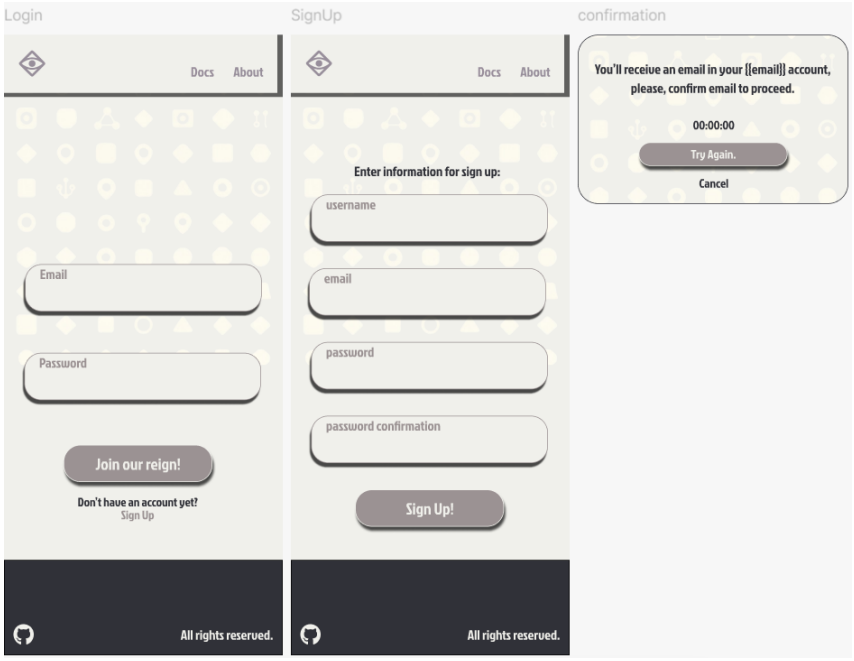

Figura 8 – Telas de Registro e Autenticação.
Fonte: Autoria Própria

A primeira interação do usuário com o sistema é a tela de autenticação, conforme mostra na Figura 8, na qual este deve inserir suas credenciais (email e senha) para obter acesso ao sistema. O mecanismo de autenticação implementa tokens JWT (*JSON Web Tokens*) assinados via algoritmo HS256, com payload contendo identificador único do usuário (sub), id do usuário (userId) e timestamps de emissão (iat), armazenados de forma segura em *http-only cookies*, visando mitigar riscos de ataques por *Cross-site scripting*) (XSS).

A expiração do *token* é de 15 minutos, necessitando de renovação periódica através de um *endpoint* /auth/refresh que valida um refresh token com duração estendida de 1 hora.

Além disso, foi implementado um esquema de controle de acesso Controle de Acesso Baseado em Papéis (RBAC do inglês *Role-Based Access Control*) (RBAC) com dois níveis de permissão, conforme mostra na Figura 9:

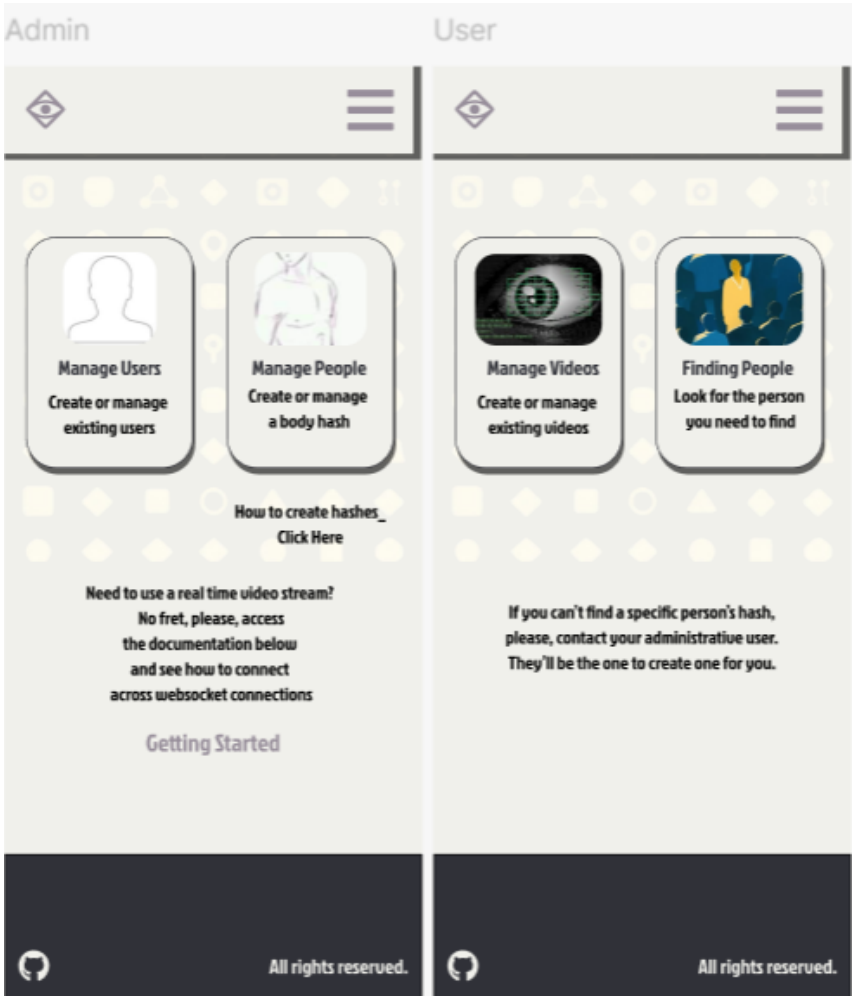

Figura 9 – Acesso RBAC pós-autenticação.
Fonte: Autoria Própria

Administrador (admin): Acesso total ao sistema, incluindo gerenciamento de usuários (criar, editar, excluir), visualização de todos os vídeos e *embeddings* e remoção de *embeddings* associados a indivíduos específicos e gerais.

Usuário Padrão: Acesso restrito ao seu próprio contexto, permitindo gerenciamento de perfil, criação e deleção de vídeos de referência, manipulação de *embeddings* próprios e busca de indivíduos em vídeos já processados ou transmissões ao vivo, conforme na Figura 10.

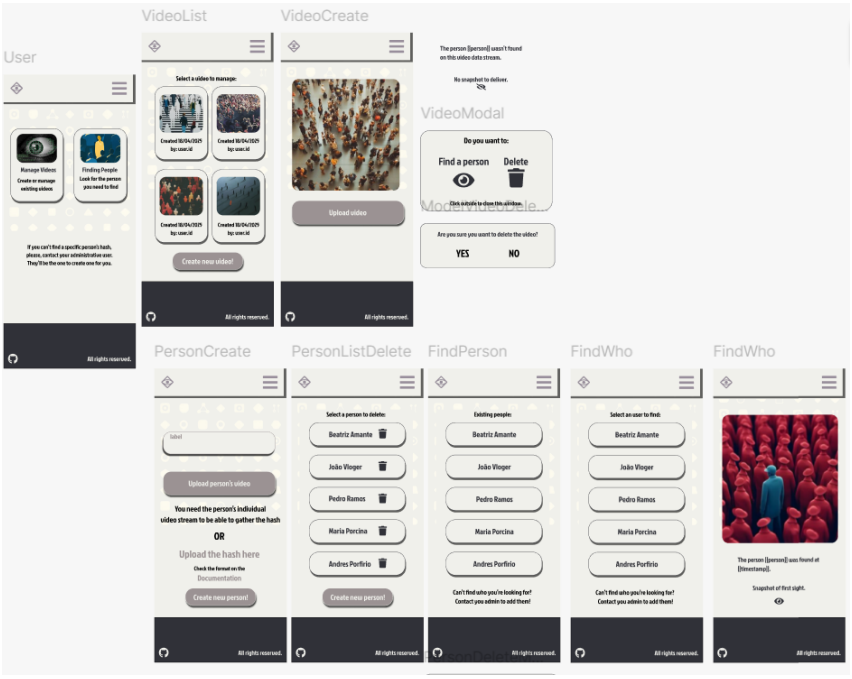

Figura 10 – Telas acessíveis ao Usuário.
Fonte: Autoria Própria

O sistema garante que campos sensíveis, como o tipo de perfil (*role*), não sejam acessíveis para usuários comuns no *frontend*, evitando escalonamentos indevidos de privilégio.

Gerenciamento de Usuários

A tela de Gerenciamento de Usuários, com exemplo mostrado na Figura 11, é acessada apenas pelo Administrador, e este pode:

- Criar um novo usuário. Por segurança, todos novos usuários iniciam suas *roles* como 'USER';
- Visualizar os usuários cadastrados;
- Fazer *update* das informações dos usuários já existentes.

O usuário padrão também terá acesso ao *update*, porém a ele não é liberada a opção de *role*.

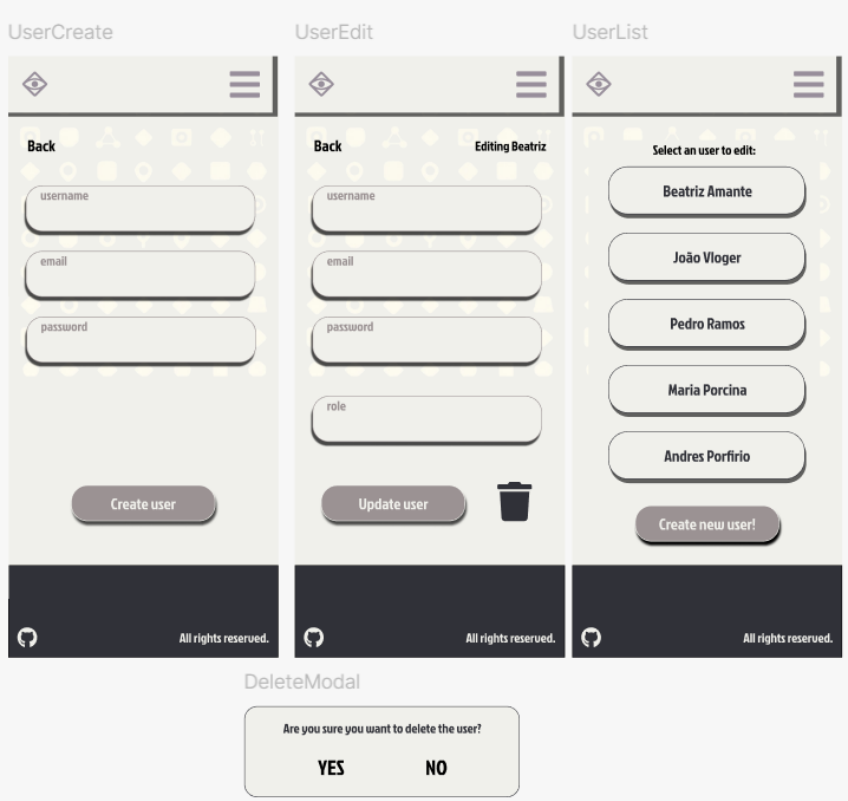

Figura 11 – Gerenciamento de usuários.
Fonte: Autoria Própria

Gerenciamento de Vídeos

Após a autenticação, o usuário é redirecionado para a interface principal, onde poderá realizar o *upload de vídeos* ou estabelecer uma conexão via *websocket* para processar transmissões em tempo real.

Usuários podem:

- Listar os vídeos já enviados;
- Selecionar um vídeo específico para análise de detecção/reidentificação.

A interface permite, de maneira intuitiva, associar um vídeo a um processo de detecção ou reidentificação com base nos *embeddings* previamente cadastrados.

Gerenciamento de Pessoas *(embeddings)*

Na tela de Gerenciamento de Pessoas, com exemplo mostrado na Figura 13, o usuário pode:

- Realizar o *upload* de uma imagem de referência (que será convertida em *embedding*);
- Visualizar os *embeddings* cadastrados relacionadas ao usuário logado em questão;

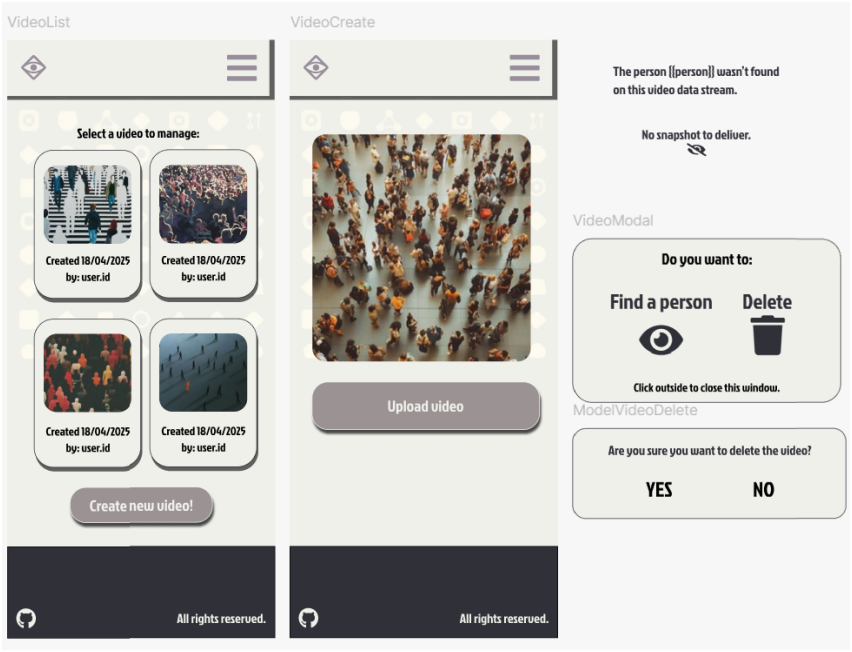

Figura 12 – Gerenciamento de vídeos.
Fonte: Autoria Própria

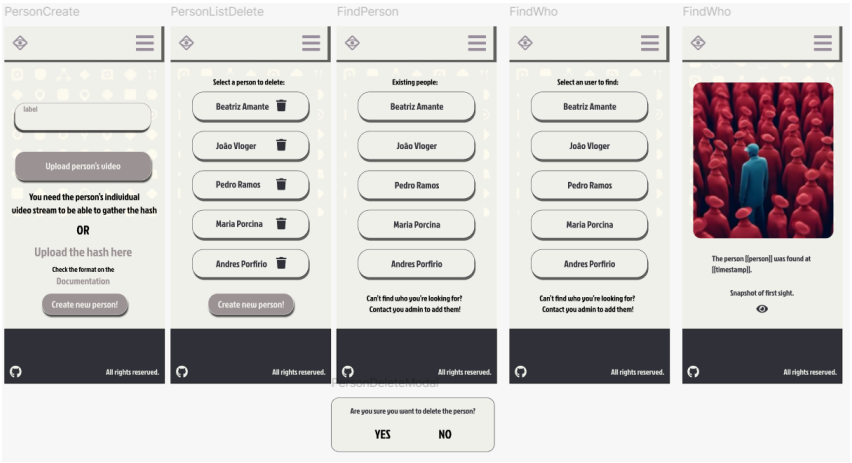

Figura 13 – Gerenciamento de pessoas.
Fonte: Autoria Própria

- Excluir imagens de referência já existentes relacionadas ao usuário logado em questão.

A busca por indivíduos é feita selecionando o *embedding* de interesse de uma lista, que é comparado com as detecções presentes nos vídeos selecionados ou na transmissão ao vivo.

Modelagem de banco

A partir dos requisitos funcionais levantados durante a fase de planejamento, foi elaborada a modelagem lógica do banco de dados para suportar as operações essenciais da aplicação de rastreamento de pessoas. O objetivo principal desta modelagem foi garantir uma estrutura que atendesse tanto às regras de negócio relacionadas ao gerenciamento de usuários e

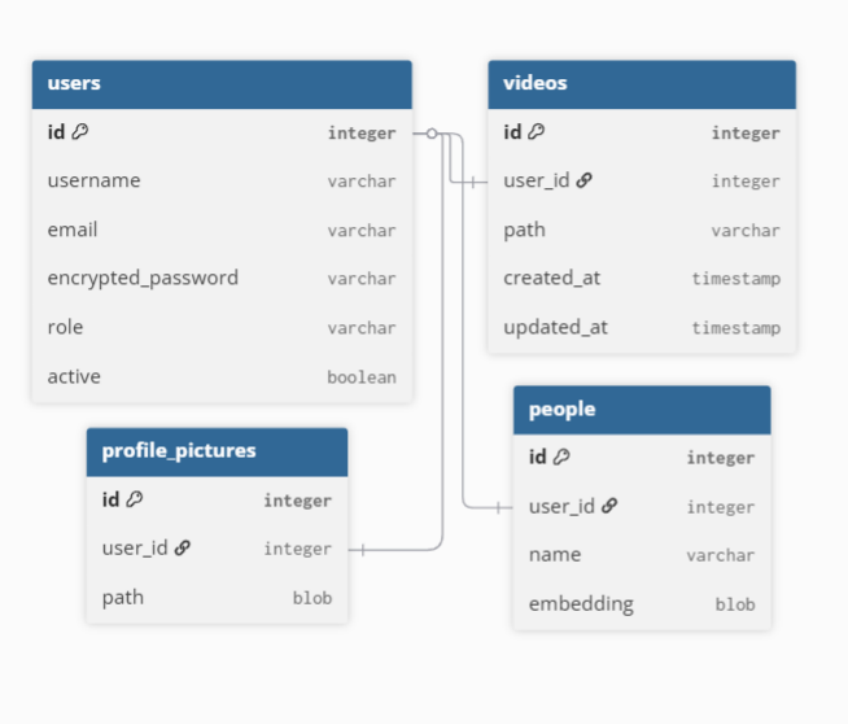

Figura 14 – Modelagem do banco de dados.
Fonte: Autoria Própria

vídeos quanto às necessidades específicas da manipulação de *embeddings* para reidentificação de pessoas.

A modelagem do banco de dados foi projetada de forma relacional, buscando garantir a integridade referencial e a escalabilidade do sistema. A estrutura foi normalizada até a Terceira Forma Normal (3NF), eliminando redundâncias e dependências transitivas, o que contribui para maior consistência e eficiência nas operações de consulta e atualização dos dados.

A Figura 14 apresenta o diagrama de entidade-relacionamento (DER) que representa o modelo lógico do sistema. A modelagem reflete diretamente os agregados definidos na camada de domínio descrita na Seção 4.2.5, mantendo alinhamento conceitual com a arquitetura orientada a domínio (*Domain-Driven Design*).

A tabela **users** constitui a entidade central do modelo, responsável por armazenar informações de autenticação e controle de acesso. Contém os atributos *username*, *email*, *password*, *active* e *role*, este último implementado como um tipo enumerado (*enum: admin|user*) para diferenciação de permissões. O identificador primário (*PK*) é o campo *id*, do tipo UUID, garantindo unicidade global e reduzindo riscos de colisão. Um índice foi criado sobre o campo *email*, a fim de otimizar o processo de autenticação.

A entidade **people** armazena as informações referentes aos indivíduos rastreados nos vídeos. Cada registro está vinculado a um usuário específico por meio do campo *user_id*, definido como chave estrangeira (*FK*) com política de exclusão em cascata (*ON DELETE CASCADE*), o que assegura isolamento e integridade dos dados entre diferentes contas. Além disso,

o campo *embedding* armazena o vetor criptografado gerado pelo modelo de reidentificação (OSNet), enquanto o campo *name* referencia o identificador nominal ou descritivo da pessoa.

A tabela **profile_pictures** representa uma relação um-para-um (*one-to-one*) com a entidade **users**, armazenando o caminho da imagem de perfil utilizada. Essa estrutura facilita a associação de uma imagem base por usuário, sem redundância de dados.

Por fim, a entidade **videos** é responsável por registrar os metadados dos arquivos de vídeo enviados à aplicação, incluindo o caminho de armazenamento (*path*), além dos campos de controle temporal *created_at* e *updated_at*, permitindo rastreabilidade e versionamento. Cada vídeo está associado a um único usuário por meio do campo *user_id*, também definido como chave estrangeira.

Essa modelagem garante modularidade e consistência entre os módulos de autenticação, rastreamento e armazenamento, além de refletir diretamente os princípios de encapsulamento e responsabilidade única adotados na arquitetura geral do sistema.

5.2 Desenvolvimento da API e Arquitetura do Sistema

O desenvolvimento da API foi implementado em *TypeScript* (v5.9.2), utilizando o *framework Fastify* (v5.4.0) para Node.js, reconhecido por sua alta performance e eficiência no tratamento de requisições HTTP. A comunicação entre os módulos da aplicação segue o padrão arquitetural RESTful, com a definição de *endpoints* específicos para as principais operações de envio e recebimento de dados.

Foram integrados diversos *middlewares* para garantir segurança e consistência nas requisições, incluindo autenticação via *JSON Web Token* (JWT), validação de esquemas de entrada com o Zod (v4.0.15) e documentação automatizada dos *endpoints* por meio do OpenAPI/Swagger (v9.5.2), que oferece uma interface visual interativa para desenvolvedores e usuários técnicos.

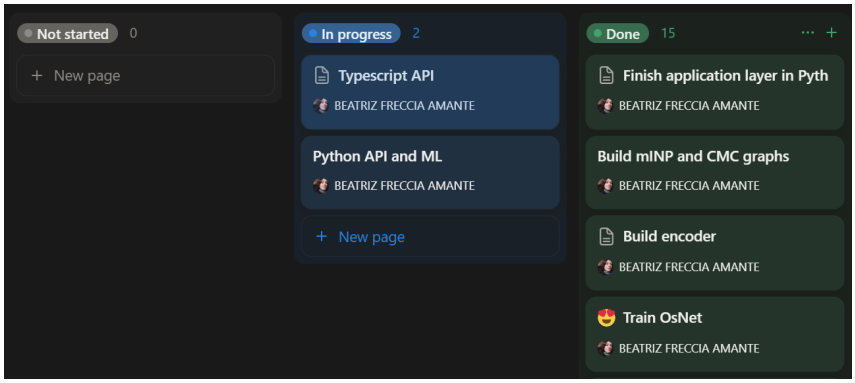

Figura 15 – Fluxo de desenvolvimento por Notion, estilo Kanban.
Fonte: Autoria própria.

A organização do fluxo de desenvolvimento seguiu o modelo ágil de *sprints*, registradas e acompanhadas por meio de quadros Kanban na ferramenta Notion, segundo a Figura 15. Essa

metodologia possibilitou a distribuição estruturada das tarefas, o acompanhamento contínuo do progresso e a adaptação iterativa das entregas, favorecendo um desenvolvimento mais eficiente e colaborativo.

A API responsável pela comunicação entre os modelos de inteligência artificial e a interface foi projetada de acordo com os princípios do DDD, conforme proposto na Seção 4.2.5, garantindo que as entidades e regras de negócio estejam consistentes e mantenham coerência em toda a aplicação. O resultado foi uma base de código mais estável, desacoplada e de fácil manutenção, além de facilitar a evolução futura do sistema e a incorporação de novos módulos. Os principais agregados do domínio (**User**, **Person** e **Video**) encapsulam as regras de negócio fundamentais do sistema.

A persistência de dados foi abstraída por meio da aplicação do *Repository Pattern*, o que permite que a lógica de acesso ao banco de dados seja isolada das regras de domínio, promovendo baixo acoplamento e alta coesão entre os componentes da arquitetura.

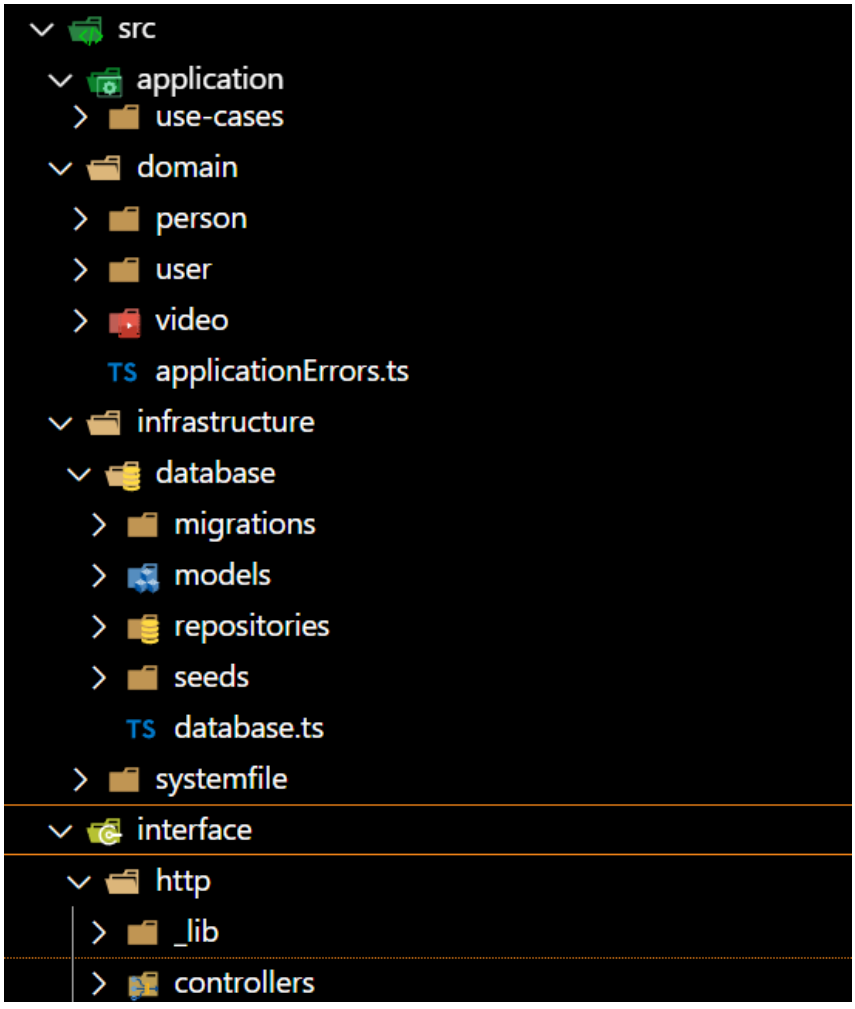

Figura 16 – Visual da arquitetura *Domain Drive Design*.
Fonte: Autoria própria.

Durante a fase inicial desse desenvolvimento, foi feita a configuração do ambiente do sistema, com as dependências e configurações necessárias, e então foi realizada a modelagem

do domínio dos agregados principais, estabelecendo suas relações e comportamentos esperados. Uma vez definidas essas entidades, iniciou-se a implementação camada por camada, respeitando a arquitetura DDD, como visto na Figura 16 e conforme o exemplo de trecho de código no apêndice A.

Então, finalizada a API, avançou-se para a próxima etapa do protótipo, descrita abaixo.

5.3 Desenvolvimento e Treinamento das IAs

O treinamento dos modelos de inteligência artificial teve início com a implementação do *framework* Ultralytics YOLOv11. O processo foi conduzido com o objetivo de otimizar o equilíbrio entre acurácia e desempenho computacional, ajustando os principais hiperparâmetros do modelo — como taxa de aprendizado, *batch size* e número de épocas, conforme descrito na Seção 4.2.1.

Os ajustes realizados demonstram que pequenas variações nos hiperparâmetros — especialmente na taxa de aprendizado e na ponderação das funções de perda — exercem impacto direto na estabilidade do treinamento e na qualidade das detecções.

Esses ajustes também refletiram diretamente na melhoria das métricas de desempenho, especialmente na precisão média (mAP50 e mAP50-95), que apresentaram crescimento constante até a estabilização por volta da 50ª época, conforme evidenciado na Figura 17. Esta ilustra a evolução das métricas ao longo das épocas.

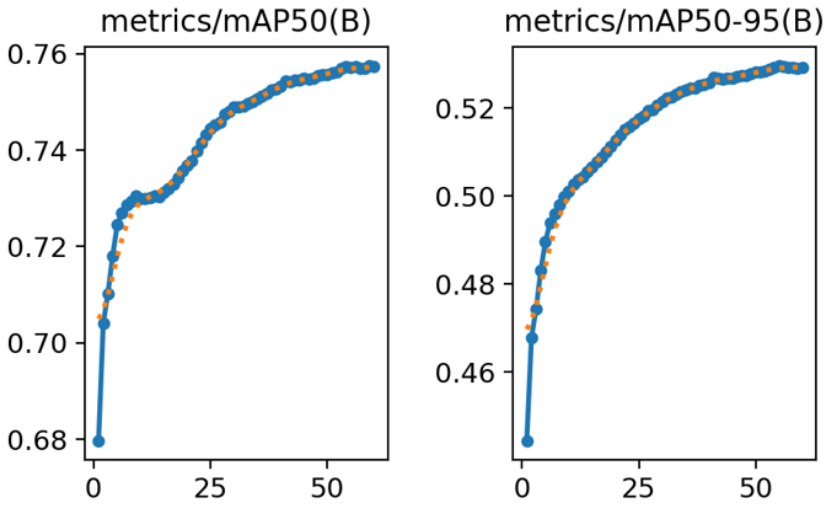

Figura 17 – Métricas mAP50 e mAP50-90 no treinamento do modelo YOLO.
Fonte: Autoria própria.

Observa-se que o modelo atingiu um mAP50 de aproximadamente 0,75 e mAP50–95 de 0,52, valores compatíveis com os obtidos em *benchmarks* de detecção em cenários de vigilância (REDMON *et al.*, 2016). Quando comparado ao YOLOv8n — que apresenta mAP50–95 em torno de 0,37 — o modelo proposto obteve um ganho de aproximadamente 40%, indicando

melhoria significativa na detecção de pessoas, mesmo em condições de iluminação e oclusão parciais

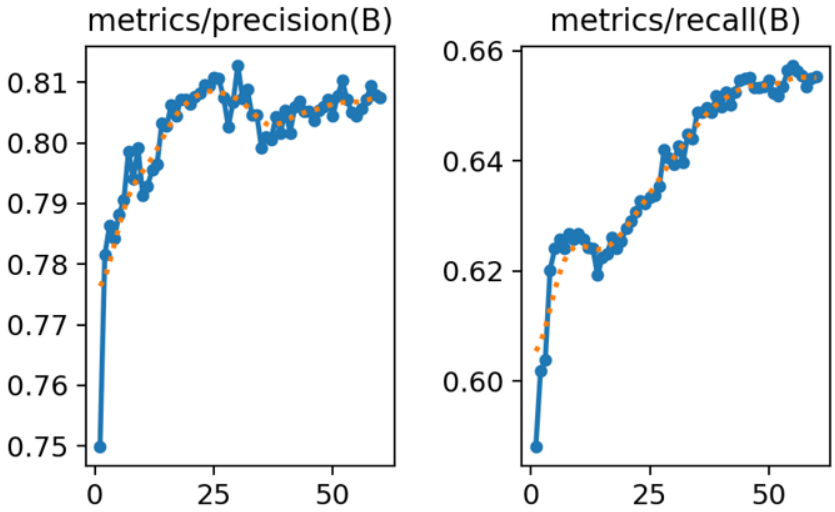

Figura 18 – Métricas *Precision* e *Recall* no treinamento do modelo YOLO.
Fonte: Autoria própria.

A métrica mAP50–95, mais rigorosa por avaliar múltiplos limiares de IoU, apresentou crescimento estável até cerca de 0,52, demonstrando maior robustez do modelo diante de variações nas *bounding boxes*.

Já na Figura 18, pôde-se avaliar que a métrica *Precision* alcançou 0,82, refletindo uma baixa taxa de falsos positivos. O *Recall*, por sua vez, estabilizou-se em torno de 0,66, valor considerado adequado para detecção de pessoas em ambientes dinâmicos, embora ainda haja espaço para otimização.

Segundo Redmon *et al.* (2016), a arquitetura YOLO tende a priorizar a precisão das detecções em detrimento da cobertura, uma vez que o modelo é projetado para minimizar predições incorretas — o que explica o leve desbalanceamento entre *Precision* e *Recall*. Além disso, conforme observado nas análises de erro, cerca de 68% dos falsos negativos ocorreram em instâncias com oclusão superior a 50%, o que sugere a necessidade de futuros refinamentos estruturais, como o uso de camadas *deconvolutional* ou mecanismos de atenção espacial para melhorar a sensibilidade a regiões parcialmente visíveis.

A Figura 19 apresenta as curvas de perda correspondentes às fases de treinamento e validação. Observa-se declínio contínuo nas perdas de classificação (cls_loss), regressão de caixas (box_loss) e função de distribuição (dfl_loss), evidenciando a efetividade do processo de aprendizado. As perdas de validação apresentaram redução mais acentuada nas primeiras 10 épocas, com tendência à estabilização após a 25ª, indicando o início da convergência do modelo.

O valor final de *val/box_loss* abaixo de 0,01 indica que as caixas delimitadoras foram bem ajustadas às regiões de interesse, com erro residual mínimo. Esse comportamento é con-

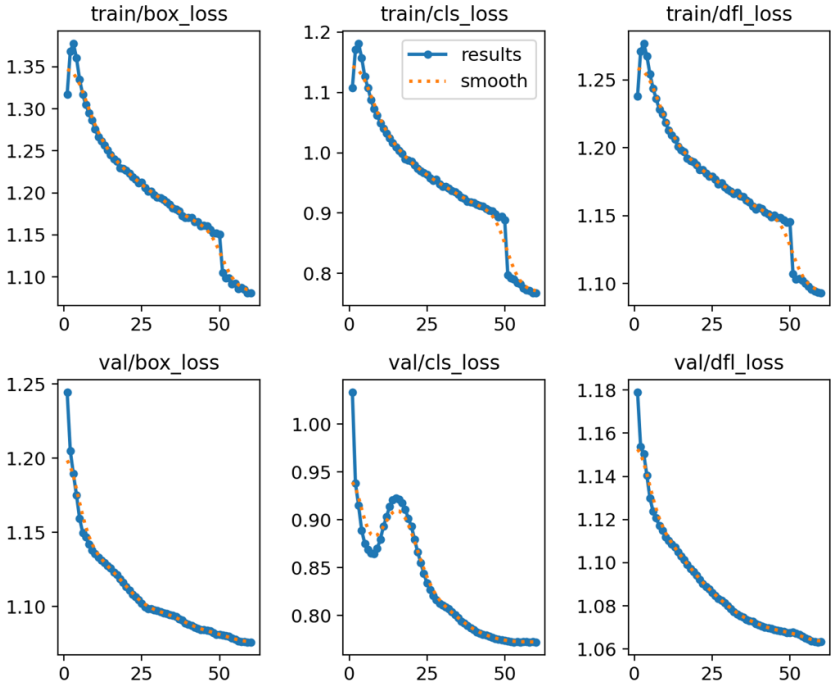

Figura 19 – Acompanhamento da convergência da Função de Perda no treinamento do modelo YOLO.

Fonte: Autoria própria.

sistente com os resultados apresentados em trabalhos correlatos sobre variações recentes do YOLO (ULTRALYTICS, 2025), reforçando a adequação do modelo à tarefa proposta.

Esses resultados demonstram que o modelo alcançou desempenho competitivo e consistente, apresentando potencial para aplicação prática em sistemas de rastreamento de pessoas. Futuras otimizações podem incluir o ajuste fino de hiperparâmetros, uso de técnicas de *data augmentation* mais específicas e reamostragem de casos de oclusão para aprimorar o *Recall* sem comprometer a precisão.

Em relação ao modelo OSNet, as métricas se apresentam da seguinte forma:

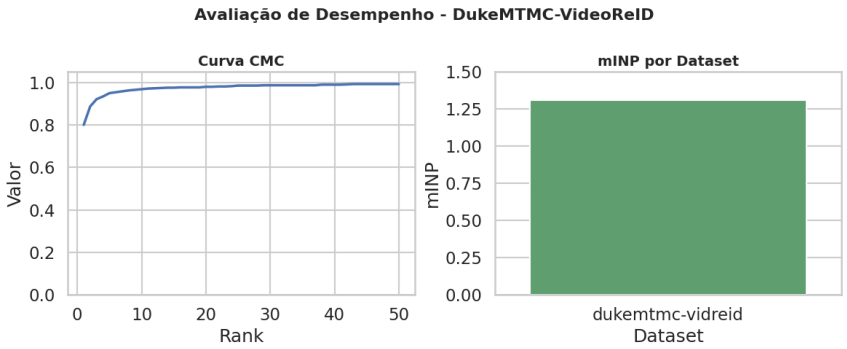

Figura 20 – Métricas *Mean Inverse Negative Penalty* e *Cumulative Matching Characteristic* no treinamento do modelo OSNet.

Fonte: Autoria própria.

As Figuras 21 e 20 apresentam os resultados obtidos pelo modelo OSNet durante o processo de reidentificação de pessoas na base Duke MTMC-VidReID. Observa-se que o mo-

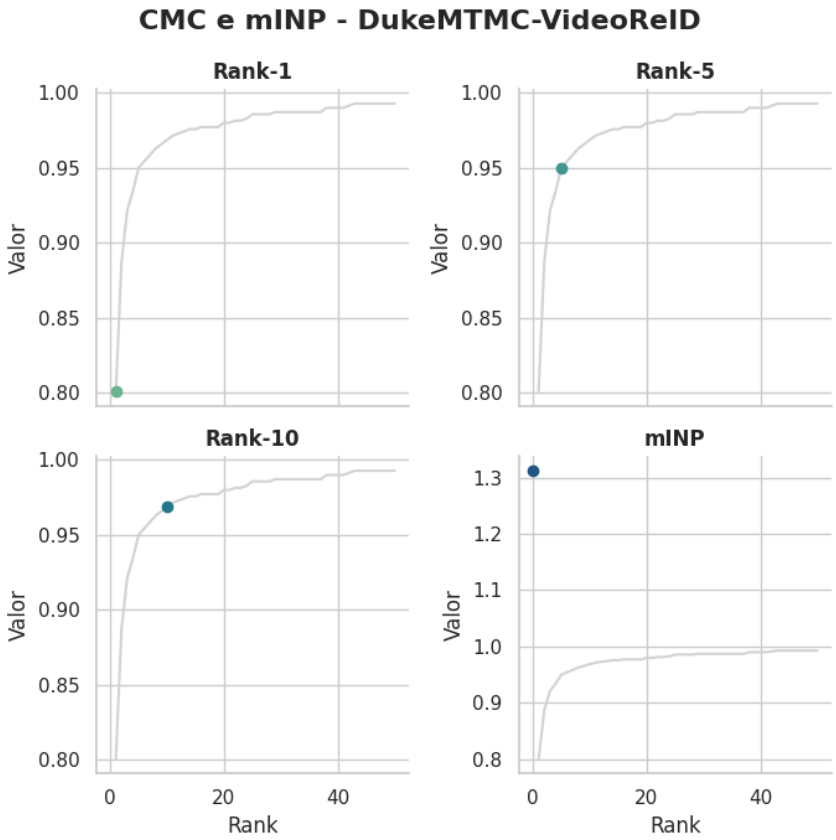

Figura 21 – Métricas *Rank-k* no treinamento do modelo OSNet.
Fonte: Autoria própria.

delo alcançou o valor de 81,9% em k-1, enquanto os valores de Rank-5, Rank-10 e Rank-20 alcançaram 95%, 96,2% e 97,3%, respectivamente.

A Curva CMC obtida evidencia esse comportamento cumulativo: à medida que o valor de k aumenta, a taxa de reconhecimento cresce de forma estável, aproximando-se de 100% em k = 20. Esse resultado demonstra que o modelo apresenta alta capacidade de generalização, reconhecendo corretamente a maioria dos indivíduos com poucas tentativas de correspondência.

Além disso, o modelo obteve um valor de mINP = 1,31, o que indica boa estabilidade na penalização de resultados negativos — ou seja, o modelo mantém desempenho consistente mesmo em casos de oclusão ou variações de iluminação, reduzindo a probabilidade de falhas severas de correspondência. Essa métrica complementa a interpretação do Rank-k, fornecendo uma medida mais robusta da consistência global da reidentificação (CHEN *et al.*, 2018).

Esses resultados demonstram que o modelo foi capaz de reconhecer corretamente a maioria dos indivíduos nas primeiras posições do ranqueamento, evidenciando alta discriminatividade dos *embeddings* gerados. A diferença entre Rank-1 e Rank-5 mostra que, mesmo quando a primeira predição não é correta, o indivíduo correto aparece entre as cinco primeiras posições em mais de 90% dos casos — um desempenho considerado competitivo para tarefas de reidentificação de pessoas em cenários com múltiplas câmeras.

Os valores estão alinhados com o desempenho relatado na literatura original do OSNet (ZHOU *et al.*, 2019), confirmando que o modelo desenvolvido neste trabalho apresenta boa capacidade de generalização, mantendo-se dentro da faixa esperada para arquiteturas leves aplicadas à reidentificação de pessoas.

Implementação dos modelos treinados para inferência em Python

Após a conclusão do treinamento dos modelos de detecção (YOLOv11) e reidentificação (OSNet), foi desenvolvida uma API em Python dedicada exclusivamente à etapa de inferência, responsável por integrar ambos os modelos dentro de um ambiente de execução contínuo. Essa API foi construída sobre um servidor com suporte a *WebSocket*, permitindo a análise de fluxos de vídeo em tempo real.

No contexto desta API, o foco recaiu sobre o Domínio **Person**, que foi isolado como a principal entidade do sistema de inferência. Essa modelagem garante consistência e rastreabilidade das identidades detectadas, mesmo em sessões distintas de análise, e permite manter a coerência sem exigir persistência de dados entre requisições, conforme o apêndice B.

Os modelos treinados foram carregados em tensores PyTorch, de forma que pudessem ser executados diretamente no servidor sem necessidade de retreino. Essa abordagem, além de reduzir a latência da inferência, permite a utilização de GPUs para acelerar as operações de convolução e geração de *embeddings*. Todo o código de treino e *scripts* de análise estatística permaneceram na camada de infraestrutura, segundo a Figura 22, assegurando modularidade e separação entre ambientes de desenvolvimento e produção.

Na camada de aplicação, foi implementado o *pipeline* de inferência, responsável pela orquestração das duas redes neurais — YOLOv11 para detecção e OSNet para reidentificação. Esse fluxo realiza a detecção inicial de pessoas em cada *frame*, recorta as regiões delimitadas (*bounding boxes*) e, em seguida, gera e compara os *embeddings* extraídos por OSNet com aqueles previamente cadastrados pelo usuário. Essa comparação utiliza medidas de similaridade vetorial (distância euclidiana) para determinar a identidade mais provável, conforme o apêndice C.

Para garantir a segurança e privacidade dos dados, todos os *embeddings* gerados são criptografados imediatamente após sua criação, utilizando o algoritmo AES-256-GCM. Na criação de Person, cada *embedding* recebe um *nonce* aleatório de 96 bits concatenado ao *ciphertext*, e o processo de descriptografia ocorre apenas no momento da comparação, evitando a persistência de dados sensíveis em memória não segura, como visto no apêndice D.

Por fim, a camada de interface foi implementada de forma a permitir tanto a comunicação via HTTP, para requisições pontuais de inferência, quanto por *WebSocket*, para fluxos contínuos de vídeo.

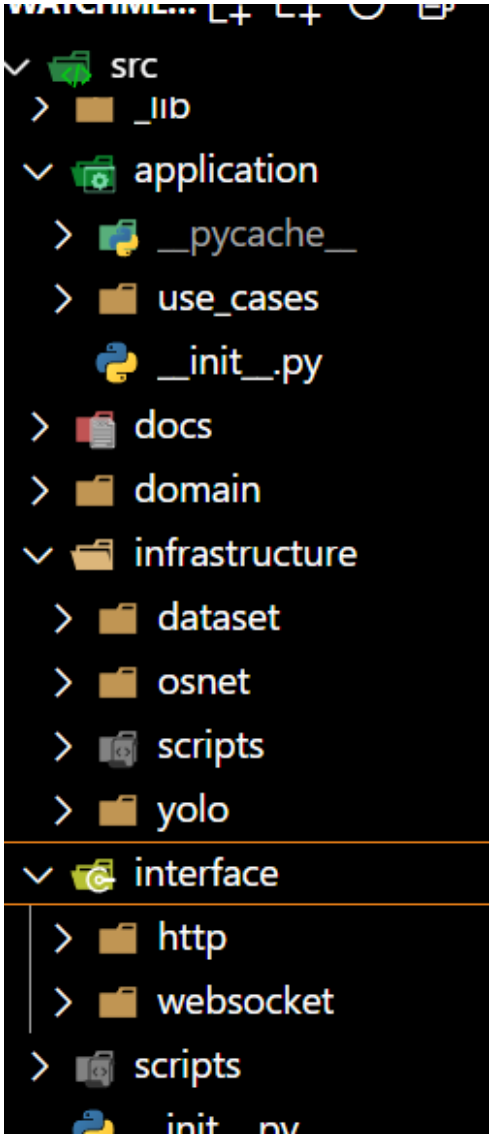

Figura 22 – Visualização da arquitetura *Domain Drive Design*.
Fonte: Autoria própria.

5.4 Documentação

A documentação técnica do sistema foi elaborada utilizando o *framework* Sphinx, que permite a geração de páginas estáticas a partir de arquivos no formato *reStructuredText*. Essa escolha foi motivada pela integração nativa com projetos em Python e pela capacidade de exportar documentação navegável, com estrutura hierárquica de módulos e classes.

O conteúdo da documentação abrange as principais etapas de desenvolvimento do sistema, incluindo:

- Estrutura e descrição dos *endpoints* da API, com exemplos de requisição e resposta;
- Instruções de execução e implantação do sistema em ambiente local ou containerizado;

- Detalhes sobre os modelos de inteligência artificial utilizados, incluindo parâmetros de inferência e formatos de entrada e saída.

A interface da documentação foi gerada com o tema *sphinx_rtd_theme*, compatível com o padrão *Read the Docs*, resultando em uma página responsiva e de fácil navegação. Além disso, foram integradas extensões para suporte a anotações de tipo do Python e blocos de código interativos, facilitando a leitura e reprodução dos exemplos.

A versão mais recente da documentação está hospedada de forma pública no GitHub Pages com GitHub Actions para rodar as atualizações futuras, e pode ser acessada em:

https://beatrizamante.github.io/watchme_ai

Essa estrutura visa não apenas descrever o funcionamento do sistema, mas também garantir sua reprodutibilidade e servir de base para futuras pesquisas ou colaborações.

6 CONSIDERAÇÕES FINAIS

Conclui-se que este projeto busca entregar uma solução funcional, acessível e escalável para o rastreamento de pessoas por vídeo com o uso de inteligência artificial, por meio do desenvolvimento de uma API *end-to-end*. A proposta se fundamenta na construção de uma ponte entre a complexidade técnica do reconhecimento visual baseado em redes neurais e a necessidade de interfaces simplificadas que viabilizem o uso por profissionais não especialistas em IA.

Especificamente, pretende-se disponibilizar uma API bem documentada, com *endpoints* REST organizados, exemplos de requisição e retorno em JSON, além de uma interface visual intuitiva, que permita o teste e a validação das funcionalidades sem exigir conhecimento avançado em programação ou aprendizado de máquina.

O projeto procura superar dificuldades recorrentes na adoção de tecnologias de rastreamento automatizado, como a ausência de ferramentas acessíveis, a falta de documentação clara em muitos *frameworks* existentes e a complexidade na integração entre modelos de detecção, rastreamento e reidentificação. Ao focar em modularidade, privacidade e facilidade de uso, o sistema proposto visa democratizar o acesso a essas tecnologias, viabilizando sua aplicação em contextos diversos — da segurança pública ao uso doméstico.

Além disso, espera-se que o protótipo desenvolvido desperte o interesse da comunidade acadêmica e técnica para novas aplicações e extensões do sistema, como a inclusão de múltiplos alvos, alertas em tempo real, integração com câmeras IP e *dashboards* analíticos para monitoramento contínuo. Em um cenário onde a privacidade dos dados é cada vez mais relevante, a proposta também levanta discussões importantes sobre ética e segurança, reforçando a necessidade de ferramentas tecnológicas aliadas a boas práticas de proteção da informação.

REFERÊNCIAS

AL-JABERY, K. K. *et al.* 3 - clustering algorithms. *In*: AL-JABERY, K. K. *et al.* (Ed.). **Computational Learning Approaches to Data Analytics in Biomedical Applications**. Academic Press, 2020. p. 29–100. ISBN 978-0-12-814482-4. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780128144824000036.

ALMASAWA, M.; ELREFAEI, L.; MORIA, K. A survey on deep learning based person re-identification systems. **IEEE Access**, PP, p. 1–1, 12 2019.

Amazon Web Services. **O que é uma Rede Generativa Adversária (GAN)?** 2023. Acesso em: 30 abr. 2025. Disponível em: https://aws.amazon.com/pt/what-is/gan/.

BERGMANN, P.; MEINHARDT, T.; LEAL-TAIXE, L. Tracking without bells and whistles. *In*: **2019 IEEE/CVF International Conference on Computer Vision (ICCV)**. IEEE, 2019. p. 941–951. Disponível em: http://dx.doi.org/10.1109/ICCV.2019.00103.

BERLYAND, L.; JABIN, P. **Mathematics of Deep Learning: An Introduction**. Berlin; Boston: De Gruyter, 2023. ISBN 9783111024318.

CHACON, S.; STRAUB, B. **Pro Git**. Apress, 2014. Disponível em: https://git-scm.com/book/en/v2.

CHEN, M. *et al.* Person re-identification by pose invariant deep metric learning with improved triplet loss. **IEEE Access**, PP, p. 1–1, 11 2018.

CORPORATION, N. **CUDA Toolkit Documentation**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://developer.nvidia.com/cuda-toolkit.

CORPORATION, N. **NVIDIA cuDNN: GPU Accelerated Deep Learning**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://developer.nvidia.com/cudnn.

DBDIAGRAM. **DBDiagram**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://dbdiagram.io/.

DOCKER. **Docker**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://www.docker.com/.

EVANS, E. **Domain-driven design: tackling complexity in the heart of software**. [*S.l.*]: Addison-Wesley Professional, 2003.

EXPO. **Expo**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://expo.dev/.

FETTE, I.; MELNIKOV, A. **The WebSocket Protocol**. 2011. RFC 6455. Disponível em: https://datatracker.ietf.org/doc/html/rfc6455.

FIELDING, R. T. **Architectural styles and the design of network-based software architectures**. 2000. Tese (Doutorado) — University of California, Irvine, 2000.

FIGMA. **Figma**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://www.figma.com/.

FRAMEWORK, F. W. **Fastify Web Framework**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://fastify.dev/.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. Upper Saddle River, N.J.: Prentice Hall, 2008. ISBN 9780131687288.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, MA: MIT Press, 2016. ISBN 9780262035613. Disponível em: https://www.deeplearningbook.org.

GOOGLE. **Google Colab Research**. 2025. Acesso em: 19 out. 2025. Disponível em: https://colab.google.

INFERENCE, Z. T. first schema validation with static type. **Zod - TypeScript-first schema validation with static type inference**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://zod.dev/.

JUNIOR, M. A. d. S.; MARTINI, J. S. C. **Utilização eficiente em larga escala de reconhecimento facial para análise preditiva de segurança em cidades inteligentes**. 2019. Dissertação (Mestrado) — Universidade de São Paulo, 2019.

KARMAKAR, A.; MISHRA, D. **Pose Invariant Person Re-Identification using Robust Pose-transformation GAN**. 2021. Disponível em: https://arxiv.org/abs/2105.00930.

KNEX.JS. **Knex.js**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://knexjs.org/.

LI, X. *et al.* Deep metric learning for few-shot image classification: A review of recent developments. **Pattern Recognition**, v. 138, p. 109381, 2023. ISSN 0031-3203. Disponível em: https://www.sciencedirect.com/science/article/pii/S0031320323000821.

LIANG, E. *et al.* RLlib: Abstractions for distributed reinforcement learning. *In*: **International Conference on Machine Learning (ICML)**. [*s.n.*], 2018. Disponível em: https://arxiv.org/pdf/1712.09381.

LIN, T.-Y. *et al.* Microsoft coco: Common objects in context. *In*: SPRINGER. **European conference on computer vision (ECCV)**. [*S.l.*], 2014. p. 740–755.

MCLAUGHLIN, N.; RINCON, J. Martinez del; MILLER, P. Recurrent convolutional network for video-based person re-identification. *In*: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [*S.l.*: *s.n.*], 2016. p. 1325–1334.

MELO, P. V.; SERRA, P. Tecnologia de reconhecimento facial e segurança pública nas capitais brasileiras: Apontamentos e problematizações. **Comunicação e Sociedade**, n. 42, 2022. Postado online em 16 dez. 2022. Acesso em: 30 abr. 2025. Disponível em: http://journals.openedition.org/cs/8111.

NATIVE, R. **React Native**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://reactnative.dev/.

NATIVEWIND. **NativeWind**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://www.nativewind.dev/.

NOTION. **Notion**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://www.notion.so/.

NUMPY. **NumPy**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://numpy.org/.

OBJECTION.JS. **Objection.js**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://vincit.github.io/objection.js/.

OPENAPI. **OpenAPI Specification (Swagger)**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://swagger.io/specification/.

OPENCV. **OpenCV**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://opencv.org/.

POSTGRESQL. **PostgreSQL**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://www.postgresql.org/.

Python Software Foundation. **Python Programming Language**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://www.python.org/.

PYTORCH. **PyTorch**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://pytorch.org/.

RAMACHANDRAN, G. R. Modern api design: Ai-first architecture, event-driven patterns, and zero-trust security. **International Journal of Computer Trends and Technology**, v. 72, n. 11, p. 220–227, 2024.

RAMíREZ, S. **FastAPI: Modern, fast (high-performance), web framework for building APIs with Python 3.7+**. 2023. Acesso em: 30 abr. 2025. Disponível em: https://fastapi.tiangolo.com.

REDMON, J. *et al.* You only look once: Unified, real-time object detection. *In*: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [*S.l.*: *s.n.*], 2016.

SCIPY. **SciPy**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://scipy.org/.

SILVA, M. L. S. **As tecnologias de reconhecimento facial para Segurança Pública no Brasil: perspectivas regulatórias e a garantia de Direitos Fundamentais**. 2022. Monografia (Graduação em Direito) - Universidade Federal do Rio Grande do Norte, Natal, 87f.

SPHINX. **Sphinx Documentation Generator**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://www.sphinx-doc.org/.

SRIVASTAVA, N. *et al.* Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 56, p. 1929–1958, 2014. Disponível em: http://jmlr.org/papers/v15/srivastava14a.html.

SUN, Y. *et al.* Learning part-based convolutional features for person re-identification. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 43, n. 3, p. 902–917, 2021.

TORCHREID. **Torchreid: Person Re-identification Framework**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://kaiyangzhou.github.io/deep-person-reid/.

TYPESCRIPT. **TypeScript**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://www.typescriptlang.org/.

ULTRALYTICS. **YOLOv11 Documentation**. 2024. Acesso em: Junho de 2025. Disponível em: https://docs.ultralytics.com/.

ULTRALYTICS. **Ultralytics YOLO**. 2025. Acesso em: 17 jun. 2025. Disponível em: https://github.com/ultralytics/ultralytics.

WEI, Y. *et al.* Deep learning for retail product recognition: Challenges and techniques. **Computational Intelligence and Neuroscience**, v. 2020, p. 8875910, 2020.

WOJKE, N.; BEWLEY, A.; PAULUS, D. Simple online and realtime tracking with a deep association metric. *In*: **2017 IEEE International Conference on Image Processing (ICIP)**. IEEE Press, 2017. p. 3645–3649. Disponível em: https://doi.org/10.1109/ICIP.2017.8296962.

WU, J. *et al.* **Segment Anything Model is a Good Teacher for Local Feature Learning**. 2024. Disponível em: https://arxiv.org/abs/2309.16992.

YADAV, S. K. *et al.* Yognet: A two-stream network for realtime multiperson yoga action recognition and posture correction. **Knowledge-Based Systems**, v. 250, p. 109097, 2022. ISSN 0950-7051. Disponível em: https://www.sciencedirect.com/science/article/pii/S095070512200541X.

ZHENG, L. *et al.* Scalable person re-identification: A benchmark. *In*: **2015 IEEE International Conference on Computer Vision (ICCV)**. [*S.l.*: *s.n.*], 2015. p. 1116–1124.

ZHENG, Z.; ZHENG, L.; YANG, Y. Unlabeled samples generated by gan improve the person re-identification baseline in vitro. **arXiv preprint arXiv:1701.07717**, 2017.

ZHOU, K. *et al.* Omni-scale feature learning for person re-identification. *In*: **Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)**. [*s.n.*], 2019. p. 3707–3716. Disponível em: https://arxiv.org/abs/1905.00953.

GLOSSÁRIO

backend parte lógica e de processamento de uma aplicação, responsável por regras de negócio, banco de dados e comunicação com o frontend. 59

benchmark referência usada para medir ou comparar o desempenho de um produto, processo ou investimento. 29, 30, 47

cookie pequeno arquivo armazenado no navegador que contém informações sobre o usuário ou sessão de um site. 40, 59

dashboard painel visual que apresenta dados e métricas em tempo real de forma organizada e interativa. 9, 26, 54

dataset coleção organizada de dados, geralmente apresentada em formato de tabela com linhas e colunas, que serve para análise, pesquisa e treinamento de modelos de inteligência artificial. 15, 29, 30

embedding representação numérica de dados complexos, como palavras ou imagens, em um espaço vetorial que facilita o processamento por modelos de aprendizado de máquina. 8, 11, 19–22, 26, 28, 30, 31, 33–35, 37, 38, 40–45, 50, 51

end-to-end abordagem em que um sistema executa todo o processo de entrada até a saída final de forma automatizada, sem necessidade de etapas manuais intermediárias. 24, 26, 54

endpoint ponto de acesso em uma API que define uma rota específica para comunicação entre cliente e servidor. 28, 29, 37, 38, 40, 45, 52, 54

feature característica ou atributo mensurável usado como entrada em modelos de aprendizado de máquina. 4, 8, 13, 20

framework estrutura ou conjunto de ferramentas e bibliotecas que fornecem uma base reutilizável para o desenvolvimento de aplicações de software. 8, 11, 27, 28, 30, 31, 33, 37, 45, 47, 52, 54

frontend parte visual e interativa de uma aplicação, com a qual o usuário final interage diretamente. 28, 41, 59

http-only atributo de segurança de cookie que impede o acesso ao seu conteúdo via JavaScript, reduzindo o risco de ataques de script. 40

middleware camada intermediária entre o frontend e o backend, usada para processar requisições, autenticação ou integração de serviços. 45

pipeline sequência de etapas que um processo deve seguir para atingir um objetivo, seja em vendas, dados, programação ou hardware. 18, 31, 51

role função ou conjunto de permissões atribuídas a um usuário em um sistema de controle de acesso. 41, 44

script conjunto de instruções executadas por um interpretador para automatizar tarefas em um sistema ou aplicação. 37, 51, 59

software conjunto de programas e instruções que controlam o funcionamento de um computador ou realizam tarefas específicas. 23, 37, 59

streaming transmissão contínua de dados, como áudio ou vídeo, permitindo a reprodução imediata sem necessidade de download completo. 27

timestamp registro temporal que indica o momento exato em que um evento ocorreu, geralmente em segundos desde uma data de referência. 37, 40

token sequência de caracteres usada para autenticação, autorização ou representação de dados em sistemas computacionais. 29, 40

upload processo de enviar arquivos ou dados de um dispositivo local para um servidor remoto ou serviço online. 9, 27, 34, 42

APÊNDICES

APÊNDICE A – Trechos de Códigos para arquitetura *Domain Driven Design* em TypeScript

Entidade de Domínio *Person*

```
1
2  export class Person {
3    public readonly id?: number;
4    public readonly user_id: number;
5    public readonly name: string;
6    public readonly embedding: Buffer;
7    //code continuation...
8  }
```

Exemplo da Camada de Aplicação para o caso de uso de *Person*

```
1  export const makeCreatePerson =
2    ({ personRepository }: Dependencies) =>
3    async ({ person }: CreatePersonParams) => {
4      const validPerson = new Person(person);
5
6      const newPerson = await personRepository.create(validPerson);
7      return PersonSerializer.serialize(newPerson);
8    };
```

Exemplo da Camada de Interface para manejo de respostas e requisições de *Person*

```
1  export const personController = {
2    create: async (request: FastifyRequest, reply: FastifyReply) => {
3      const { bodyData, file } = await multiformFilter(parts);
4      const parseResult = CreatePersonInput.safeParse(bodyData);
5      const { createPerson } = createRequestScopedContainer();
6
7      if (!parseResult.success) {
8        return reply.status(400).send({
9          error: "Invalid input",
10         details: parseResult.error.issues,
11       });
12     }
13
14     if (!file) {
15       throw new InvalidPersonError({
16         message: "To find a person, you need to add a picture",
17       });
18     }
```

```
19
20      fileSizePolicy({ file });
21
22      //... call AI api...
23
24      const result = await createPerson({
25        person: {
26          name: parseResult.data.name,
27          user_id: userId,
28          embedding,
29        },
30      });
31
32      return reply.status(201).send(result);
33    }
34  }
```

APÊNDICE B – Trecho de código em Python para Predição de *Person* por *Embedding*

Exemplo do *handle* do domínio *Person* em Python, sem persistência de dados

```python
@router.post("/upload-embedding")
async def upload_person_image(request: ImageModel):
    """
    Upload an image and get the person embedding.
    """
    print(f"Received base64 image")

    try:
        image_bytes = base64.b64decode(request.image)

        nparr = np.frombuffer(image_bytes, np.uint8)
        image = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

        if image is None:
            raise ValueError("Could not decode image file")

        embedding = create_person_embedding(image)

        embedding_b64 = base64.b64encode(embedding.tobytes()).decode('utf-8')

        return {
            "embedding": embedding_b64,
            "shape": list(embedding.shape),
            "dtype": str(embedding.dtype),
            "status": "success"
        }

def create_person_embedding(file):
    encode = OSNetEncoder()
    person_bbox_list = predict(file)

    if not person_bbox_list or not person_bbox_list[0]['detections']:
        raise ValueError("No person detected, please try with another image")

    first_detection = person_bbox_list[0]['detections'][0]
    cropped_image = first_detection['cropped_image']
    try:
        encoding = encode.encode_single_image(cropped_image)

        if encoding is None or encoding.size == 0:
```

```
41              raise ValueError("Failed to generate person embedding")
42
43          encrypted_embedding = encrypt_embedding(encoding)
44
45          return encrypted_embedding
```

APÊNDICE C – Trecho de *Script* para calculo de distância euclidiana

```python
import numpy as np

def compute_euclidean_distance(embedding1, embedding2):
    return float(np.linalg.norm(embedding1 - embedding2))

def compute_batch_distances(target_embedding, candidate_embeddings):
    if target_embedding.ndim != 1:
        target_embedding = target_embedding.flatten()

    if candidate_embeddings.ndim == 1:
        candidate_embeddings = candidate_embeddings.reshape(1, -1)

    distances = np.linalg.norm(candidate_embeddings - target_embedding, axis=1)
    return distances
```

APÊNDICE D – Trecho de código encriptação de dados

Encriptação

```
def encrypt_embedding(embedding: np.ndarray) -> bytes:
    data = embedding.tobytes()
    cipher = AES.new(key_setting.key_bytes, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(data)
    return cipher.nonce + tag + ciphertext
```

Desencriptação

```
def decrypt_embedding(encrypted: bytes, shape, dtype) -> np.ndarray:
    nonce = encrypted[:16]
    tag = encrypted[16:32]
    ciphertext = encrypted[32:]
    cipher = AES.new(key_setting.key_bytes, AES.MODE_EAX, nonce=nonce)
    data = cipher.decrypt_and_verify(ciphertext, tag)
    return np.frombuffer(data, dtype=dtype).reshape(shape)
```

Uso em criação

```
#... begin...
    encoding = encode.encode_single_image(cropped_image)

    if encoding is None or encoding.size == 0:
        raise ValueError("Failed to generate person embedding")

    encrypted_embedding = encrypt_embedding(encoding)

    return encrypted_embedding
#... continuation...
```

Desencriptado em predição

```
#... begin
for frame_result in people_results:
        for detection in frame_result['detections']:
            all_cropped_images.append(detection['cropped_image'])
            all_bboxes.append(detection['bbox'])
```

```
6
7       if not all_cropped_images:
8           return []
9
10      decrypted_embedding = decrypt_embedding(chosen_person, shape=(512,), dtype='float32
11
12      encoded_batch = encoder.encode_batch(all_cropped_images)
13      matches = []
14  for i, encoded_person in enumerate(encoded_batch):
15          distance = compute_euclidean_distance(decrypted_embedding, encoded_person)
16          if distance < 0.8:
17              matches.append({
18                  "bbox": all_bboxes[i],
19                  "distance": distance
20              })
21
22      return matches
23      #Never returns the decrypted embedding, just the coordinates and distance
```

ANEXO A – Checagem de Qualidade

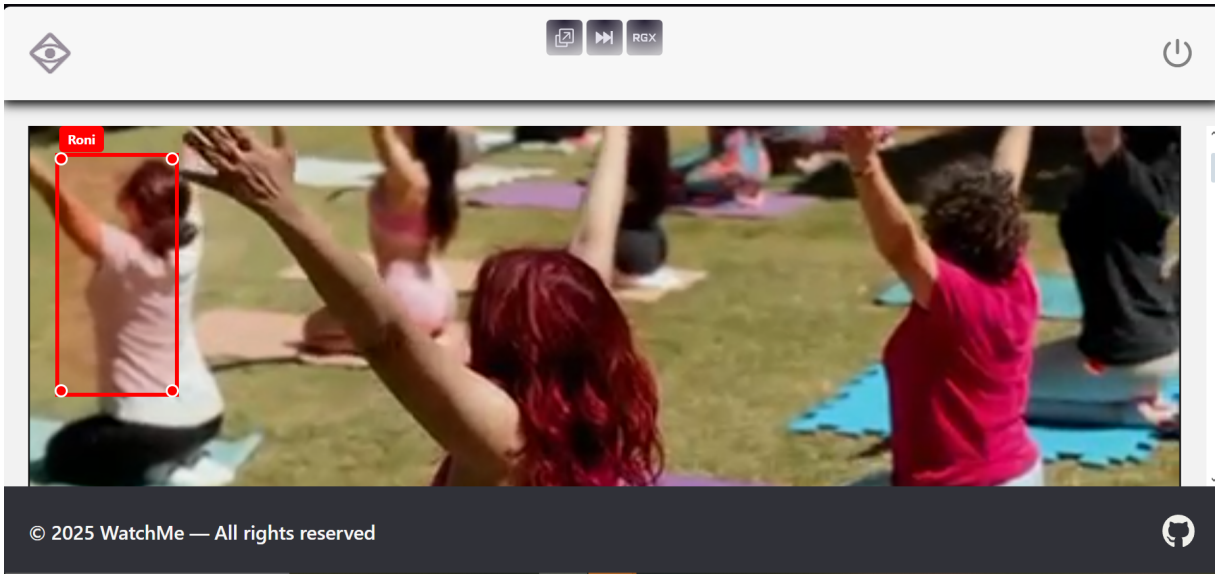

Figura 23 – Imagem da *bounding box* desenhada na pessoa a ser checada.
Fonte: Autoria própria.

ANEXO B – Checagem de Qualidade

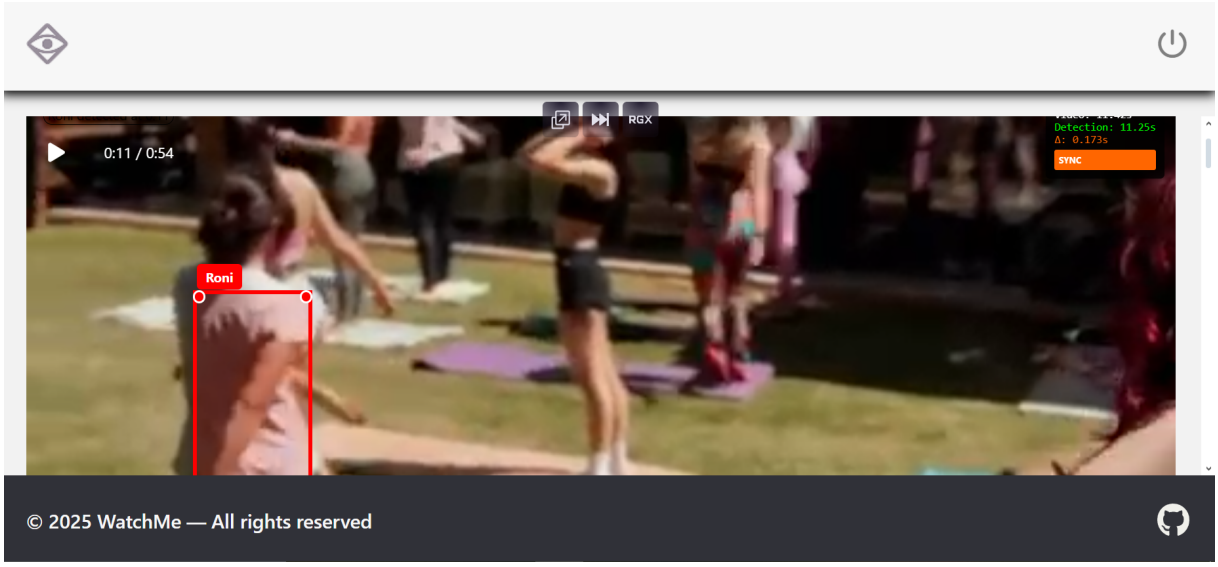

Figura 24 – Imagem da *bounding box* desenhada na pessoa a ser checada.
Fonte: Autoria própria.