UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA - UESB

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LARA SANTOS PEREIRA SOARES

Sistema de Reconhecimento e Autenticação Facial utilizando Rede Neural Convolucional

VITÓRIA DA CONQUISTA - BA
2025

LARA SANTOS PEREIRA SOARES

Sistema de Reconhecimento e Autenticação Facial utilizando Rede Neural Convolucional

Trabalho de Conclusão de Curso apresentado ao curso de Ciência da Computação da Universidade Estadual do Sudoeste da Bahia - UESB como requisito parcial para a obtenção do título de Bacharelado em Cientista da Computação.

Orientador: Prof. Roque Mendes Prado Trindade

VITÓRIA DA CONQUISTA - BA
2025

Dedico este trabalho a minha querida irmã Helena, presente de Deus em minha vida.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, por iluminar e abençoar a minha vida.

Aos meus pais, Maria Rita e Gilberto, pelo amor, carinho, educação e valores que me permitiram seguir em frente.

Ao meu tio Allan, por compartilhar generosamente seu conhecimento e me auxiliar nos momentos necessários.

Aos professores, pela dedicação, expertise e compromisso em transmitir conhecimento, fundamentais para o meu crescimento acadêmico.

E, em especial, ao meu orientador, Professor Roque Mendes Prado Trindade, pela paciência, empenho e relevância de suas sugestões e correções.

À minha amiga Celina, pelo afeto demonstrado e pelas constantes palavras de incentivo.

Ao meu amigo Gustavo, pela colaboração inestimável e pela generosidade em compartilhar seus conhecimentos.

Aos colegas de curso, pelo convívio colaborativo, pelas reflexões compartilhadas e pelo desenvolvimento mútuo que engrandeceu nossa trajetória acadêmica.

Por fim, agradeço a todos que, de alguma forma, contribuíram para que eu alcançasse mais esta importante conquista.

"O propósito da IA não é criar uma mente, mas estender a capacidade cognitiva do homem.".

EDSGER DIJKSTRA

RESUMO

O presente trabalho tem como objetivo o desenvolvimento de um sistema de autenticação baseado em reconhecimento facial, visando aprimorar o controle de acesso a ambientes físicos. Para alcançar esse propósito, foi realizada uma revisão da literatura sobre redes neurais artificiais, com ênfase em arquiteturas aplicadas ao reconhecimento facial e suas principais aplicações em sistemas de autenticação. Diferentes modelos de redes neurais convolucionais foram analisados e comparados quanto à eficiência, precisão e robustez. A partir dos testes e experimentações conduzidos, foi possível avaliar o desempenho das arquiteturas estudadas e determinar aquela mais adequada à resolução do problema proposto. Por fim, o sistema de controle de acesso e autenticação foi implementado com base na arquitetura selecionada, destacando-se pela utilização de tecnologias de visão computacional e aprendizado de máquina, proporcionando uma solução eficiente e escalável para ambientes que requerem maior nível de segurança.

Palavras-chave: Inteligência Artificial. Reconhecimento e Autenticação facial. Controle de Acesso.

ABSTRACT

This work presents the development of an authentication system based on facial recognition, designed to improve physical access control. To this end, a literature review of artificial neural networks was performed, focusing on architectures applied to facial recognition and their primary applications in authentication systems. Various convolutional neural network models were analyzed and compared in terms of their efficiency, accuracy, and robustness. Through conducted tests and experiments, the performance of the studied architectures was evaluated to identify the most suitable solution for the proposed problem. Ultimately, the access control and authentication system was implemented using the selected architecture. The system leverages computer vision and machine learning technologies to provide an efficient and scalable solution for environments requiring a higher level of security.

Keywords: Artificial Intelligence. Facial Recognition and Authentication. Access Control.

SUMÁRIO

1. Introdução

Este capítulo estabelece a fundamentação do projeto, partindo da crescente vulnerabilidade dos sistemas de autenticação tradicionais. A contextualização do problema é evidenciada por meio de exemplos concretos e recentes de falhas de segurança, como fraudes em plataformas governamentais (gov.br) e vazamentos de dados em grandes corporações, que demonstram a insuficiência das medidas atuais. Diante desse cenário de insegurança, o trabalho propõe o desenvolvimento de um sistema de autenticação robusto, fundamentado em reconhecimento facial. A relevância desta tecnologia é corroborada pelo estado da arte, que a aponta como uma solução estratégica e consolidada para o controle de acesso seguro.

Para alcançar este propósito, o capítulo detalha os objetivos do trabalho, que englobam desde a revisão da literatura sobre Redes Neurais e reconhecimento facial até a implementação e avaliação de um modelo específico. A pesquisa se apoia na metodologia Projeto de Pesquisa Científica, uma abordagem focada na criação de um artefato tecnológico — neste caso, o sistema de autenticação — para solucionar um problema prático. Por fim, são delineadas as etapas do projeto, que incluem a análise de trabalhos relacionados, a seleção e o treinamento de um modelo de rede neural, sua implementação prática e, por fim, a avaliação de seu desempenho, estabelecendo um roteiro claro para a construção da solução proposta.

1.1 Motivação e Contextualização do Problema

Conforme elucidado pelo GRUPO SIDE ([s.d.]), o controle de acessos constitui uma atividade primordialmente voltada à gestão e monitoramento meticuloso da entrada e saída de indivíduos, veículos ou objetos em um determinado local, bem como ao gerenciamento de dispositivos em servidores, garantindo que apenas sujeitos devidamente autorizados possam transitar em áreas que demandam restrição de acesso.

A vulnerabilidade de sistemas de autenticação, mesmo os que utilizam biometria, é constantemente demonstrada por notícias de fraudes bem-sucedidas em plataformas digitais no Brasil. A Polícia Federal revela um exemplo emblemático dessa vulnerabilidade ao desarticular um esquema de fraude na plataforma gov.br — o sistema unificado de acesso aos serviços digitais do governo federal. Conforme noticiado pelo portal G1 (2023):

> O esquema criminoso logrou êxito ao fraudar especificamente o mecanismo de "*liveness*", utilizado para verificar se a imagem capturada pertence a uma pessoa real e viva. Ao burlar essa verificação, os fraudadores conseguiam

> simular os traços faciais de terceiros para acessar indevidamente as contas digitais das vítimas. A quadrilha atuava na violação de contas de pessoas falecidas para sacar valores a receber do Banco Central e no acesso a contas de pessoas vivas para autorizar consignações fraudulentas no aplicativo "Meu INSS". A investigação inicial apontou que pelo menos três mil contas foram acessadas, mas a Polícia Federal suspeita que a extensão da fraude seja consideravelmente maior.

Além disso, a fragilidade da segurança se evidencia em diversos casos de invasões e vazamentos de dados confidenciais no Brasil. Em 2024, a empresa XP informou uma parcela de sua base de clientes acerca de um vazamento de dados pessoais de proporções significativas. Tais informações comprometidas incluíam dados sensíveis como nome completo, número de telefone, endereço de e-mail, data de nascimento, Código de Endereçamento Postal (CEP), estado civil, gênero, cargo profissional e nacionalidade, evidenciando a vulnerabilidade sistêmica a que dados críticos estão expostos em diversas esferas.

A recorrência desses eventos no território nacional ressalta a insuficiência das medidas de segurança atualmente empregadas e a urgente reavaliação das estratégias de proteção de dados e infraestrutura. A implementação de protocolos de autenticação multifator, o uso de tecnologias biométricas avançadas e o desenvolvimento contínuo de políticas de segurança da informação adaptadas às novas ameaças digitais tornam-se imperativos.

1.2 Estado da Arte

Entre as soluções tecnológicas atualmente disponíveis para o controle de acesso e autenticidade, destacam-se métodos que aliam eficiência, segurança e praticidade. Segundo GUIMARÃES (2025), com o avanço significativo das tecnologias aplicadas à segurança, os sistemas de reconhecimento facial têm se consolidado como uma abordagem robusta para o controle de acesso a ambientes físicos e digitais. Essa tecnologia possibilita a identificação automática de indivíduos com elevada precisão, reduzindo significativamente a suscetibilidade às fraudes e acessos não autorizados. Tais características tornam o reconhecimento facial uma solução estratégica para ambientes que demandam alto nível de segurança.

De acordo com ISHIDA (2023), o avanço das tecnologias de segurança tem proporcionado níveis cada vez mais elevados de proteção em diferentes setores da sociedade, incluindo a utilização de biometria em transações bancárias, a verificação de identidade em dispositivos móveis e, de forma destacada, o reconhecimento facial. Essa

última tecnologia tem sido amplamente empregada no controle de acesso a ambientes como edifícios corporativos, condomínios residenciais, instituições educacionais e empresas em geral.

A restrição de acesso em áreas de aglomeração populacional representa um fator crítico para a eficácia operacional de diversas organizações, sejam elas públicas ou privadas. Nesse contexto, consoante ARRUDA (2020), a visão computacional emerge como uma tecnologia capaz de habilitar máquinas a interpretar e extrair características relevantes de imagens capturadas por sensores variados, permitindo o reconhecimento, manipulação e processamento de informações sobre os objetos presentes nas cenas.

1.3 Trabalhos Relacionados

Arquiteturas neurais profundas estão cada vez mais sendo empregadas para aprimorar modelos em reconhecimento de imagem e outras áreas, gerando um crescente interesse em engenharia de arquitetura especializada no *Deep Learning*. A automação do design de redes neurais profundas tem avançado, com foco em tarefas complexas como a restauração de imagens de super-resolução (SR). O estudo de GARCÍA, MONROY e HERNÁNDEZ (2024) explora abordagens modernas para o design automático de redes neurais, examina a evolução das técnicas de busca de arquitetura para restauração de imagens e apresenta um protocolo experimental para uma comparação justa de modelos de SR.

Um estudo relevante para esta pesquisa foi conduzido por LEMOS et al. (2024), que descreve o desenvolvimento de um sistema baseado em ESP32Cam, ESP32 WROOM, tela Transistor de Película Fina (*Thin-Film Transistor* – TFT) e fecho eletromagnético, permitindo a autenticação facial em tempo real e o monitoramento remoto de acessos.

Em diversos contextos, o controle de acesso a salas e laboratórios em ambientes restritos apresenta custos elevados, decorrentes do uso de fichas, chaves físicas e da necessidade de presença de pessoal para gerenciamento de entradas e saídas. O trabalho desenvolvido por BORGES et al. (2023) propõe uma solução baseada em dispositivos Internet das Coisas (IOT) operando de forma totalmente autônoma, eliminando a exigência de que usuários portem dispositivos ou acessórios de autenticação — a simples presença do indivíduo no local é suficiente para a identificação. Além disso, o sistema realiza a geração automatizada de relatórios de acesso. A arquitetura proposta é composta por quatro módulos principais de IOT, integrados como resposta eficiente ao problema

identificado. Em sua fase de testes, o sistema demonstrou alto desempenho e capacidade para suprir as demandas de um controle de acesso autônomo e seguro.

O reconhecimento facial humano é um tema de pesquisa amplamente estudado, em virtude da sua notável eficiência e precisão, com ampla aplicação em diversas indústrias. LIU et al. (2022) analisam neste artigo os recursos técnicos, as aplicações e os algoritmos comuns de reconhecimento facial, destacando suas vantagens e desvantagens. A revisão concentra-se na aplicação dessa tecnologia no setor médico e discute as perspectivas para seu desenvolvimento futuro.

O estudo de SINGH et al. (2022) busca desenvolver um sistema avançado de reconhecimento facial para diversas aplicações, superando as limitações das técnicas atuais, como baixa precisão e alta taxa de falsos alarmes. Dada a crescente preocupação com a segurança e o avanço dos dispositivos móveis, a detecção facial tem se tornado um campo de pesquisa em expansão. O artigo se concentra na implementação de um sistema de reconhecimento facial usando a biblioteca *OpenCV* com *Python*, enfatizando suas aplicações atuais e visando aprimorar a precisão e o tempo de reconhecimento.

Outro estudo relevante para esta pesquisa foi realizado por ANWARUL e DAHIYA (2019). O artigo oferece uma visão detalhada de diversas estratégias essenciais para enfrentar problemas de reconhecimento facial, destacando a precisão dessas abordagens e os fatores que afetam negativamente o desempenho dos métodos analisados.

O trabalho elaborado por SILVA et al. (2019) investiga como o uso de reconhecimento facial na segurança pública pode reforçar o racismo estrutural no Brasil. Com base em métodos dedutivo, histórico e bibliográfico, conclui-se que a tecnologia apresenta falhas que afetam principalmente a população negra, exigindo transparência e cuidados jurídicos para evitar discriminação algorítmica.

O artigo de MAIA e TRINDADE (2016) descreve a implementação, em *Matlab*, de algoritmos para detecção e reconhecimento facial em imagens coloridas. O método utiliza uma rede neural *feedforward* para segmentação de pele, seguida pela detecção de olhos e lábios, alinhamento facial e delimitação da face. O reconhecimento facial é realizado por um classificador que compara matrizes de cores das faces. Testado no conjunto de dados *Faces* 1999, o sistema alcançou taxas de detecção de 96,9% para faces, 89% para olhos e 94% para lábios, além de uma taxa de reconhecimento facial de 70,7%.

1.4 Objetivos

1.4.1 Objetivo Geral

Desenvolver um sistema de autenticação e controle de acesso utilizando reconhecimento facial.

1.4.2 Objetivos Específicos

- Realizar a revisão da literatura relacionada a redes neurais, reconhecimento facial e suas aplicações em sistemas de autenticidade.
- Comparar diferentes modelos de rede neural convolucional aplicados ao reconhecimento facial.
- Avaliar os resultados obtidos nos testes e experimentações.
- Determinar o modelo de rede neural convolucional mais adequada para a resolução do problema proposto.
- Implementar o sistema de controle de acesso e autenticação baseado no modelo selecionado.

1.5. Metodologia

A metodologia adotada neste trabalho fundamenta-se no Projeto de Pesquisa Científica, em inglês *Design Science Research* (DSR), abordagem orientada à concepção e avaliação de artefatos que oferecem soluções práticas para problemas complexos. Em contraste com os métodos tradicionais das ciências naturais, que privilegiam a observação e a experimentação teórica, o DSR enfatiza a criação, implementação e validação de artefatos como instrumentos para a geração de conhecimento inovador.

Conforme BAX (2015), “A DSR envolve construir, investigar, validar e avaliar artefatos, tais como construtos, arcabouços, modelos, métodos e instâncias de sistema de informações, a fim de resolver novos problemas práticos”.

Por meio da DSR, é possível desenvolver um artefato voltado para controle de acesso utilizando reconhecimento facial. Esse processo é dividido nas seguintes etapas principais:

A primeira etapa é voltada para a revisão bibliográfica e estado da arte em Redes Neurais e reconhecimento facial, com levantamento sistemático de artigos, livros de referência e portais especializados, a fim de consolidar terminologias, arquiteturas clássicas e os avanços.

A segunda etapa envolve a coleta e preparação de dados para treinamento e validação, reunindo imagens de *datasets* públicos, aplicando detecção de faces,

alinhamento, normalização de intensidade e técnicas de aumento de dados para garantir representatividade e robustez.

Na terceira etapa é escolhido e configurado o modelo de rede neural convolucional mais adequado, a partir de critérios de desempenho (acurácia, perda e tempo computacional).

A quarta etapa consiste na implementação e integração da rede selecionada ao sistema de controle de acesso.

2. Fundamentos Teóricos

Este capítulo estabelece os fundamentos teóricos para o uso do reconhecimento facial em controle de acesso. A análise parte da Inteligência Artificial, contrastando as abordagens simbólica e conexionista e abordando o recente renascimento da área, impulsionado pelo aumento da capacidade computacional e de dados. Em seguida, o capítulo, detalha as arquiteturas de redes neurais essenciais, que são a base para o processamento de imagem e a visão computacional. Por fim, o capítulo conclui analisando como essas tecnologias impactam os modernos sistemas de controle de acesso, revolucionando a segurança e a autenticação de identidades.

2.1 Paradigmas da Inteligência Artificial

A história da Inteligência Artificial (IA) é marcada pela coexistência e competição de duas abordagens filosóficas e técnicas distintas: a IA Simbólica, que busca replicar a inteligência humana por meio da manipulação de símbolos e regras lógicas formais, e a IA Conexionista, que se inspira na estrutura do cérebro para criar sistemas que aprendem padrões diretamente a partir dos dados. Esses dois paradigmas, com suas premissas e métodos fundamentalmente diferentes, moldaram a evolução do campo e definem as bases para as tecnologias atuais de aprendizado de máquina.

2.1.1. Inteligência Artificial Simbólica

A IA simbólica, ou IA baseada em regras, tem como objetivo modelar o pensamento humano utilizando símbolos e regras lógicas. Conforme MARIUTTI (2023), a IA simbólica baseia-se na ideia de que a inteligência consiste na habilidade de manipular símbolos que representam o mundo e as diversas experiências que vivenciamos. Essa manipulação é efetuada por meio de inferências lógicas, as quais são derivadas de um conjunto de regras formais que delimitam os critérios de aceitabilidade.

No entanto, a IA simbólica demonstra limitações significativas, a complexidade do mundo real e a dificuldade de representar o conhecimento de maneira completa e precisa, limitaram seu progresso. Além disso, essa abordagem é vulnerável, pois pequenas alterações nas regras podem resultar em comportamentos inesperados e imprevisíveis, comprometendo sua eficácia.

2.1.2. A Inteligência Artificial Conexionista

Enquanto a IA simbólica busca replicar o raciocínio humano por meio de processos lógicos e regras explícitas, a IA conexionista inspira-se no funcionamento do cérebro humano, com foco no uso de redes neurais artificiais. Tais redes são constituídas por unidades interconectadas, que processam informações de maneira paralela e são capazes de aprender por meio de exemplos.

De acordo com SICHMAN (2024), o paradigma conexionista modela a linguagem a partir de redes compostas por elementos simples, em analogia à estrutura neural do cérebro humano. Nessa abordagem, os neurônios artificiais conectados entre si aprendem e generalizam padrões com base em exemplos, por meio da aproximação de funções via regressão não linear.

A IA conexionista demonstrou-se significativamente mais eficaz em tarefas que demandam aprendizado e reconhecimento de padrões, como visão computacional e processamento de linguagem natural. No entanto, por um longo período, o desenvolvimento dessa abordagem foi limitado pela insuficiência de poder computacional e pela elevada complexidade das redes neurais.

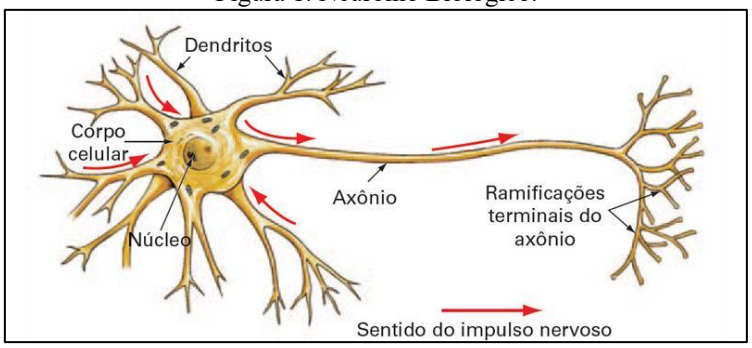
2.1.3. Renascimento da Inteligência Artificial

Após um rápido desenvolvimento nas décadas de 1950 e 1960, a IA entrou em um extenso período de inércia, conhecido como "inverno da IA". De acordo com COELHO (2023), diversos fatores contribuíram para esses períodos de estagnação, dentre eles, destacam-se as limitações tecnológicas da época, que restringiam o processamento de dados e a capacidade de armazenamento, essenciais para o progresso da IA. Além disso, a dificuldade de criar sistemas de IA que pudessem enfrentar tarefas do mundo real foi subestimada, resultando em promessas não cumpridas e, consequentemente, em uma perda de confiança e interesse.

2.2. Redes Neurais Artificiais

As redes neurais artificiais (RNA), são um subconjunto de Aprendizagem de Máquina (AM) e estão no cerne dos algoritmos de Aprendizagem Profundo (*Deep Learning)*. Conforme FLECK (2016), as RNA são algoritmos computacionais baseados em modelos matemáticos inspirados na estrutura de organismos inteligentes, permitindo a representação simplificada do funcionamento do cérebro humano em sistemas computacionais, como mostrado na figura 1.

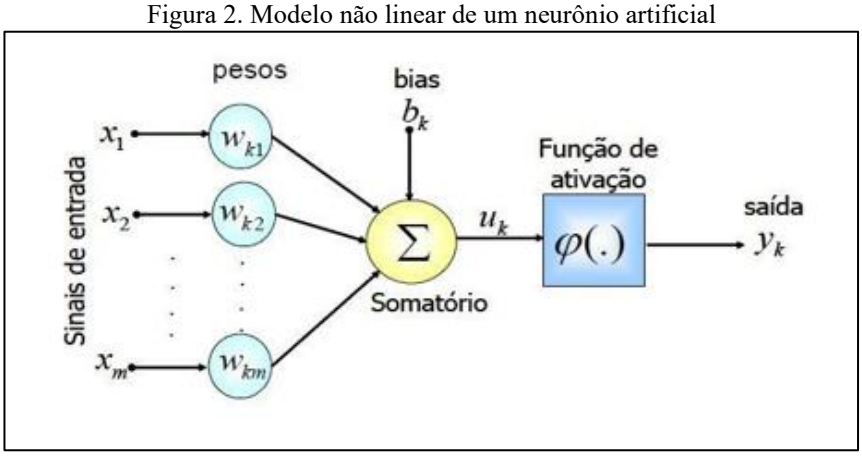
Figura 1. Neurônio Biológico.



Fonte: DEEP LEARNING BOOK, [s.d.].

Uma rede neural é composta por três componentes fundamentais. O primeiro desses elementos é o "neurônio", representado na figura 2. Os neurônios são as unidades computacionais básicas da rede, responsáveis por processar informações e transmiti-las entre as diferentes camadas da estrutura neural.

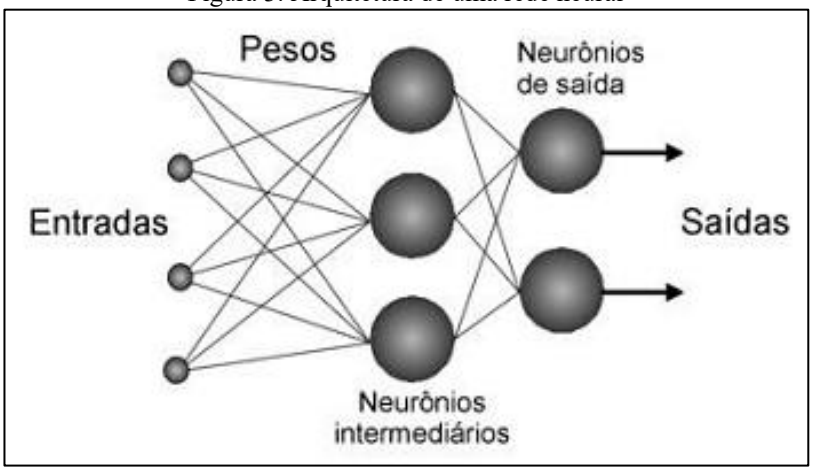
Figura 2. Modelo não linear de um neurônio artificial



Fonte: SOARES e DA SILVA, 2011.

O segundo componente é a "arquitetura" da rede, representada na figura 3. Essa arquitetura refere-se à estrutura topológica que determina a forma como os neurônios estão interconectados, estabelecendo o fluxo dentro da rede neural.

Figura 3. Arquitetura de uma rede neural



Fonte: COSTA, Bruno Carlos da C. et al , 2009.

Por fim, o terceiro elemento é o processo de "aprendizagem". A aprendizagem desempenha um papel crucial na capacidade da rede neural se adaptar para realizar tarefas específicas. Conforme destacado por HAYKIN (2001), as redes neurais apresentam semelhanças com o cérebro humano, pois adquirem conhecimento a partir da interação com o ambiente por meio do processo de aprendizagem, e armazenam esse conhecimento por meio dos pesos sinápticos — que representam a força das conexões entre os neurônios.

Esses três componentes - neurônios, arquitetura e aprendizagem - são elementos essenciais para o funcionamento e eficácia dos modelos de AM. Cada um desempenha um papel único no processamento e na adaptação da rede para realizar suas funções.

2.2.1 Principais Arquiteturas de Redes Neurais

A versatilidade das redes neurais propiciou o surgimento de múltiplas arquiteturas especializadas, cada uma projetada com uma topologia própria, visando otimizar o desempenho conforme as exigências de diferentes tipos de tarefas.

2.2.1.1 *Perceptron*

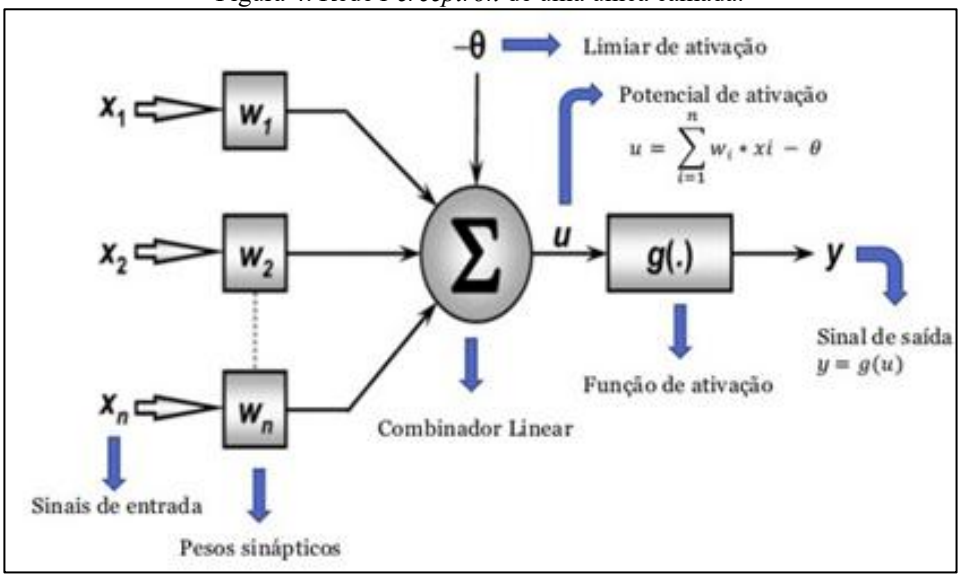
O *perceptron* é um tipo simples de rede neural proposto por Frank Rosenblatt em 1957 e é considerado um dos fundamentos iniciais das redes. O *perceptron* é projetado para aprender uma função que mapeia um conjunto de entradas para uma saída binária.

Conforme DATA SCIENCE ACADEMY (2022), o *perceptron* é um modelo matemático inspirado no funcionamento de um neurônio biológico. Nesse modelo, os sinais elétricos são representados por valores numéricos de entrada, que formam o

chamado vetor de entrada. A modulação desses sinais, que ocorre nas sinapses, é simulada pela multiplicação de cada entrada por um peso, e o conjunto desses pesos constitui o vetor de pesos. O neurônio artificial então calcula a soma ponderada das entradas e a submete a uma função de ativação. Essa função determina a saída do *perceptron*, simulando o disparo de um neurônio real quando um limiar de estímulo é ultrapassado. Os pesos são inicialmente aleatórios e ajustados durante o processo de treinamento da rede para que o modelo aprenda a executar sua tarefa. A estrutura básica do Perceptron está representada graficamente na figura 4.

Durante o treinamento, os pesos são ajustados iterativamente com base nas diferenças entre a saída desejada e a saída real. O objetivo é minimizar o erro na classificação.

Figura 4. Rede *Perceptron* de uma única camada.



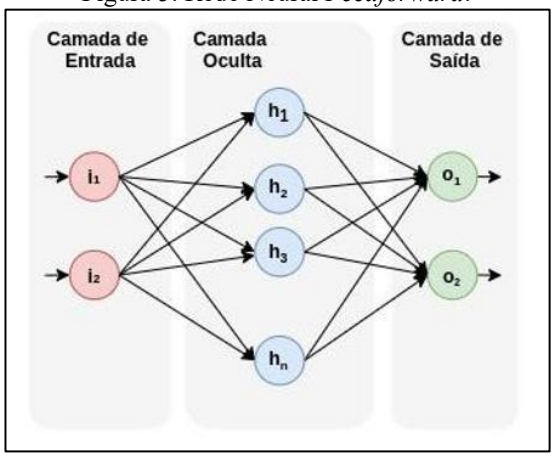
Fonte: PALMIERE, 2016.

2.2.1.2 Redes Neurais Alimentação Direta

Embora o perceptron de camada única tenha representado um marco significativo, sua arquitetura apresentava uma limitação crítica: a incapacidade de resolver problemas que não são linearmente separáveis, como o clássico problema XOR. Para superar essa barreira, foram desenvolvidas as redes neurais de alimentação direta (*feedforward*), frequentemente referidas como *perceptrons* multicamadas (MLP). Esses sistemas de neurônios artificiais são organizados em camadas interconectadas, onde a informação flui em uma única direção, da camada de entrada para a camada de saída, permitindo a modelagem de relações de alta complexidade.

Conforme o site ESTATÍSTICA FÁCIL ([s.d.]), a arquitetura de uma *feedforward* é organizada em uma sequência de três tipos de camadas, como ilustrada na figura 5. O processo inicia-se na camada de entrada, responsável por receber os dados brutos. Em seguida, esses dados atravessam uma ou mais camadas ocultas, onde ocorrem as principais transformações e a extração de características por meio de funções de ativação. O fluxo culmina na camada de saída, que consolida o processamento e fornece o resultado final do modelo, seja ele uma classificação, uma previsão ou outra inferência. É essa estrutura hierárquica que capacita a rede a aprender padrões de alta complexidade.

Figura 5. Rede Neural *Feedforward*.



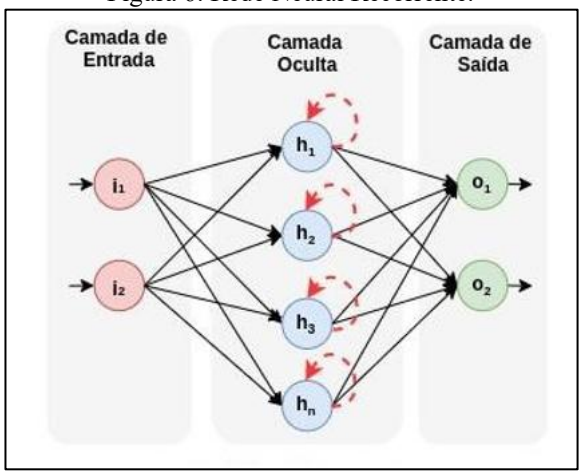
Fonte: BARBOSA et al, 2021.

2.2.1.3 Redes Neurais Recorrentes

Conforme o site AWARI (2023), as redes neurais recorrentes (RNN) apresentam diversas vantagens no aprendizado a partir de dados sequenciais. Entre as principais vantagens, destacam-se a capacidade de lidar com sequências de comprimento variável, a modelagem de dependências temporais, a memória de longo prazo e a habilidade de aprender o contexto das informações.

A estrutura básica de uma RNN consiste em unidades recorrentes (também conhecidas como células) que têm conexões retroativas, permitindo que o conhecimento seja propagado de uma etapa de tempo para a seguinte. Cada unidade recorrente recebe uma entrada na etapa de tempo atual e uma entrada do estado oculto (ou estado da célula) da etapa de tempo anterior, como exemplificado na figura 6. Por fim, produzem uma saída na etapa de tempo atual e atualizam seu estado interno para a próxima etapa de tempo.

Figura 6. Rede Neural Recorrente.



Fonte: BARBOSA et al, 2021.

2.2.1.4 Redes Neurais Convolucionais

A Rede Neural Convolucional (CNN), também conhecida como *ConvNet*, é um tipo de rede neural profunda projetada especificamente para tarefas de visão computacional, como classificação de imagens, detecção de objetos e segmentação semântica. As CNN são inspiradas na forma como o cérebro humano processa percepções visuais e são altamente eficazes na extração de características de imagens.

Consoante com VARGAS et al. (2016), a principal característica das CNN é a camada de convolução. Essa camada aplica filtros em dados visuais, preservando a relação de vizinhança entre os pixels da imagem durante o processamento. Isso permite a extração de características locais, como bordas, texturas e padrões.

De acordo com a IBM ([s.d.]), a estrutura de uma CNN é tipicamente composta por três tipos principais de camadas, cada uma com uma função específica no processo de extração de características. Representada na figura 7.

Camada Convolucional: Esta é a camada fundamental e a primeira de uma CNN. Seu principal objetivo é detectar características locais na imagem de entrada. Para isso, ela utiliza filtros (ou *kernels*), que são pequenas matrizes de pesos que deslizam sobre toda a imagem. Cada filtro é especializado em identificar um padrão específico, como uma aresta vertical, uma curva ou uma determinada cor. O resultado dessa operação é um mapa de características (*feature map*), que indica em quais regiões da imagem o padrão do filtro foi encontrado.

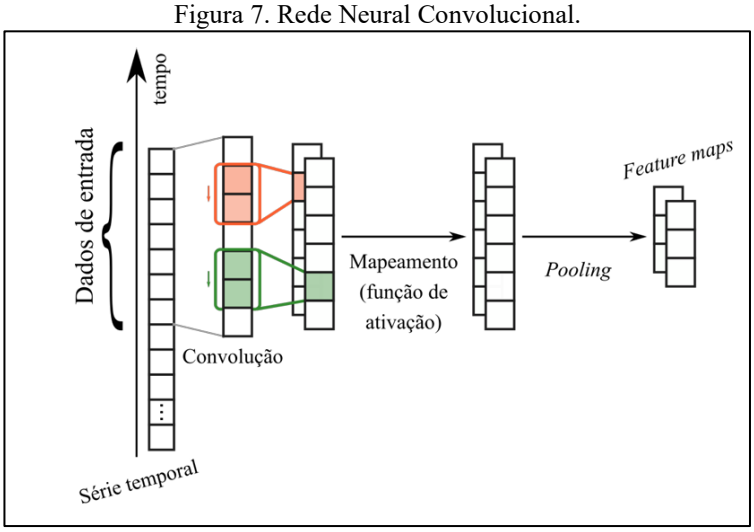
Camada de *Pooling* (Agrupamento): Frequentemente posicionada após uma camada convolucional, sua função é reduzir a dimensionalidade espacial dos mapas de características. Ao diminuir a altura e a largura dos dados, ela torna a representação mais

gerenciável, reduz o custo computacional e ajuda a criar uma invariância a pequenas translações na imagem, tornando o modelo mais robusto.

Camada Totalmente Conectada (*Fully Connected* - FC): Localizada ao final da arquitetura, esta camada recebe os mapas de características de alto nível, já processados e refinados pelas camadas anteriores, e os utiliza para realizar a tarefa de classificação. Ela conecta cada neurônio da sua entrada a cada um de seus próprios neurônios, funcionando como um classificador que mapeia as características extraídas para a saída final desejada, como a probabilidade de a imagem pertencer a uma determinada classe.

Esse processo ocorre de forma hierárquica. Conforme os dados avançam pela rede, as primeiras camadas convolucionais aprendem a identificar características simples e genéricas (como bordas e texturas). As camadas mais profundas, por sua vez, combinam essas características simples para reconhecer padrões mais complexos e específicos do objeto (como olhos, rodas ou um focinho), culminando na identificação do objeto completo pela camada totalmente conectada.

Figura 7. Rede Neural Convolucional.



Fonte: BARINO e DOS SANTOS, 2020.

2.3 Processamento Digital de Imagens

Conforme TRAINA et al (2003), processamento digital de imagens (PDI) envolve técnicas de transformação de imagens, nas quais tanto a imagem original quanto a imagem resultante são apresentadas em uma representação visual (geralmente matricial). Essas transformações têm como objetivo melhorar as características visuais da imagem, como aumentar o contraste, a nitidez ou reduzir ruídos e distorções.

2.3.1 Visão Computacional

Segundo BARROS et al (2024) “A visão computacional trata da extração de informações das imagens e da identificação e classificação de objetos presentes nessa imagem [...]”. A visão computacional busca capacitar as máquinas a interpretar e entender o conteúdo visual da mesma forma que os humanos. Baseada em princípios de processamento de imagens, a visão computacional utiliza algoritmos avançados para analisar imagens e vídeos, identificar padrões e realizar tarefas complexas como detecção de objetos, reconhecimento de cenas e análise de movimento. Essa tecnologia é fundamental para aplicações que vão desde veículos autônomos e sistemas de vigilância até aplicativos de realidade aumentada.

2.3.1.1 Reconhecimento Facial

O reconhecimento facial tem como objetivo identificar ou verificar a identidade de indivíduos com base nas características únicas de seus rostos, sendo aplicado em áreas como segurança (autenticação e controle de acesso), investigações policiais, desbloqueio de dispositivos e transações financeiras.

Conforme OLIVEIRA (2022), o reconhecimento de imagens pode identificar diversos elementos, como lugares, logotipos, pessoas, objetos e edifícios, entre outros, em imagens digitais. Assim, um computador ou dispositivo consegue "ver" a imagem através dos valores numéricos dos pixels, proporcionando uma análise mais detalhada dos dados contidos na imagem.

Entretanto, a eficácia de um sistema de reconhecimento facial está diretamente relacionada à arquitetura da rede neural utilizada. Por isso, é essencial realizar um estudo comparativo dessas estruturas para identificar as soluções mais eficazes em termos de aplicações e controle de acesso.

2.3.1.1.1 Técnicas Clássicas vs. Aprendizado Profundo

O reconhecimento facial evoluiu significativamente com o avanço do aprendizado profundo. Técnicas clássicas, como a Análise de Componentes Principais (PCA), a Análise Discriminante Linear (LDA) e o método *Eigenfaces*, baseavam-se na extração manual de características e apresentavam desempenho limitado em ambientes com variações de iluminação, pose e expressão facial.

Consoante KSHIRSAGAR et al (2011), a abordagem *Eigenfaces* utiliza o algoritmo PCA para reduzir a dimensionalidade das imagens, facilitando o

reconhecimento facial. O processo envolve a transformação das imagens em um espaço de características de menor dimensão e a identificação das chamadas "*Eigenfaces*", que representam as variações mais significativas entre diferentes rostos. Apesar de sua simplicidade e eficiência computacional, esse método apresenta limitações, especialmente em condições adversas, como variações acentuadas de iluminação, pose e expressões faciais.

De acordo com SILVA (2016), a LDA é uma técnica de redução de dimensionalidade supervisionada, que opera sobre um conjunto de dados previamente rotulados. Seu principal objetivo é projetar os dados de um espaço de alta dimensão para um espaço com menos dimensões, de forma a maximizar a separabilidade entre as diferentes classes. Em outras palavras, a LDA busca encontrar uma nova representação de menor dimensionalidade onde os grupos de dados sejam os mais distintos e compactos possível.

Em comparação com os métodos modernos baseados em aprendizado profundo, os métodos tradicionais tendem a ser menos robustos frente a variações de iluminação, ângulo de visão e expressões faciais. Conforme aponta SANTOS (2017), o reconhecimento facial enfrenta desafios devido à alta dimensionalidade dos dados, uma vez que as imagens são representadas por matrizes de *pixels*. Técnicas como PCA e LDA são empregadas para reduzir essa dimensionalidade, mas ainda são impactadas por fatores externos que afetam a acurácia.

Com o avanço das redes neurais convolucionais, muitas das limitações dos métodos tradicionais foram superadas. As CNN permitem a extração automática e hierárquica de características visuais, sendo mais precisas e eficazes em condições diversas. Ainda assim, os métodos clássicos possuem importância histórica e teórica, sendo fundamentais para a compreensão da evolução dos sistemas de reconhecimento facial ao longo do tempo.

2.3.1.1.2 Detecção Facial vs. Reconhecimento Facial

É fundamental diferenciar os conceitos de detecção facial e reconhecimento facial. A detecção facial é a etapa inicial e consiste em localizar rostos dentro de uma imagem. Isso é feito utilizando técnicas como *Haar Cascades*, que identificam padrões faciais com base em características simples.

Como aponta SILVA (2018, p. 20), "antes de executar um algoritmo de reconhecimento facial, é de praxe realizar uma detecção facial a fim de concentrar os

esforços do reconhecedor facial apenas nas áreas relevantes". Ou seja, a detecção facial precede o reconhecimento, focando o reconhecimento facial nas áreas relevantes encontradas. O reconhecimento facial, por outro lado, envolve identificar ou verificar a identidade de um rosto detectado, comparando-o com um banco de dados de rostos previamente cadastrados. Em resumo, enquanto a detecção facial localiza o rosto, o reconhecimento facial identifica a pessoa.

No sistema proposto neste trabalho, ambas as etapas são integradas: a detecção é utilizada como passo inicial para isolar o rosto na imagem capturada e, em seguida, o reconhecimento facial é realizado para autenticar o usuário e permitir ou negar o acesso ao ambiente

2.4 Sistemas de Controle de Acesso

Controle de acesso refere-se à gestão e restrição do acesso a recursos físicos (edifícios, salas) ou digitais (sistemas, dados) a indivíduos ou entidades autorizadas, sendo essencial para a segurança e conformidade. Conforme aponta SORAGGI (2024), a implementação deste processo é vital para evitar danos financeiros e reputacionais, uma vez que garante a proteção de dados, a conformidade regulatória e a prevenção contra acessos não autorizados.

Segundo GUIMARÃES (2021), o controle de acesso é dividido em duas categorias: lógico e físico. O controle de acesso lógico envolve o uso de tecnologias para impedir o acesso não autorizado a documentos, dados e informações, realizando a verificação de identidade por meio de *login* e senha ou biometria, com dados armazenados em bancos online. Já o controle de acesso físico gerencia a entrada e saída de pessoas, veículos e materiais em áreas restritas, utilizando registros de data, horário, responsável e, no caso de visitantes, o acompanhamento por funcionários e identificação com foto e/ou digital.

A utilização do reconhecimento facial como método de autenticação tem se mostrado eficaz e eficiente na identificação de pessoas com rapidez e precisão, além de remover dispositivos físicos, como carteiras de identidade, a detecção oferece segurança adicional, dificultando o acesso não autorizado. De acordo com AHMADI (2020) empregando estruturas de inteligência artificial para identificar e mitigar riscos à segurança da informação, demonstra a eficiência na redução de ataques cibernéticos e auxiliando gestores na avaliação e gestão desses riscos.

3. Desenvolvimento

O presente capítulo descreve a implementação prática do projeto. Primeiramente, detalha-se o ciclo de treinamento e validação de três modelos de CNN, englobando a descrição do conjunto de dados, o pré-processamento das imagens, a definição e justificativa da arquitetura, a configuração dos hiperparâmetros e os critérios de validação. Subsequentemente, são apresentadas as técnicas para salvamento do modelo final e os formatos de visualização utilizados para interpretar os resultados.

3.1 Conjunto de Dados

Segundo o site DESCOMPLICA.IA (2024), um conjunto de dados é uma coleção estruturada de informações que pode ser utilizada para análise, treinamento de modelos de inteligência artificial e aprendizado de máquina. Esses dados podem vir em diferentes formatos, como tabelas, arquivos *csv*, bancos de dados e até mesmo imagens. A qualidade e a quantidade dos dados contidos em um conjunto são cruciais para o desempenho de algoritmos de IA, pois eles aprendem a partir das informações que recebem.

O conjunto de dados empregado neste trabalho foi derivado do *CelebA* (*CelebFaces Attributes Dataset*), um *benchmark* de larga escala amplamente utilizado na comunidade de visão computacional. Embora o *dataset* completo contenha mais de 200.000 imagens de mais de 10.000 celebridades, para viabilizar a experimentação dentro das restrições de hardware e tempo deste projeto, foi selecionado um subconjunto de 2.000 identidades únicas.

A principal vantagem deste *dataset* reside na sua diversidade, que abrange significativas variações de pose, iluminação e fundo. Para a presente pesquisa, o fator crucial foi a disponibilidade dos rótulos de identidade, que permitiu a estruturação dos dados deste subconjunto em pares e tripla, um pré-requisito fundamental para o treinamento dos modelos de aprendizagem métrica[1] profunda. A divisão entre os conjuntos de treinamento e validação foi realizada sobre estas 2.000 identidades, garantindo que não houvesse sobreposição de indivíduos entre os conjuntos.

[1] A Aprendizagem Métrica consiste em aprender uma função de distância ou similaridade a partir dos dados. Em vez de utilizar métricas genéricas, este paradigma otimiza um mapeamento para um espaço de características (*embedding space*) no qual a distância inter-classes é maximizada e a distância intra-classe é minimizada. Este processo é frequentemente orientado por funções de perda especializadas, como a perda tripla (*triplet loss*) ou a contrastiva (*contrastive loss*).

3.2 Pré-Processamento dos Dados

O pré-processamento de dados configura-se como uma etapa essencial no desenvolvimento de modelos de redes neurais, uma vez que visa transformar dados brutos em formatos apropriados para o aprendizado de máquina. De acordo com PINHEIRO (2021), o pré-processamento "envolve um conjunto de atividades que convertem dados brutos em dados preparados, ou seja, em formatos úteis e eficientes, abrangendo a preparação, organização e estruturação dos dados."

As seguintes etapas de pré-processamento foram aplicadas a todas as imagens antes de serem inseridas nos modelos:

1. Leitura e Decodificação: As imagens foram lidas do disco e decodificadas para o formato de tensor.
2. Redimensionamento: Todas as imagens foram redimensionadas para um tamanho padrão (por exemplo, $128x128$ *pixels*), garantindo consistência dimensional na entrada da rede.
3. Normalização: Os valores dos *pixels*, originalmente no intervalo, foram normalizados para o intervalo. Essa prática estabiliza e acelera o treinamento, garantindo que todas as características de entrada tenham uma escala semelhante.
4. Aumento de Dados (*data augmentation*): Aplicado seletivamente em um dos modelos, esta técnica gera novas imagens de treinamento por meio de transformações aleatórias (como rotações e inversões). O objetivo é aumentar artificialmente a diversidade do *dataset* para mitigar o sobreajuste e melhorar a capacidade de generalização do modelo.

3.3 Implementação dos Modelos baseados em CNN

Para este trabalho, três arquiteturas baseadas em CNN, com foco na geração de *embeddings* discriminativos, foram desenvolvidas e avaliadas para a tarefa de reconhecimento facial. Inicialmente, o modelo Siamês com função de perda contrastiva demonstrou eficácia ao mapear rostos em um espaço vetorial, aproximando pares semelhantes e distanciando os distintos, por meio do aprendizado supervisionado com pares rotulados. Em seguida, o modelo baseado em *Triplet Loss* destacou-se pela acurácia, promovendo uma representação vetorial mais robusta e sensível às variações faciais, ao otimizar simultaneamente a compactação intra-classe e a separação inter-classe. Por fim, uma versão refinada da arquitetura tripla introduziu melhorias significativas no *pipeline* de dados, com ênfase na estabilidade e eficiência computacional

durante o treinamento, mantendo a mesma estrutura de perda. A avaliação de desempenho, conduzida com base em limiares de distância euclidiana, confirmou a superioridade dos modelos baseados em *Triplet Loss*, estabelecendo-os como referência técnica para aplicações de autenticação facial.

3.3.1 Modelo siamês com função de perda contrastiva

O modelo com a função de perda contrastiva (*Contrastive Loss*) tem como objetivo organizar os vetores em um espaço onde a distância entre eles reflita diretamente à similaridade facial — aproximando vetores de rostos da mesma pessoa e afastando os de indivíduos diferentes. Essa estratégia permite que a rede crie uma "impressão digital" numérica para cada rosto, viabilizando a verificação e identificação por meio da comparação das distâncias entre esses vetores.

Passo 1. Configuração de hiperparâmetros

No primeiro passo, é estabelecido as bases para o experimento. São definidos os caminhos para o conjunto de dados (*dataset*) em dataset_path e para o arquivo onde o melhor *modelo* (*model*) será salvo, em model_checkpoint_path. Em seguida, são configurados os *hiperparâmetros* que governam tanto a arquitetura quanto o processo de treinamento. Parâmetros como img_size = (96, 96), batch_size = 32 e embedding_dim = 128 definem as dimensões das imagens de entrada, o tamanho dos lotes de dados e a dimensionalidade do vetor de características final (*embedding*). O *hiperparâmetro* mais crucial nesta seção, específico para a função de perda utilizada, é $margin = 1.0$. Este valor estabelece a distância mínima desejada entre vetores de imagens de pessoas diferentes, sendo um pilar para o funcionamento da *contrastive loss*.

Passo 2. *Pipeline* de dados com *tf.data.from_generator*

Esta seção é dedicada à preparação e ao carregamento dos dados de forma eficiente. Inicialmente, o código divide as identidades das pessoas em conjuntos de treino e validação para garantir que o modelo seja avaliado em dados não vistos. O núcleo do *pipeline* é a função *create_pair_generator_from_dict*, que implementa a lógica de geração de pares. Com uma probabilidade de 50%, ela produz um par positivo (duas imagens da mesma pessoa, com rótulo 1.0) ou um par negativo (imagens de duas pessoas distintas, com rótulo 0.0). A função de pré-processamento, *preprocess_image* contém as

etapas essenciais de leitura, redimensionamento e normalização, sem aumento de dados. Por fim, *tf.data.Dataset.from_generator* transforma esse processo em um *pipeline* de dados do *TensorFlow*, otimizado com os métodos *.batch()* e *.prefetch()* para garantir um fluxo de dados rápido e contínuo para a *GPU* durante o treinamento.

Passo 3. Arquitetura do modelo

A arquitetura do modelo é definida em duas partes principais. Primeiro, a função *build_embedding_model* constrói a rede que serve como extrator de características. Esta rede processa uma imagem e a transforma em um vetor de *embedding_dim* dimensões. Uma parte crucial é o uso da camada $layers.UnitNormalization(axis = -1)$. Esta camada realiza a normalização *L2* do vetor de saída, garantindo que todos os *embeddings* tenham comprimento 1, o que estabiliza o treinamento e é uma prática recomendada.

Em seguida, o modelo siamês completo é montado, possuindo duas entradas (*input_a, input_b*) que são alimentadas na mesma *instância* do *embedding_model*, garantindo o compartilhamento de pesos (*weight sharing*). A saída desses dois ramos é então passada para uma camada *layers.Lambda*, que calcula a distância *euclidiana* entre os dois vetores. Uma atualização técnica vital é a especificação do argumento *output_shape* nesta camada, o que resolve problemas de serialização e permite que o modelo seja salvo e carregado corretamente.

Passo 4. Função de perda

Neste passo, o modelo é preparado para o treinamento. A peça central é a função *contrastive_loss*, que define como o erro (*loss*) será calculado. Para pares positivos ($y\_true = 1$), a perda é o quadrado da distância predita, incentivando o *modelo* a minimizar essa distância. Para pares negativos ($y\_true = 0$), a perda só é calculada se a distância for menor que a *margin* pré-definida. Esse mecanismo força o modelo a afastar as representações de classes diferentes até que atinjam essa margem de separação, sem gastar esforço computacional em pares que já estão bem separados. O modelo siamês é então compilado com esta função de perda e o otimizador *Adam*, configurado com uma taxa de aprendizado (*learning rate*) de 2^{-4}. A compilação une a

arquitetura que calcula a distância com a função de *loss* que a interpreta, preparando tudo para o aprendizado.

Passo 5. Treinamento e *Callbacks*

O processo de treinamento é iniciado com o método *.fit()*, que alimenta o *modelo* com os *datasets* de treino e validação. A estratégia de treinamento aqui é mais direta, utilizando um único *callback*: *ModelCheckpoint*. Este *callback* monitora a perda no conjunto de validação (*val_loss*) a cada época e salva o *modelo* apenas quando um novo valor mínimo é alcançado. Isso garante que, ao final do treinamento, o arquivo .h5 conterá a versão do modelo com o melhor desempenho em dados não vistos.

Passo 6. Avaliação de performance

Após o término do treinamento, a performance do modelo é avaliada de forma rigorosa. Primeiro, o melhor modelo salvo pelo *ModelCheckpoint* é carregado. Em seguida, um grande lote de pares de validação é gerado para uma avaliação estatisticamente significativa. O passo final e mais importante é encontrar o limiar de decisão (*threshold*) ideal. O código itera sobre uma gama de possíveis *thresholds*, e para cada um, calcula a acurácia, que é a porcentagem de pares classificados corretamente. O *threshold* que resulta na maior acurácia é selecionado como o ideal, e essa acurácia máxima é reportada como a métrica final de performance do modelo.

3.3.2 Modelo *triplet* com processamento ágil de dados

O modelo utiliza a capacidade da função *Triplet Loss* de estruturar um espaço de características onde a distância vetorial reflete diretamente à similaridade facial. O modelo foi otimizado para agrupar as representações da mesma pessoa (minimização da distância intra-classe) e separar as de pessoas diferentes (maximização da distância inter-classe), gerando uma assinatura vetorial única e altamente discriminativa para cada face.

Passo 1. Configuração de hiperparâmetros

No primeiro passo, é estabelecido as bases para o experimento. A estrutura de caminhos (*dataset_path, model_checkpoint_path*) e os parâmetros de entrada (*img_size, batch_size*) foram mantidos consistentes. Um hiperparâmetro de destaque nesta configuração é a *margin*, definida em 0.5. Este valor, que define a fronteira de separação

mínima exigida pela *Triplet Loss*, provou ser um ajuste ótimo para o equilíbrio entre a compactação dos grupos intra-classe e a separação dos grupos inter-classe, contribuindo diretamente para o desempenho final.

Passo 2. *Pipeline* de dados com *tf.data.from_generator*

Um dos fatores mais determinantes para o desempenho deste modelo é a abordagem adotada no pipeline de dados. A principal característica deste modelo é a ausência intencional de aumento de dados. Contrariando a intuição de que a falta de regularização a nível dos dados poderia levar ao sobreajuste, o modelo demonstrou uma notável capacidade de generalização. Este resultado sugere que a arquitetura do *backbone* é inerentemente robusta, capaz de aprender as características faciais discriminativas diretamente do *dataset* original, sem a necessidade de variações artificiais para obter um bom desempenho.

Passo 3. Arquitetura do modelo

A arquitetura do modelo é composta por uma sequência de camadas *Conv2D*, *BatchNormalization*, *MaxPooling2D* e *Dropout*, estabelecendo um extrator de características eficaz. Uma mudança significativa é a inclusão da camada *layers*.*UnitNormalization*. Ao normalizar os *embeddings* de saída para terem norma L2 unitária, esta camada estabilizou o treinamento e garantiu que a *Triplet Loss* otimizasse a orientação angular dos vetores, um fator essencial para o desempenho em funções de perda baseadas em distância.

Passo 4. Função de perda

A consistência no mecanismo de aprendizado foi um pilar para o resultado obtido. O modelo utilizou a implementação robusta da *triplet_loss* customizada, que aplica a fórmula $max(0, positive\_dist - negative\_dist + margin)$. A compilação com o otimizador *Adam* e uma taxa de aprendizado de 1^{-4} demonstrou ser a combinação ideal para minimizar esta função de perda de forma estável, O que confere ao modelo uma maior capacidade de aplicar seu aprendizado a situações não vistas anteriormente.

Passo 5. Treinamento e *Callbacks*

A estratégia de treinamento representa uma abordagem madura e eficaz. O processo, gerenciado pelo método *.fit*, é otimizado com dois *callbacks* fundamentais. O

ModelCheckpoint garante que o modelo com a menor perda no conjunto de validação (*val_loss*) seja salvo, capturando o exato momento de pico de performance. Complementarmente, o *ReduceLROnPlateau* ofereceu uma camada extra de robustez, ajustando a taxa de aprendizado para escapar de platôs e refinar a convergência.

Passo 6. Avaliação de performance

Para avaliar a performance, o melhor modelo salvo é carregado e seu núcleo (*EmbeddingModel*) gera *embeddings* para um lote de teste com pares positivos e negativos. Uma busca exaustiva sobre as distâncias euclidianas calculadas identifica o limiar de decisão ótimo, que maximizou a acurácia da classificação.

3.3.3 Modelo *triplet* com otimização estável de *pipeline*

Esta implementação refina o modelo de aprendizado de similaridade baseado na Perda de Trinca. O objetivo central permanece o mesmo: treinar uma rede para gerar *embeddings* onde a distância entre eles reflete a semelhança facial. A principal inovação desta versão não reside na arquitetura da rede neural ou na função de perda, mas sim na reengenharia do *pipeline* de dados. Ao adotar uma abordagem mais robusta e eficiente, o modelo garante estabilidade durante treinamentos longos, eliminando potenciais erros de execução e otimizando o uso de recursos computacionais.

Passo 1. Configuração de hiperparâmetros

A configuração inicial estabelece os parâmetros fundamentais para o treinamento. Os caminhos para o conjunto de dados e para o salvamento do modelo são definidos, juntamente com os hiperparâmetros essenciais. O valor da *margin* é definido em 0.5, um padrão comum para a Perda de Trinca que estabelece o "espaço" desejado entre a distância de um par positivo e um par negativo. Os demais parâmetros, como *img_size*, *batch_size* e *embedding_dim*, formam a base consistente para a arquitetura de extração de características e o processo de treinamento.

Passo 2. Pipeline de dados com arquitetura $Generator -> Map$

A mudança mais significativa nesta versão está na construção do pipeline de dados, projetado para máxima estabilidade. O problema de instabilidade em geradores *Python* que realizam operações de I/O (leitura de arquivos) durante o treinamento é resolvido com uma estratégia de duas etapas:

1. Geração de caminhos: Um gerador *Python* (*triplet_path_generator*) é responsável exclusivamente pela lógica de seleção das trincas. Ele não carrega as imagens, mas apenas produz os caminhos dos arquivos (*strings*) para a âncora, o positivo e o negativo.
2. Mapeamento e processamento: Uma segunda função *(load_and_preprocess_triplets*), projetada para operar dentro do grafo do TensorFlow, recebe esses caminhos. O método *.map()* aplica essa função de forma eficiente e paralela a cada conjunto de caminhos gerado, realizando a leitura, decodificação, redimensionamento e normalização das imagens.

Passo 3. Arquitetura do modelo

A arquitetura do modelo consiste em uma CNN customizada, com blocos sequenciais de *Conv2D*, *BatchNormalization*, *MaxPooling2D* e *Dropout*. Esta estrutura é projetada para extrair características hierárquicas das imagens de face. A camada final do *embedding_model* é a *layers.UnitNormalization*, que é crucial para normalizar o vetor de características para um comprimento unitário (norma L2), garantindo que as distâncias calculadas posteriormente sejam consistentes e comparáveis. O *triplet_model* final é composto por três ramos de entrada, cada um processado pela mesma instância compartilhada do *embedding_model*, com os três *embeddings* resultantes concatenados em um único tensor de saída.

Passo 4. Função de perda

Esta implementação utiliza a função de perda de trinca em sua formulação padrão. A função calcula a perda para cada trinca com base na fórmula: $loss = max(0, D(A,P)^2 - D(A,N)^2 + margin)$, onde $D(A,P)$ é a distância entre a âncora e o positivo, e $D(A,N)$ é a distância entre a âncora e o negativo. O objetivo é forçar a distância do par negativo a ser maior que a do par positivo por pelo menos o valor da margem. Esta abordagem fundamental, sem estratégias de mineração de exemplos difíceis ("*hard mining*") na função de perda, serve como uma linha de base sólida e eficaz.

Passo 5. Treinamento e *Callbacks*

O processo de treinamento é acionado pelo método *.fit*. A estratégia de *callbacks* é completa, utilizando duas ferramentas essenciais:

- *ModelCheckpoint*: Monitora a perda no conjunto de validação (*val_loss*) e salva o modelo apenas quando ocorre uma melhoria, garantindo que a melhor versão seja preservada.
- *ReduceLROnPlateau*: Monitora a mesma métrica (*val_loss*) e reduz automaticamente a taxa de aprendizado caso o treinamento estagne (não haja melhoria por um número definido de épocas), ajudando o modelo a encontrar um ótimo local com mais precisão.

Passo 6. Avaliação de performance

A avaliação de performance segue a metodologia padrão para modelos de aprendizado de similaridade. O objetivo é encontrar o limiar de distância ideal que melhor separa pares da mesma pessoa de pares de pessoas diferentes. O processo envolve carregar o melhor modelo salvo pelo *ModelCheckpoint*, isolar o *embedding_model* para gerar *embeddings* de imagens individuais, e então utilizar essas características para calcular a distância Euclidiana para um grande lote de pares de validação. Por fim, um laço itera sobre múltiplos limiares de decisão, calculando a acurácia para cada um, e o limiar que maximiza a acurácia é selecionado como o ideal, com a acurácia correspondente sendo reportada como a métrica final de desempenho.

4. Métricas Utilizadas para Treinamento e Avaliação dos Modelos

De acordo com o site PURESTOREAGE (2025), a importância das métricas de desempenho em aprendizado de máquina reside em sua capacidade de fornecer uma avaliação quantitativa e objetiva da eficácia de um modelo. Elas são cruciais para a seleção de modelos, permitindo a comparação sistemática entre diferentes algoritmos para identificar o mais adequado a uma tarefa específica. Além disso, as métricas guiam o ajuste de hiperparâmetros, orientando a otimização do modelo para alcançar a melhor performance possível. Do ponto de vista prático, elas conectam o desempenho técnico do modelo ao impacto nos negócios, ajudando a escolher o modelo que melhor atende aos objetivos estratégicos (como minimizar falsos negativos em um diagnóstico médico). Por fim, após a implantação, o monitoramento contínuo dessas métricas é essencial para detectar a degradação ou o desvio do modelo ao longo do tempo, garantindo sua confiabilidade e eficácia contínuas em aplicações do mundo real.

Para analisar de forma completa o desempenho e o comportamento dos modelos de reconhecimento facial desenvolvidos, foram empregadas duas categorias principais de

métricas. A primeira categoria, composta pela Função de Perda (*Loss*), é utilizada para monitorar e guiar o processo de treinamento em tempo real. A segunda, focada na Acurácia de Verificação, é utilizada para avaliar o desempenho prático e final do modelo treinado em uma tarefa de verificação facial. Juntas, essas métricas fornecem uma visão que abrange desde a convergência interna do modelo até sua eficácia em um cenário de aplicação real.

4.1 Métrica de Treinamento: Função de Perda (*Loss*)

A Função de Perda (*Loss*) é a métrica fundamental que orienta o aprendizado da rede neural. Sua definição é a de uma função matemática que quantifica o erro do modelo em relação às suas previsões durante o treinamento. O objetivo desta métrica é produzir um valor numérico que o algoritmo otimizador (como o *Adam*) tenta minimizar. Um valor de perda menor indica que o modelo está cumprindo melhor sua função objetivo, que, no caso destes modelos, é organizar corretamente o espaço de *embedding*. Durante a análise, a perda foi observada em dois contextos:

- Perda de Treinamento (*Training Loss*): Calculada em cada lote de dados de treinamento, ela indica o quão bem o modelo está se ajustando aos dados que está vendo para aprender. A observação de sua queda contínua é um sinal de que o aprendizado está ocorrendo.
- Perda de Validação (*Validation Loss*): Calculada em um conjunto de dados separado que o modelo não usa para aprender, ela serve como uma estimativa do erro de generalização do modelo. O objetivo de monitorá-la em conjunto com a perda de treinamento é diagnosticar o sobreajuste (*overfitting*). Se a perda de treinamento continua a cair enquanto a de validação começa a subir, significa que o modelo está memorizando os dados de treino em vez de aprender características gerais. O ponto de menor perda de validação geralmente indica a "Melhor Época", ou seja, o ponto em que o modelo atingiu seu melhor poder de generalização. Nos modelos analisados, foram utilizadas duas funções de perda específicas: a *Triplet Loss* e a *Contrastive Loss,* ambas com o objetivo final de estruturar o espaço de *embedding* com base na distância.

4.2 Métrica de Avaliação de Performance: Acurácia de Verificação

Após o treinamento, a métrica utilizada para avaliar a performance prática do modelo foi a acurácia de verificação. Sua definição neste contexto é a proporção de pares de imagens (um par sendo duas imagens faciais) que são corretamente classificados como "mesma pessoa" ou "pessoas diferentes". Diferentemente de uma tarefa de classificação tradicional, o modelo não produz uma classe diretamente, mas sim uma distância. Portanto, a acurácia está intrinsecamente ligada a um limiar de decisão (*Decision Threshold*). O objetivo da análise de acurácia foi duplo, envolvendo duas métricas derivadas:

- Acurácia Máxima: Esta métrica representa o maior valor de acurácia alcançado ao testar uma vasta gama de possíveis limiares de decisão. Seu objetivo é servir como a principal medida de desempenho do modelo, quantificando sua melhor performance possível no conjunto de validação. É um valor único que permite comparar objetivamente os diferentes modelos.
- Limiar Ideal: Esta métrica corresponde ao valor numérico do limiar de decisão que resultou na Acurácia Máxima. Seu objetivo é prático e operacional: se o modelo fosse ser implementado em um sistema real, este seria o valor de limiar a ser utilizado para binarizar a decisão (se a distância < limiar, é a mesma pessoa; caso contrário, são pessoas diferentes) e obter o desempenho ótimo. O segundo gráfico gerado em todas as análises visualiza exatamente essa relação, plotando a acurácia para cada limiar e destacando o ponto ideal.

4.3 Análise Visual do Desempenho e Treinamento do Modelo

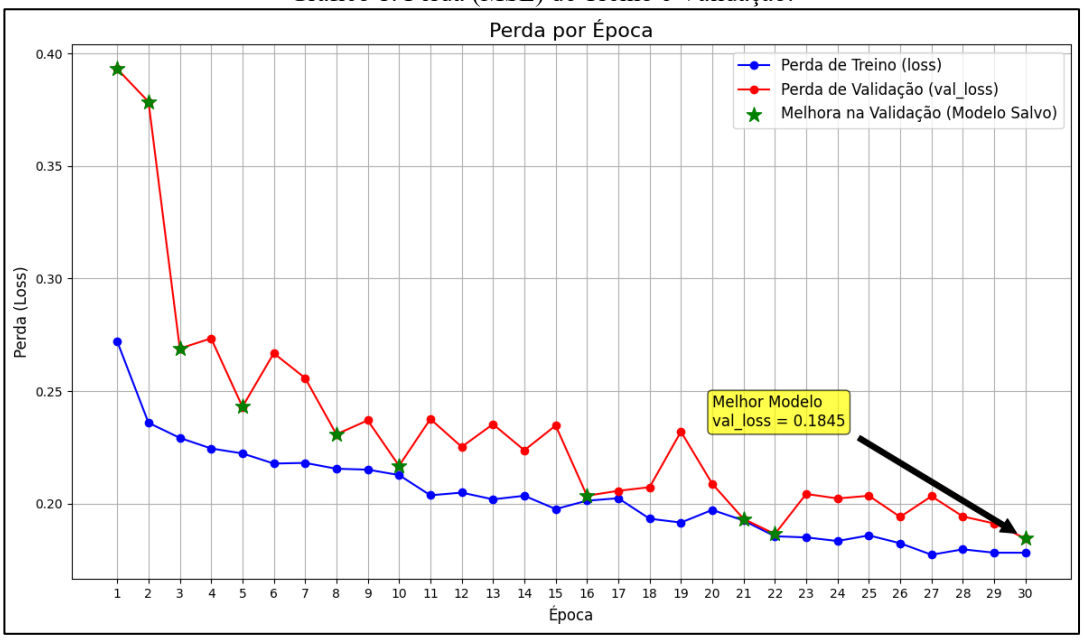
4.3.1 Modelo Siamês com Função de Perda Contrastiva

A análise do modelo revela um processo de aprendizado ao longo de 30 épocas, guiado pela *Contrastive Loss*. O primeiro gráfico, "Perda por Época", visualiza fielmente o log de treinamento, onde a linha azul ("Perda de Treino") exibe uma queda consistente de um valor inicial de 0.27 para 0.17, indicando que o modelo se ajustou progressivamente aos dados. A linha vermelha ("Perda de Validação"), embora mais volátil, também apresenta uma tendência geral de queda, com as estrelas verdes marcando os momentos exatos em que o *callback ModelCheckpoint* salvou uma nova versão do modelo devido a uma melhora na *val_loss*. Conforme destacado no gráfico e confirmado

na última linha do log, o melhor modelo foi obtido na Época 30, alcançando a menor perda de validação de 0.1845.

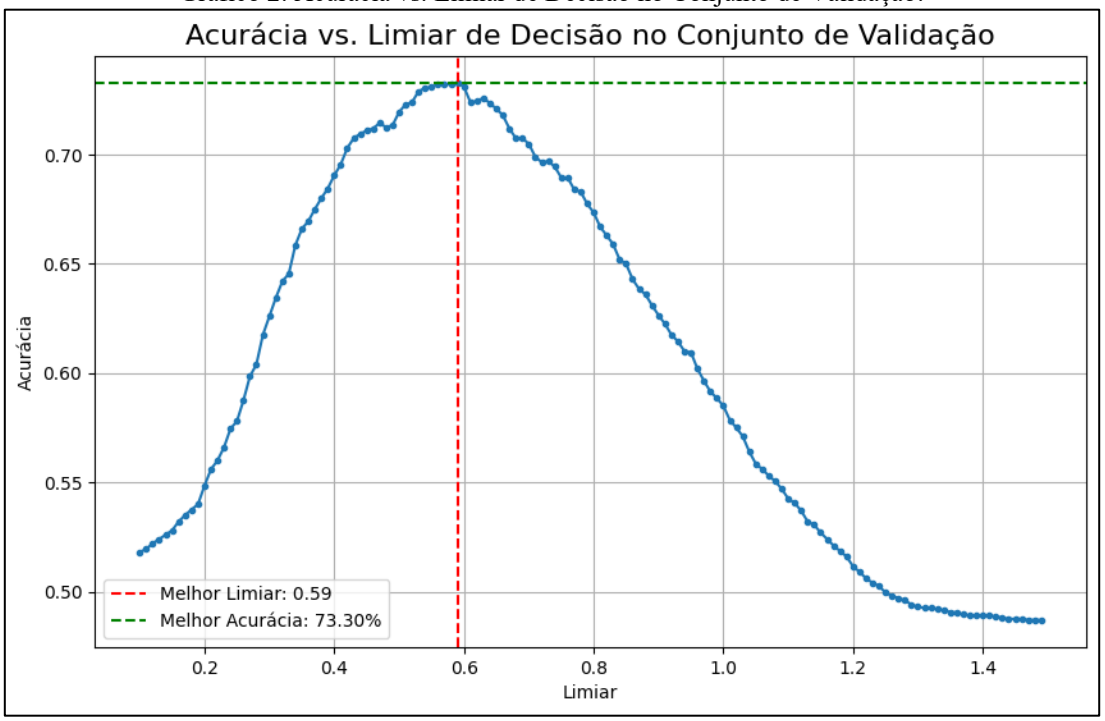
A performance prática do melhor modelo salvo é quantificada no segundo gráfico, "Acurácia vs. Limiar de Decisão". Este gráfico demonstra como a capacidade do modelo de distinguir corretamente entre pares de rostos varia conforme o limiar de distância utilizado para a decisão. O resultado principal indica que o modelo atingiu uma acurácia máxima de 73.30% no conjunto de validação, representando seu desempenho ótimo. Para alcançar essa performance, foi determinado um limiar ideal de 0.59, valor que corresponde ao ponto de equilíbrio que melhor separa pares de imagens da mesma pessoa de pares de pessoas diferentes, conforme indicado pela linha vertical vermelha no pico da curva.

Gráfico 1. Perda (MSE) de Treino e Validação.



Fonte: A autora, 2025.

Gráfico 2. Acurácia vs. Limiar de Decisão no Conjunto de Validação.



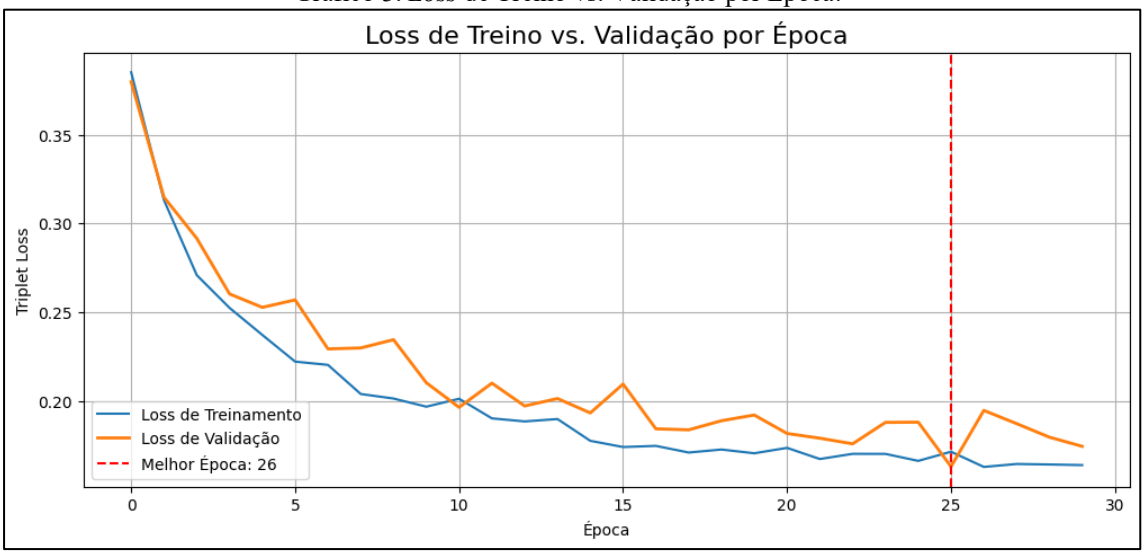
Fonte: A autora, 2025.

4.3.2 Modelo *triplet* com processamento ágil de dados

A análise do treinamento do modelo demonstra um processo de aprendizado bem-sucedido ao longo de 30 épocas. A perda de treinamento (*loss*), que inicia em 0.4289, exibe uma queda consistente e significativa, finalizando em 0.1612, o que indica que o modelo conseguiu se ajustar aos dados. Simultaneamente, a perda de validação (*val_loss*) também mostra uma tendência de queda, sinalizando que o modelo estava generalizando bem. O log evidencia a ação dos *callbacks*: o *ModelCheckpoint* salvou o modelo múltiplas vezes, sempre que uma nova melhor *val_loss* era alcançada, com o melhor desempenho sendo registrado na Época 26, com uma perda de 0.1633. Além disso, o *ReduceLROnPlateau* ajustou a taxa de aprendizado dinamicamente, como na Época 14 e 29, para refinar o treinamento quando a melhora estagnava.

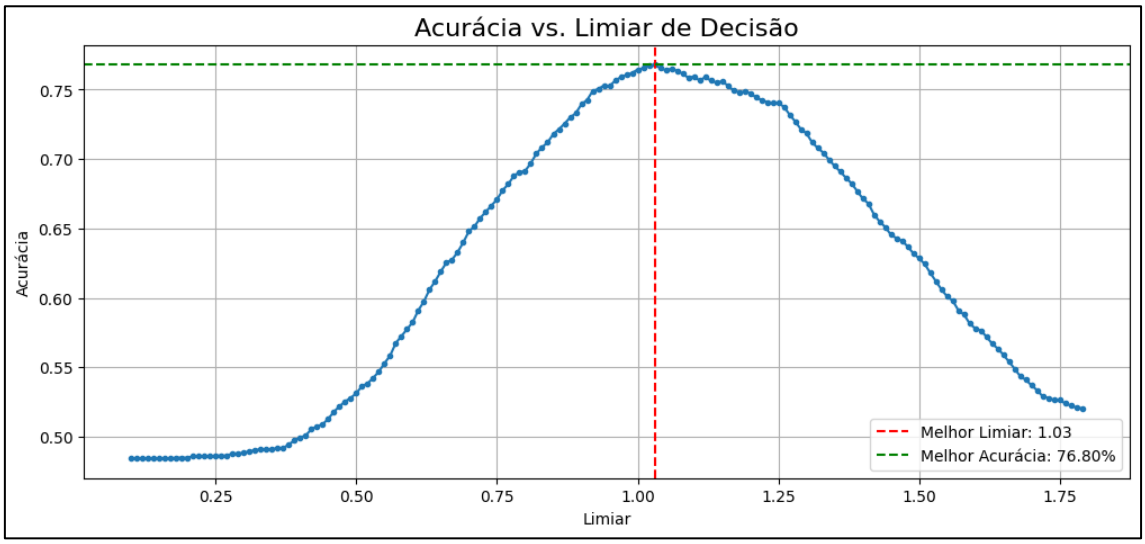
Após o treinamento, o melhor modelo (da Época 26) foi avaliado para determinar sua performance prática. A avaliação final revelou que o modelo alcançou uma acurácia máxima de 76.80% no conjunto de validação. Este valor representa a capacidade ótima do modelo em distinguir corretamente entre pares de imagens da mesma pessoa e de pessoas diferentes. Para atingir essa acurácia, o Limiar Ideal de decisão foi determinado em 1.03, que é o valor de distância que melhor equilibra as classificações corretas, correspondendo ao pico da curva em um gráfico de "Acurácia vs. Limiar".

Gráfico 3. *Loss* de Treino vs. Validação por Época.



Fonte: A autora, 2025.

Gráfico 4. Acurácia vs. Limiar de Decisão



Fonte: A autora, 2025.

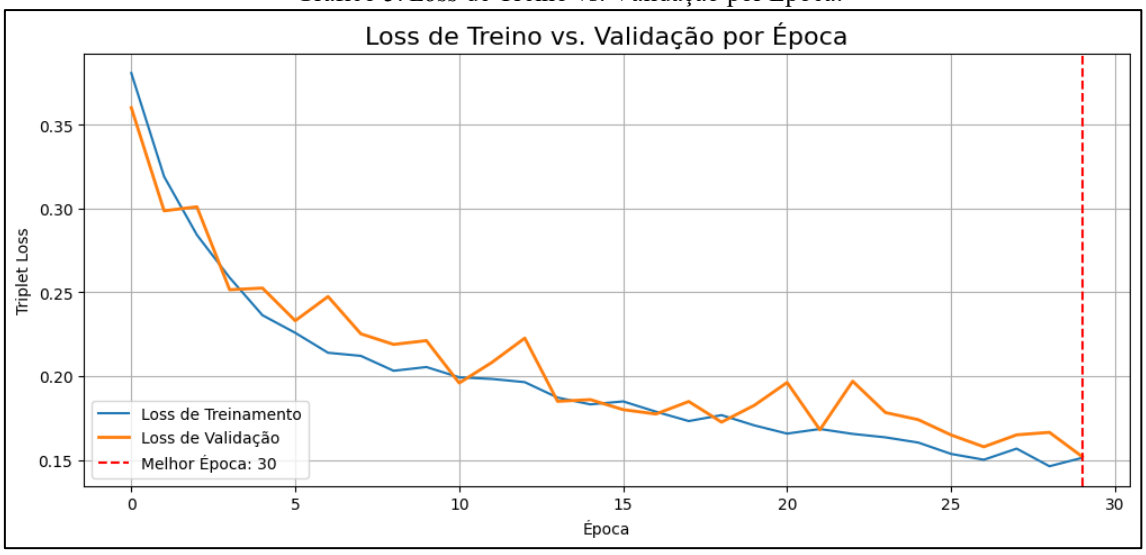
4.3.3 Modelo *triplet* com otimização estável de *pipeline*

A análise do treinamento do modelo documenta o processo de otimização iterativa de um modelo de deep learning, treinado ao longo de 30 épocas. A análise das métricas de perda de treinamento (*loss*) e de validação (*val_loss*) demonstra a convergência do modelo, com uma tendência geral de decréscimo em ambas, indicando que o algoritmo aprendeu a representar os padrões dos dados e a generalizar para amostras não vistas. A estratégia de *checkpointing* foi utilizada para persistir o modelo apenas quando ocorria uma melhoria na perda de validação, uma prática que visa mitigar o *overfitting* ao reter a versão com melhor performance de generalização. Um evento notável ocorre na época

25, quando a estagnação da *val_loss* ativa o *callback ReduceLROnPlateau*, que realiza um ajuste fino ao reduzir a taxa de aprendizado de 1^{-4} para 2^{-5}. Essa intervenção se mostrou eficaz, pois o modelo alcançou novos mínimos de perda nas épocas subsequentes, culminando com o melhor desempenho na época 30, com *val_loss* de 0.15192.

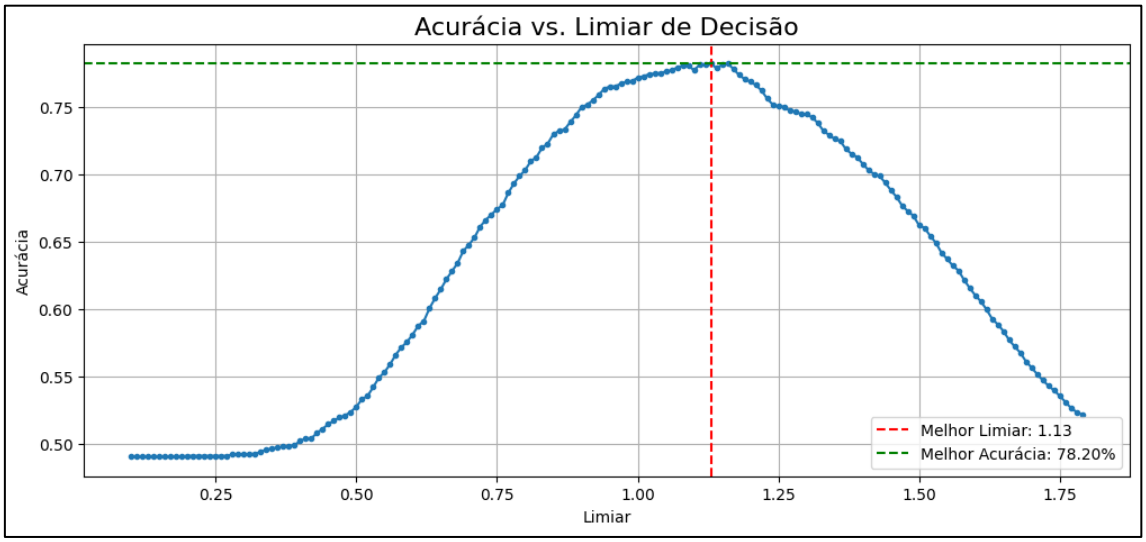
Após a fase de treinamento, o melhor modelo salvo foi submetido a uma avaliação de performance final, focada em sua capacidade de distinção em uma tarefa de classificação baseada em similaridade, provavelmente utilizando a função de perda *triplet* mencionada. O modelo atingiu uma acurácia máxima de 78,20% no conjunto de validação. Este resultado de acurácia é condicionado a um limiar de decisão (threshold) ótimo de 1.13. Em modelos baseados em *embedding*, esse limiar representa a distância no espaço de características que melhor separa os pares de amostras similares dos pares de amostras dissimilares. Portanto, o valor de 78,20% não reflete apenas a qualidade do espaço de características aprendido, mas também a eficácia deste limiar específico para a tarefa de classificação, estabelecendo um critério quantitativo para a aplicação prática do modelo.

Gráfico 5. *Loss* de Treino vs. Validação por Época.



Fonte: A autora, 2025.

Gráfico 6. Acurácia vs. Limiar de Decisão



Fonte: A autora, 2025.

5. Sistema web de reconhecimento facial

Ao término das etapas de treinamento, validação e otimização, o **modelo *triplet* com otimização estável de *pipeline*** emergiu como a solução mais eficaz para o núcleo do sistema de reconhecimento facial. Sua seleção foi rigorosamente fundamentada em métricas de desempenho superiores, notadamente a obtenção da menor perda (loss) no conjunto de dados de validação e o alcance da maior acurácia após a calibração do limiar de decisão. Este resultado validou o modelo como a escolha técnica ideal para prosseguir à fase de implementação.

Com a rede neural CNN definida, o projeto avança para a etapa de aplicação prática: a construção de um sistema web funcional e acessível. A principal decisão arquitetônica para esta fase foi a adoção do modelo Cliente/Servidor. Esta abordagem foi escolhida por ser a mais adequada para estruturar a comunicação entre a interface do usuário (cliente) e o processamento de dados, onde o modelo de reconhecimento facial opera (servidor).

A arquitetura Cliente/Servidor, conforme conceitua DA CRUZ (2025), é um modelo de processamento distribuído no qual múltiplos clientes e servidores interagem utilizando uma plataforma base, como o sistema operacional e os serviços de rede. Neste paradigma, ocorre uma cooperação intrínseca entre diferentes componentes de software e hardware, que se comunicam para executar a aplicação. Essa comunicação é majoritariamente realizada através do protocolo HTTP, mas pode envolver tecnologias como *APIs* e Web Services para trocas de dados mais complexas, como as exigidas pelo nosso sistema.

A relevância desta escolha arquitetônica é reforçada pelos benefícios práticos que ela oferece, os quais são cruciais para a longevidade e eficiência do projeto. De acordo com AWARI (2023), as principais vantagens incluem a modularidade, que foi essencial para permitir o desenvolvimento independente da interface do usuário e da lógica de reconhecimento; a escalabilidade, que garante que o sistema possa se adaptar a um aumento futuro no número de usuários ou de dados; e a facilidade de manutenção, decorrente da clara separação de responsabilidades entre as camadas do sistema.

5. 1 Implementação do sistema

O sistema de reconhecimento facial ponta a ponta implementado, utiliza uma arquitetura de rede neural siamesa para a geração de *embeddings* faciais e uma interface web construída com o framework *FastAPI* para interação em tempo real. O sistema é estruturado em seções lógicas que abrangem a configuração, as funções de processamento principais, a interface do usuário e os *endpoints* da API para registro e reconhecimento.

Passo 1. Configuração

A fase inicial de execução do sistema é dedicada à configuração do ambiente computacional e à instanciação dos recursos necessários. Este processo envolve a definição de parâmetros operacionais globais e o carregamento de ativos críticos, como o modelo de *deep learning* e os classificadores de visão computacional.

- Parâmetros e Caminhos: Constantes como *img_size*, *triplet_model_path* e threshold são definidas para garantir consistência entre o servidor e o modelo treinado. O *threshold* (1.03) é um hiperparâmetro crítico, determinado empiricamente durante a fase de avaliação do modelo, que define o limiar de distância para considerar duas faces como sendo da mesma pessoa. A criação da pasta pessoas (*people_dir*) estabelece uma estrutura de armazenamento persistente para as imagens de referência dos usuários cadastrados.
- Carregamento do Modelo Neural: O processo de inicialização carrega o modelo treinado (best_model.h5). Como o modelo foi salvo após ser treinado com uma função de perda customizada, a *triplet_loss*, é necessário passar um dicionário *custom_objects* para a função *load_model* do *Keras*, permitindo que o *TensorFlow* reconstrua a arquitetura corretamente. Subsequentemente, o script extrai a camada *EmbeddingModel*, que é o *backbone* de CNN responsável por gerar os vetores de características. Esta é uma etapa fundamental, pois a aplicação

não utiliza o modelo siamês completo para inferência, mas apenas seu componente extrator de *embeddings*.

- Carregamento de Dependências: O classificador em cascata *Haar* (haarcascade_frontalface_default.xml), um algoritmo clássico de visão computacional, é carregado via *OpenCV*. Sua função é detectar a localização de faces nas imagens recebidas, permitindo que apenas a região de interesse (o rosto) seja enviada para o modelo neural, o que melhora a precisão e a eficiência.

Passo 2. Gerenciamento do banco de dados em memória

Para o gerenciamento dos dados de referência, o sistema emprega uma abordagem de banco de dados em memória. Nesta arquitetura, os dados são carregados e mantidos na RAM para acesso de altíssima velocidade, enquanto o sistema de arquivos serve como mecanismo de persistência. Esta estratégia foi selecionada por otimizar a performance das consultas de reconhecimento, que são críticas em tempo real, e por simplificar a infraestrutura do protótipo, acelerando o desenvolvimento.

- Estrutura de Dados: A variável global *db_reference_images* atua como o banco de dados em memória. Ela armazena uma lista de tuplas, onde cada tupla contém o ID do usuário, o nome do arquivo de imagem e o *embedding* (vetor de características) correspondente àquela imagem.
- Função *reload_db_from_storage():* Esta função é o núcleo do sistema de gerenciamento de dados. Ela percorre a estrutura de diretórios da pasta pessoas, lê cada imagem de referência de cada usuário, passa a imagem pela função *get_embedding* para gerar seu vetor de características e popular a lista *db_reference_images*. Esta função é chamada na inicialização do servidor para carregar todos os usuários existentes e também após cada novo cadastro, garantindo que o banco de dados em memória esteja sempre atualizado e consistente com os dados persistidos no disco.

Passo 3. Definição das rotas da API

Esta seção detalha a camada de serviço da aplicação, onde a lógica de negócio é exposta através de *endpoints* HTTP. O mecanismo de roteamento do FastAPI é utilizado para associar funções Python a rotas de URL e métodos HTTP, um processo realizado por meio de decoradores. Cada função decorada, portanto, atua como um *handler* de

requisição, sendo responsável por executar uma funcionalidade específica do sistema em resposta a uma chamada de cliente.

- *Endpoint* de Reconhecimento (POST /*recognize*): Esta rota é projetada para receber um *array* de 3 imagens de teste. Primeiramente, ela valida a requisição para garantir que os dados estão no formato esperado. Em seguida, para cada imagem recebida, ela detecta a face e gera seu *embedding*. O núcleo da lógica de reconhecimento consiste em um loop aninhado que calcula a distância euclidiana (usando *np.linalg.norm*) entre cada um dos *embeddings* de teste e cada um dos *embeddings* de referência armazenados em *db_reference_images*. A rota mantém o registro da menor distância encontrada em todas as comparações. A classificação final é realizada pela comparação entre a menor distância global e o limiar de decisão. A identidade é confirmada se a distância apurada for inferior a este limiar.
- *Endpoint* de Cadastro (POST /register): Esta rota gerencia o processo de registro de um novo usuário, utilizando 3 imagens.
    1. Detecção e Extração: Primeiro, extrai os rostos das 3 imagens fornecidas.
    2. Verificação de Duplicidade: Antes de criar um novo usuário, o sistema executa uma verificação para evitar duplicatas. Ele calcula o *embedding* médio das 3 novas imagens e o compara com os *embeddings* de todos os usuários existentes para encontrar a menor distância. Se essa distância for menor que o *threshold*, a API retorna uma resposta indicando que o usuário provavelmente já está cadastrado.
    3. Criação do Usuário: Se a verificação de duplicidade for negativa, um novo ID de usuário único (UUID) é gerado. Uma nova pasta com este ID é criada dentro do diretório pessoas, e as 3 imagens de rosto recortadas são salvas nela.
    4. Atualização do Banco de Dados: Por fim, a função reload_db_from_storage() é chamada para que o novo usuário e seus *embeddings* sejam imediatamente carregados na memória e fiquem disponíveis para futuras requisições de reconhecimento, sem a necessidade de reiniciar o servidor.

Passo 4. Rotas da interface do usuário

As rotas mapeadas para os *endpoints* /*recognize* e /*register* constituem os pontos de acesso às funcionalidades interativas primárias do sistema.

A função associada a *@app.get("/recognize")* é responsável por servir o documento recognize.html, que encapsula a interface do cliente para iniciar a inferência de reconhecimento facial.

De forma análoga, a rota *@app.get("/register")* serve o register.html, provendo a interface para o processo de aquisição de dados biométricos de um novo usuário. Uma característica notável destas rotas é a injeção de estado do servidor no *template* do cliente. A variável *user_count*, que representa a cardinalidade do conjunto de usuários registrados, é passada como parte do contexto de *renderização* do Jinja2. Este mecanismo de *server-side rendering* parcial permite que a interface do usuário seja dinamicamente preenchida com informações contextuais do *back*-end, como o número total de perfis no banco de dados, enriquecendo a experiência do usuário sem a necessidade de uma requisição de API adicional e assíncrona.

5.2 Fluxo operacional do sistema

O fluxo operacional do sistema foi projetado como uma sequência determinística de interações e processamentos, garantindo um ciclo de verificação e registro de identidade robusto e intuitivo. A arquitetura do fluxo divide-se em duas jornadas principais: a de reconhecimento e a de cadastro, ambas iniciadas a partir da interface principal da aplicação. A seguir, descreve-se cada fase deste processo.

Jornada 1: fluxo de reconhecimento de identidade

Este fluxo é iniciado quando o usuário opta por realizar a verificação de sua identidade.

- Passo 1: Requisição de acesso ao dispositivo de captura
  Em conformidade com os protocolos de privacidade e como pré-requisito técnico, a aplicação web solicita ao navegador o consentimento explícito do usuário para acessar a câmera do dispositivo. A concessão desta permissão é mandatória para habilitar o hardware de captura de imagens.

- Passo 2: Início da captura sequencial

  A interação do usuário com o elemento de interface "REALIZAR RECONHECIMENTO" atua como o gatilho que inicia o processo de aquisição de dados biométricos. O sistema, então, executa a captura automática de três imagens sequenciais do rosto do usuário, com um intervalo pré-definido para permitir pequenas variações de pose e expressão.

- Passo 3: Pré-processamento e geração de *embeddings*

  Cada uma das três imagens capturadas é transmitida ao servidor. No *back-end*, cada imagem passa por uma etapa de pré-processamento que inclui a detecção da face via classificador *Haar Cascade* e o redimensionamento para as dimensões de entrada do modelo. Em seguida, o *backbone* da CNN (o *EmbeddingModel*) processa cada face, gerando três vetores de características distintos (*embeddings*) para a sessão de teste.

- Passo 4: Comparação e cálculo de distância mínima

  Neste passo, o sistema executa uma busca exaustiva. Cada um dos três *embeddings* de teste é comparado, através do cálculo da distância euclidiana, com todos os *embeddings* de referência armazenados no banco de dados. O sistema identifica e armazena a menor distância global encontrada em todo o espaço de comparação.

- Passo 5: Tomada de decisão e *feedback*

  A menor distância global calculada é comparada com o limiar de decisão (*threshold*) pré-definido.

  - Caso de Sucesso (Distância < Limiar): Se a distância for inferior ao limiar, a identidade do usuário é considerada verificada. O sistema o redireciona para uma página de boas-vindas (/*welcome*), frequentemente personalizada com o ID do usuário reconhecido, concluindo o fluxo com sucesso.
  - Caso de Falha (Distância ≥ Limiar): Se a distância for igual ou superior ao limiar, o sistema conclui que não há uma correspondência válida. O usuário é então redirecionado para uma página informativa (/*not_registered*), indicando que sua identidade não foi encontrada.

Jornada 2: Fluxo de registro de nova identidade

A funcionalidade de inclusão de novos perfis biométricos no banco de dados é iniciada a partir do acesso do usuário à página designada para o cadastro.

- Passo 6: Aquisição e verificação de duplicidade

  De forma análoga ao fluxo de reconhecimento, o sistema primeiramente captura três imagens sequenciais do novo usuário. Contudo, antes de proceder com o registro, essas imagens são submetidas à mesma lógica de reconhecimento descrita nos Passos 3 e 4. O objetivo desta verificação prévia é evitar o cadastro de identidades duplicadas. Se o sistema encontrar uma correspondência existente, o usuário é informado e redirecionado, interrompendo o fluxo de cadastro.

- Passo 7: Persistência dos dados de referência

  Apenas se a verificação de duplicidade resultar negativa, o sistema prossegue com o cadastro. Um Identificador Único Universal (UUID) é gerado para o novo usuário. Uma subpasta, nomeada com este UUID, é criada no diretório pessoas, e as três imagens de rosto capturadas são salvas neste local, servindo como os dados biométricos de referência persistidos.

- Passo 8: Sincronização

  Imediatamente após a persistência dos arquivos de imagem, o banco de dados em memória do servidor é atualizado para carregar os novos *embeddings*, tornando a identidade recém-cadastrada imediatamente disponível para futuras verificações. O usuário é então redirecionado para a página de boas-vindas (*/welcome*), recebendo um feedback positivo de que o registro foi concluído com sucesso e que o acesso foi concedido.

6. Implementação do sistema de reconhecimento facial utilizando Esp32-Cam

O sistema desenvolvido para a ESP32-CAM constitui uma evolução arquitetônica substancial em relação a implementações de modo único. A abordagem anterior, que fixava a funcionalidade do dispositivo em tempo de compilação, foi substituída por uma arquitetura multimodal reativa, controlada em tempo de execução. Nesta nova configuração, o dispositivo opera como um nó de captura de dados dentro do paradigma Cliente-Servidor, sendo agora capaz de alternar dinamicamente entre os processos de registro biométrico (cadastro) e verificação (reconhecimento). A seleção de funcionalidade é delegada a comandos do usuário, eliminando a necessidade de reprogramação do microcontrolador para cada tarefa. Essa flexibilidade arquitetônica

representa um avanço fundamental na prototipagem de um dispositivo de borda interativo e funcional.

Passo 1. Configuração

A seção de configuração e a função *setup()* estabelecem as bases operacionais do dispositivo, com algumas atualizações para suportar a nova lógica.

- Parâmetros operacionais: Além das credenciais de Wi-Fi e do endereço do servidor, novas constantes como fotos_necessarias, intervalo_fotos_ms e intervalo_reconhecimento_s foram introduzidas. Centralizar esses valores no topo do arquivo é uma prática de engenharia de software robusta, permitindo que o comportamento da captura (número de fotos, tempo entre elas) seja facilmente ajustado sem alterar a lógica principal do programa.
- Inicialização da câmera e Wi-Fi: O processo de inicialização do hardware da câmera e da conectividade Wi-Fi permanece o mesmo, sendo a base fundamental para a operação do dispositivo. Uma mudança notável é a configuração de $fb\_count = 2$, que aloca dois *framebuffers* para a câmera. Isso pode aumentar a estabilidade e a velocidade na captura de imagens sequenciais, prevenindo erros de alocação de memória durante a operação.
- Instruções para o usuário: Ao final da função *setup()*, o sistema agora imprime um bloco de instruções claras no monitor serial. Isso melhora significativamente a usabilidade, informando ao operador como interagir com o dispositivo para acionar as diferentes funcionalidades, um aspecto essencial para um sistema interativo.

Passo 2. Lógica de operação

A principal atualização arquitetônica neste firmware é a decomposição da lógica em funções especializadas e reutilizáveis, o que torna o código mais modular, legível e fácil de manter.

- *capture_multiple_images_as_base64_array()*: Esta função auxiliar abstrai a complexidade da captura sequencial de imagens. Sua responsabilidade única é executar um loop para capturar o número definido de fotos, codificar cada uma em Base64 e construir dinamicamente uma *string* formatada como um *array* JSON. Esta abstração evita a duplicação de código, já que tanto o cadastro quanto o reconhecimento necessitam desta mesma sequência de ações.

- *send_request_and_print_response()*: De forma similar, esta função encapsula toda a lógica de comunicação HTTP. Ela recebe como parâmetros o *endpoint* do servidor e o *payload* de imagens, e é responsável por montar a requisição POST, enviá-la e, crucialmente, processar e exibir a resposta do servidor. Isso centraliza o tratamento da comunicação em um único local.
- *realizar_cadastro()* e *realizar_reconhecimento()*: Estas são as duas funções de alto nível que orquestram as operações. Cada uma delas simplesmente chama as funções auxiliares na ordem correta, primeiro para capturar as imagens e depois para enviá-las ao *endpoint* específico de sua tarefa. Esta separação de responsabilidades é a chave para a funcionalidade *dual-mode*.

3. Loop principal e interação com o usuário

A função *loop()*, que é o coração do programa, foi transformada em um interpretador de comandos simples, aguardando ativamente por uma entrada do usuário para acionar uma ação.

- Lógica de Comando via Serial: O *loop()* verifica continuamente se há dados disponíveis na porta serial com *Serial.available()*. Quando o usuário envia um caractere pelo Monitor Serial, o código o lê e utiliza uma estrutura condicional para determinar qual ação tomar.
- Mapeamento de comandos: Os comandos foram mapeados de forma intuitiva: o caractere 'c' (ou 'C') aciona a função *realizar_cadastro(),* enquanto o caractere 'r' (ou 'R') aciona *realizar_reconhecimento()*. Este mecanismo simples transforma o Monitor Serial em uma interface de controle para o dispositivo.
- Modo de Operação Reativo: Diferentemente de um loop que executa uma tarefa continuamente, esta abordagem torna o dispositivo reativo, ou seja, ele permanece em estado de espera até que uma instrução explícita seja dada. Isso é ideal para testes e para cenários onde a operação não precisa ser contínua, economizando processamento e tráfego de rede. A presença de um bloco de código comentado para reconhecimento automático periódico também demonstra a flexibilidade da arquitetura, que poderia ser facilmente adaptada para um modo de vigilância contínua.

7. Considerações Finais

O presente trabalho propôs e validou uma solução para controle de acesso físico fundamentada em CNN, visando superar as limitações de sistemas de autenticação convencionais. A metodologia desenvolvida se baseia na aquisição de dados biométricos faciais, que são subsequentemente processados por um modelo de *deep learning*. Este modelo, previamente treinado com uma função de perda métrica, é capaz de extrair vetores de características de alta dimensionalidade que representam numericamente a identidade facial de forma única e discriminativa.

A autenticação é realizada comparando-se o *embedding* de um usuário em tempo real com um banco de dados de perfis previamente registrados, permitindo uma verificação de identidade automática, sem contato físico e independente de dispositivos tradicionais como cartões ou senhas. A aplicação de CNN confere ao sistema um grau elevado de robustez e precisão, demonstrando potencial para operar sob condições variáveis de iluminação, pose e expressão facial.

Apesar dos resultados promissores, a execução deste projeto encontrou limitações de ordem prática e computacional que devem ser consideradas na interpretação dos resultados. A principal restrição residiu no tempo computacional exigido para o treinamento dos modelos de *deep learning*. Utilizando um processador AMD *Ryzen* 5, um componente de uso geral sem aceleração por GPU dedicada para tarefas de aprendizado de máquina, cada época de treinamento demandou um tempo considerável, limitando o número total de épocas e a complexidade das arquiteturas que poderiam ser experimentalmente avaliadas dentro do cronograma do projeto.

Esta limitação de hardware impactou diretamente a capacidade de realizar uma exploração mais exaustiva do espaço de hiperparâmetros, como a variação da profundidade da rede ou a aplicação de técnicas de aumento de dados mais intensivas. Consequentemente, embora a arquitetura proposta seja válida, seu desempenho ótimo pode não ter sido alcançado, abrindo um caminho claro para futuras pesquisas que utilizem ambientes com Unidades de Processamento Gráfico (GPU), capazes de reduzir drasticamente o tempo de treinamento e permitir a experimentação em maior escala.

Dessa forma, esta pesquisa contribui significativamente para o campo de controle de acesso ao demonstrar a viabilidade de uma solução segura, eficiente e moderna. Os resultados obtidos constituem uma base empírica sólida que não apenas valida a abordagem arquitetônica, mas também serve como um ponto de partida para investigações futuras focadas em otimização, escalabilidade e implementação em

ambientes de produção. Dessa forma, o projeto favorece o avanço científico e tecnológico na interseção entre reconhecimento facial e sistemas de segurança física, alinhando-se às tendências de automação e inteligência artificial.

7.1 Trabalhos Futuros

A linha de pesquisa futura foca na aplicabilidade prática e na robustez do sistema em ambientes de produção. Para viabilizar a inferência diretamente em dispositivos de borda mais potentes, como uma *Raspberry Pi*, seria fundamental aplicar técnicas de quantização e converter o modelo para o formato *TensorFlow Lite* (*TFLite*). Essa otimização reduziria o tamanho do modelo e a latência de inferência, tornando a execução local viável. Adicionalmente, para mitigar vulnerabilidades de segurança, uma expansão crítica seria a implementação de um módulo de detecção de vivacidade (*liveness detection*). Esta funcionalidade garantiria que o sistema é capaz de distinguir entre um rosto real e uma tentativa de fraude utilizando uma foto ou vídeo, um requisito indispensável para qualquer sistema de controle de acesso físico seguro.

REFERÊNCIAS

AHMADI, Reza et al. Study of artificial neural networks in information security risk assessment. *Journal of Management and Accounting Studies*, v. 8, n. 2, p. 1-10, 2020.

ANDREJEVIC, Mark; SELWYN, Neil. **Facial recognition**. [S.l.]: John Wiley & Sons, 2022.

ANWARUL, Shahina; DAHIYA, Susheela. A comprehensive review on face recognition methods and factors affecting facial recognition accuracy. *In*: ICRIC 2019: RECENT INNOVATIONS IN COMPUTING, 2020. **Proceedings [...]**. 2020. p. 495-514.

ARRUDA, Cícero Gilma de Oliveira. **4usecurity: reconhecimento facial para controle de acesso**. 2020. Monografia (Curso não especificado) - Instituição não especificada, 2020.

AWARI. Arquitetura de Software Cliente-Servidor: Estrutura e Interação entre os Componentes. *Awari*, 2023. Disponível em: https://awari.com.br/arquitetura-de-software-cliente-servidor-estrutura-e-interacao-entre-os-componentes/. Acesso em: 27 abr. 2025.

AWARI. Deep Learning RNN – Utilizando redes RNN (Recurrent Neural Networks) no Deep Learning. *Awari*, 2023. Disponível em: https://awari.com.br/deep-learning-rnn-utilizando-redes-rnnrecurrent-neural-networks-no-deep-learning/. Acesso em: 17 out. 2024.

BARBOSA, G. et al. Segurança em redes 5g: Oportunidades e desafios em detecção de anomalias e predição de tráfego baseadas em aprendizado de máquina. *In*: SIMPÓSIO BRASILEIRO DE SEGURANÇA DA INFORMAÇÃO E DE SISTEMAS COMPUTACIONAIS, 21., 2021, Online. **Anais [...]**. Belém, 2021. p. 4-7.

BARINO, Felipe Oliveira; DOS SANTOS, Alexandre Bessa. Rede neural convolucional 1d aplicada a previsão da vazão no rio madeira. *In*: SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES E PROCESSAMENTO DE SINAIS, 38., 2020. **Anais [...]**. 2020.

BAX, Marcello Peixoto. Design science: filosofia da pesquisa em ciência da informação e tecnologia. *Ciência da Informação*, v. 42, n. 2, 2013.

BORGES, Robson Pires; BORGES, Ronaldo Pires. Controle de acesso e automação de ambientes com Artificial Intelligence of Things. *In*: ENCONTRO UNIFICADO DE COMPUTAÇÃO DO PIAUÍ (ENUCOMPI), 2023. **Anais [...]**. SBC, 2023. p. 121-129.

CLUBEDOGIS. O que é: Distância Euclidiana. *Clube do GIS*, 2025. Disponível em: https://clubedogis.com.br/glossario/o-que-e-distancia-euclidiana-definicao-e-aplicacoes/. Acesso em: 27 abr. 2025.

COELHO, Lucíola. Inverno da IA: uma perspectiva sobre os períodos de estagnação na Inteligência Artificial. *LinkedIn*, 2023. Disponível em: https://pt.linkedin.com/pulse/inverno-da-ia-uma-perspectiva-sobre-os-per%C3%ADodos-dena-luc%C3%ADola-coelho-9prhf. Acesso em: 17 out. 2024.

COSTA, Bruno Carlos da C. et al. Desenvolvimento de Software Educacional para Representação e Reconhecimento de Som Aplicado à Ausculta Cardiovascular. *Revista Eletrônica TECCEN*, v. 2, n. 1, p. 17-26, 2009.

DA CRUZ, I. M. **Fundamentos da Arquitetura Cliente/Servidor**. [S.l.: s.n.], [s.d.]. Disponível em: https://www.academia.edu/8212738/Fundamentos_da_Arquitetura_Cliente_Servidor. Acesso em: 11 jun. 2025.

DATACAMP. Perceptrons multicamadas em aprendizado de máquina: um guia abrangente. *DataCamp*, 24 abr. 2024. Disponível em: https://www.datacamp.com/pt/tutorial/multilayer-perceptrons-in-machinelearning. Acesso em: 25 out. 2024.

DATA SCIENCE ACADEMY. O Neurônio Biológico e Matemático. *Deep Learning Book*, 21 jun. 2020. Disponível em: https://www.deeplearningbook.com.br/o-neuronio-biologico-e-matematico/. Acesso em: 21 jul. 2025.

DATA SCIENCE ACADEMY. O Perceptron – Parte 2. *Deep Learning Book*, 2022. Disponível em: https://www.deeplearningbook.com.br/o-perceptron-parte-2/. Acesso em: 5 nov. 2023.

DESCOMPLICA.IA. Conjunto de Dados: entenda sua importância na IA. *Descomplica.IA*, 2024. Disponível em: https://descomplicaia.com.br/glossario/conjunto-de-dados-importancia-ia/. Acesso em: 3 jul. 2025.

ENTENDA como funcionava esquema que fraudou perfis em plataforma do governo federal. *G1*, Brasília, 14 maio 2025. Disponível em: https://g1.globo.com/df/distrito-federal/noticia/2025/05/14/entenda-como-funcionava-esquema-que-fraudou-perfis-em-plataforma-do-governo-federal.ghtml. Acesso em: 20 jul. 2025.

EQUIPE ESTATÍSTICA FÁCIL. O que é: Rede Neural Feedforward. *Estatística Fácil*, [s.d.]. Disponível em: https://estatisticafacil.org/glossario/o-que-e-rede-neural-feedforward-entenda-aqui/. Acesso em: 21 jul. 2025.

FLECK, Leandro et al. Redes neurais artificiais: princípios básicos. *Revista Eletrônica Científica Inovação e Tecnologia*, v. 1, n. 13, p. 47-57, 2016.

GARCÍA, Jesús L. Llano; MONROY, Raúl; HERNÁNDEZ, Víctor Adrián Sosa. Neural architecture search for image super-resolution: a review on the emerging state-of-the-art. *Neurocomputing*, p. 128481, 2024.

GRUPO SIDE. Controle de acesso: como otimizar esse serviço?. *LinkedIn*, 2025. Disponível em: https://www.linkedin.com/pulse/controle-de-acesso-l%C3%B3gico-x-f%C3%ADsico-j%C3%A9ssica-guimar%C3%A3es/. Acesso em: 26 abr. 2025.

GUIMARÃES, Alexandre Magnus Fernandes; FERREIRA, Rhendson Alexandre. Desenvolvimento de um sistema de controle de acesso de baixo custo por reconhecimento facial. *In*: **Engenharia, Gestão e Inovação**. v. 6, p. 71, [s.d.].

GUIMARÃES, Jéssica. Controle de acesso lógico x Controle de acesso físico. *LinkedIn*, 2025. Disponível em: https://www.linkedin.com/pulse/controle-de-acesso-como-otimizar-esse-servi%C3%A7o-q1a0f/. Acesso em: 26 abr. 2025.

GÜLEN, Kerem. Erro absoluto médio (mae). *DataconomyPT*, 2025. Disponível em: https://pt.dataconomy.com/2025/03/28/erro-absoluto-medio-mae/. Acesso em: 27 abr. 2025.

HAYKIN, Simon. **Redes neurais: princípios e práticas**. 2. ed. São Paulo: Bookman, 2001.

IBM. O que são redes neurais convolucionais?. *IBM*, [s.d.]. Disponível em: https://www.ibm.com/topics/convolutional-neural-networks. Acesso em: 21 jul. 2025.

IBM. O que são redes neurais recorrentes?. *IBM*, [s.d.]. Disponível em: https://www.ibm.com/br-pt/topics/recurrent-neural-networks. Acesso em: 2 set. 2024.

IJIRI, Yoshihisa; SAKURAGI, Miharu; LAO, Shihong. Security Management for Mobile Device by Face Recognition. *In*: EVENTO NÃO ESPECIFICADO, 2006. **Anais [...]**. 2006.

ISHIDA, Tales Hiro Cardoso. **Sistema de controle de acesso por reconhecimento facial utilizando sistemas embarcados**. 2023. Trabalho de Conclusão de Curso (Curso não especificado) - Instituição não especificada, 2023.

KASPERSKY. O que é reconhecimento facial – definição e explicação. *Kaspersky*, 2024. Disponível em: https://www.kaspersky.com.br/resource-center/definitions/what-is-facial-recognition. Acesso em: 15 out. 2024.

KAUR, Paramjit et al. Facial-recognition algorithms: a literature review. *Medicine, Science and the Law*, v. 60, n. 2, p. 131-139, 2020.

KSHIRSAGAR, V. P.; BAVISKAR, M. R.; GAIKWAD, M. E. Face recognition using Eigenfaces. *In*: 3RD INTERNATIONAL CONFERENCE ON COMPUTER RESEARCH AND DEVELOPMENT, 2011. **Anais [...]**. IEEE, 2011. p. 302-306.

LEMOS, R. P. et al. Sistema de controle de acesso através de reconhecimento facial com monitoramento remoto. *Revista Brasileira Militar de Ciências*, v. 10, n. 24, 2024.

LI, Jessica. CelebFaces Attributes (CelebA) Dataset. *Kaggle*, [s.d.]. Disponível em: https://www.kaggle.com/datasets/jessicali9530/celeba-dataset. Acesso em: 21 jul. 2025.

LIU, Xiaobin et al. The application and influencing factors of computer vision: focus on human face recognition in medical field. *Наука, образование, инновации: актуальные вопросы и современные аспекты*, p. 32-37, 2022.

MAIA, Deise; TRINDADE, Roque. Face detection and recognition in color images under matlab. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, v. 9, n. 2, p. 13-24, 2016.

MARIUTTI, Eduardo Barros. Inteligência Artificial Simbólica e Conexionista. *Observatório de Geopolítica do Jornal GGN*, 3 abr. 2023. Disponível em: https://jornalggn.com.br/geopolitica/inteligencia-artificial-simbolica-econexionista-por-eduardo-barros-mariutti/. Acesso em: 17 out. 2024.

MORIGGI FILHO, Eduardo. **Controle de acesso por reconhecimento facial e controle de ambientes**. 2018. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) - Faculdade de Tecnologia de Americana, Americana, 2018.

NAGASAI524. Facial Keypoint-Detection. *Kaggle*, [s.d.]. Disponível em: https://www.kaggle.com/datasets/nagasai524/facial-keypointdetection. Acesso em: 18 nov. 2024.

OLIVEIRA, Thiago. O que é processamento digital de imagens?. *Pix Force*, 7 jul. 2022. Disponível em: https://pixforce.ai/pt-br/o-que-e-processamento-digital-de-imagens/. Acesso em: 31 ago. 2024.

PALMIERE, Sérgio Eduardo. Rede Perceptron de uma única camada. *Embarcados*, 9 set. 2016. Disponível em: https://embarcados.com.br/rede-perceptron-de-uma-unica-camada/. Acesso em: 21 jul. 2025.

PINHEIRO, Nina. Pré-processamento de dados com Python. *Medium*, 7 ago. 2021. Disponível em: https://medium.com/data-hackers/pr%C3%A9-processamento-de-dados-compython-53b95bcf5ff4. Acesso em: 25 out. 2024.

PURE STORAGE. O que são métricas de desempenho de aprendizado de máquina?. *Pure Storage*, 2024. Disponível em: https://www.purestorage.com/br/knowledge/machine-learning-performance-metrics.html. Acesso em: 24 jul. 2025.

RODRIGUES, Diogo Francisco Borba; LOPES, Ezequiel Gomes. Técnicas de reconhecimento facial com Machine Learning. *In*: **Engenharias: qualidade, produtividade e inovação tecnológica**. [S.l.]: Atena Editora, 2023. p. 266-295. Disponível em: https://atenaeditora.com.br/catalogo/ebook/engenharias-qualidade-produtividade-e-inovacao-tecnologica. Acesso em: 27 abr. 2025.

SANTOS, Antony Kevin Ferreira. **Um estudo comparativo envolvendo classificadores e técnicas de redução de dimensionalidade aplicadas ao reconhecimento facial**. 2017. 53 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade Estadual do Ceará, Fortaleza, 2017. Disponível em: http://siduece.uece.br/siduece/trabalhoAcademicoPublico.jsf?id=86028. Acesso em: 27 abr. 2025.

SANTOS, L. A. F. d; GULO, C. A. Segurança da informação. *Repositório Institucional da UFSCar*, 2018.

SANTOS, Vagner Vieira dos et al. **Sistema de reconhecimento facial com base em técnicas de aprendizado de máquina**. 2022. Trabalho de Conclusão de Curso (Curso não especificado) - Instituição não especificada, 2022.

SICHMAN, Jaime Simão. Inteligência Artificial e sociedade: avanços e riscos. *Estudos Avançados*, v. 35, p. 37-50, 2021.

SILVA, Alex Lima. **Redução de características para classificação de imagens de faces**. 2016. Dissertação (Mestrado) - Universidade Federal Rural do Semi-Árido, 2016.

SILVA, J. B. **Explorando o algoritmo de Viola-Jones na detecção e reconhecimento facial**. 2018. 91 f. Trabalho de Conclusão de Curso (Engenharia da Computação) - Universidade Federal de São Carlos, São Carlos, 2018.

SILVA, Rosane Leal da; SILVA, Fernanda dos Santos Rodrigues da. Reconhecimento facial e segurança pública: os perigos do uso da tecnologia no sistema penal seletivo brasileiro. *In*: CONGRESSO INTERNACIONAL DE DIREITO E CONTEMPORANEIDADE, 2019, Santa Maria. **Anais [...]**. Santa Maria, RS, 2019.

SINGH, Gurpreet et al. Face recognition using open source computer vision library (OpenCV) with Python. *In*: 10TH INTERNATIONAL CONFERENCE ON RELIABILITY, INFOCOM TECHNOLOGIES AND OPTIMIZATION (TRENDS AND FUTURE DIRECTIONS) (ICRITO), 2022. **Anais [...]**. IEEE, 2022. p. 1-6.

SOARES, Pablo Luiz Braga; DA SILVA, José Patrocínio. Aplicação de redes neurais artificiais em conjunto com o método vetorial da propagação de feixes na análise de um acoplador direcional baseado em fibra ótica. *Revista Brasileira de Computação Aplicada*, v. 3, n. 2, p. 58-72, 2011.

SORAGGI, Cristiano. Gestão de Acesso: Princípios, Desafios, Melhores Práticas, Aplicabilidade. *LinkedIn*, 2024. Disponível em: https://www.linkedin.com/pulse/gest%C3%A3o-de-acesso-princ%C3%ADpios-desafios-melhores-pr%C3%A1ticas-soraggi-knt6f/?originalSubdomain=pt. Acesso em: 25 mar. 2025.

TRAINA, Agma Juci Machado; DE OLIVEIRA, Maria Cristina Ferreira. **Apostila de Computação Gráfica**. 2003.

TREVISAN, Vinícius. Como funcionam as Redes Neurais Convolucionais (CNNs). *Medium*, 15 dez. 2021. Disponível em: https://medium.com/data-hackers/como-funcionam-as-redes-neurais-convolucionais-cnns-71978185c1. Acesso em: 3 jul. 2025.

VARGAS, Ana Caroline Gomes; PAES, Aline; VASCONCELOS, Cristina Nader. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. *In*: CONFERENCE ON GRAPHICS, PATTERNS AND IMAGES, 29., 2016. **Anais [...]**. 2016.

APÊNDICES

```
from tensorflow.keras import layers, Model
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random
import os

# ==============================================================================
# 1. CONFIGURAÇÕES - REDE (ESTILO TRIPLET)
# ==============================================================================
print("\n--- PASSO 1: CONFIGURAÇÕES - REDE (ESTILO TRIPLET) ---")

DATASET_PATH = ‘cnn/dataset'
MODEL_CHECKPOINT_PATH = 'checkpoints/best_model.h5'

IMG_SIZE = (96, 96)
BATCH_SIZE = 32
STEPS_PER_EPOCH = 150
VALIDATION_STEPS = 50
EMBEDDING_DIM = 128
EPOCHS = 30
MARGIN = 0.5
VALIDATION_SPLIT = 0.2

if not os.path.isdir(DATASET_PATH):
    raise ValueError(f"ERRO: O diretório '{DATASET_PATH}' não foi encontrado.")
os.makedirs(os.path.dirname(MODEL_CHECKPOINT_PATH), exist_ok=True)
print(f"Dataset encontrado em: {DATASET_PATH}")
print(f"Checkpoints serão salvos em: {os.path.dirname(MODEL_CHECKPOINT_PATH)}")


# ==============================================================================
# 2. PREPARAÇÃO DOS DADOS (COM CORREÇÃO NO PIPELINE)
# ==============================================================================
print("\n--- PASSO 2: PREPARANDO PIPELINE DE DADOS ---")

def preprocess_image_original(path):
    img = tf.io.read_file(path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, IMG_SIZE)
    img = img / 255.0
    return img

all_person_ids = [d for d in os.listdir(DATASET_PATH) if os.path.isdir(os.path.join(DATASET_PATH,
d))]
```

```
random.shuffle(all_person_ids)
split_index = int(len(all_person_ids) * (1 - VALIDATION_SPLIT))
train_person_ids = all_person_ids[:split_index]
validation_person_ids = all_person_ids[split_index:]

def create_path_dictionary(person_id_list):
    path_dict = {}
    for person_id in person_id_list:
        person_path = os.path.join(DATASET_PATH, person_id)
        files = [os.path.join(person_path, f) for f in os.listdir(person_path)]
        if len(files) >= 2:
            path_dict[person_id] = files
    return path_dict

train_paths = create_path_dictionary(train_person_ids)
validation_paths = create_path_dictionary(validation_person_ids)
print(f"Dataset dividido: {len(train_paths)} pessoas para treino, {len(validation_paths)} para validação.")

def triplet_path_generator(path_dict):
    person_list = list(path_dict.keys())
    while True:
        anchor_person_id = random.choice(person_list)
        while len(path_dict.get(anchor_person_id, [])) < 2:
            anchor_person_id = random.choice(person_list)

        anchor_path, positive_path = random.sample(path_dict[anchor_person_id], 2)

        negative_person_id = random.choice(person_list)
        while negative_person_id == anchor_person_id:
            negative_person_id = random.choice(person_list)
        negative_path = random.choice(path_dict[negative_person_id])

        # Apenas retorna os caminhos e uma label dummy
        yield (anchor_path, positive_path, negative_path), 0.0

output_signature_paths = (
    (tf.TensorSpec(shape=(), dtype=tf.string),
     tf.TensorSpec(shape=(), dtype=tf.string),
     tf.TensorSpec(shape=(), dtype=tf.string)),
    tf.TensorSpec(shape=(), dtype=tf.float32)
)

# Esta função recebe os caminhos e aplica o pré-processamento.
def load_and_preprocess_triplets(paths, label):
    anchor_path, positive_path, negative_path = paths
    anchor_img = preprocess_image_original(anchor_path)
    positive_img = preprocess_image_original(positive_path)
    negative_img = preprocess_image_original(negative_path)
    return (anchor_img, positive_img, negative_img), label
```

```
train_dataset = tf.data.Dataset.from_generator(
    lambda: triplet_path_generator(train_paths),
    output_signature=output_signature_paths
).map(
    load_and_preprocess_triplets,
    num_parallel_calls=tf.data.AUTOTUNE
).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

validation_dataset = tf.data.Dataset.from_generator(
    lambda: triplet_path_generator(validation_paths),
    output_signature=output_signature_paths
).map(
    load_and_preprocess_triplets,
    num_parallel_calls=tf.data.AUTOTUNE
).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)

print("Pipeline de dados (corrigido com .map()) criado com sucesso.")

# ==============================================================================
# 3. CONSTRUÇÃO DO MODELO
# ==============================================================================
print("\n--- PASSO 3: CONSTRUINDO O MODELO ---")
def build_embedding_model(input_shape, embedding_dim):
    initializer = 'he_uniform'
    inputs = layers.Input(shape=input_shape)
    x = layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer=initializer)(inputs)
    x = layers.Conv2D(64, (3, 3), padding='same', activation='relu', kernel_initializer=initializer)(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Dropout(0.3)(x)
    x = layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_initializer=initializer)(x)
    x = layers.Conv2D(128, (3, 3), padding='same', activation='relu', kernel_initializer=initializer)(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Dropout(0.4)(x)
    x = layers.Conv2D(256, (3, 3), padding='same', activation='relu', kernel_initializer=initializer)(x)
    x = layers.BatchNormalization()(x)
    x = layers.MaxPooling2D(pool_size=(2, 2))(x)
    x = layers.Dropout(0.4)(x)
    x = layers.Flatten()(x)
    x = layers.Dense(embedding_dim, activation=None, kernel_initializer=initializer)(x)
    x = layers.UnitNormalization(axis=-1)(x)
    return Model(inputs, x, name="EmbeddingModel")

embedding_model = build_embedding_model(IMG_SIZE + (3,), EMBEDDING_DIM)
input_anchor = layers.Input(shape=IMG_SIZE + (3,), name="input_anchor")
input_positive = layers.Input(shape=IMG_SIZE + (3,), name="input_positive")
input_negative = layers.Input(shape=IMG_SIZE + (3,), name="input_negative")
```

```
processed_a = embedding_model(input_anchor)
processed_p = embedding_model(input_positive)
processed_n = embedding_model(input_negative)

merged_vector = layers.Concatenate(axis=-1, name="merged_embeddings")([processed_a, processed_p,
processed_n])
triplet_model = Model(inputs=[input_anchor, input_positive, input_negative], outputs=merged_vector)

# ================================================================================
# 4. COMPILAÇÃO DO MODELO
# ================================================================================
print("\n--- PASSO 4: COMPILANDO O MODELO COM TRIPLET LOSS ---")
def triplet_loss(y_true, y_pred):
    anchor, positive, negative = y_pred[:, :EMBEDDING_DIM], y_pred[:,
EMBEDDING_DIM:2*EMBEDDING_DIM], y_pred[:, 2*EMBEDDING_DIM:]
    positive_dist = tf.reduce_sum(tf.square(anchor - positive), axis=-1)
    negative_dist = tf.reduce_sum(tf.square(anchor - negative), axis=-1)
    loss = tf.maximum(0.0, positive_dist - negative_dist + MARGIN)
    return tf.reduce_mean(loss)

optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
triplet_model.compile(loss=triplet_loss, optimizer=optimizer)
triplet_model.summary()

# ================================================================================
# 5. TREINAMENTO
# ================================================================================
print("\n--- PASSO 5: INICIANDO TREINAMENTO ---")
checkpoint = ModelCheckpoint(MODEL_CHECKPOINT_PATH, monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, verbose=1, min_lr=1e-6)

history = triplet_model.fit(
    train_dataset,
    steps_per_epoch=STEPS_PER_EPOCH,
    epochs=EPOCHS,
    verbose=1,
    validation_data=validation_dataset,
    validation_steps=VALIDATION_STEPS,
    callbacks=[checkpoint, reduce_lr]
)
print("\n--- TREINAMENTO CONCLUÍDO ---")

# ================================================================================
# 6. AVALIAÇÃO DO MELHOR MODELO (sem alterações)
# ================================================================================
print("\n--- PASSO 6: AVALIANDO O MODELO DE EMBEDDING ---")
```

```
best_triplet_model = tf.keras.models.load_model(MODEL_CHECKPOINT_PATH,
custom_objects={'triplet_loss': triplet_loss})
embedding_model_eval = best_triplet_model.get_layer('EmbeddingModel')

def create_evaluation_pair_generator(path_dict):
    person_list = list(path_dict.keys())
    while True:
        try:
            if random.random() < 0.5:
                person_id = random.choice(person_list)
                if len(path_dict[person_id]) < 2: continue
                img1_path, img2_path = random.sample(path_dict[person_id], 2)
                label = 1.0
            else:
                p1_id, p2_id = random.sample(person_list, 2)
                img1_path = random.choice(path_dict[p1_id])
                img2_path = random.choice(path_dict[p2_id])
                label = 0.0

            img1 = preprocess_image_original(img1_path)
            img2 = preprocess_image_original(img2_path)
            yield (img1, img2), label
        except Exception:
            continue

output_signature_eval = ((tf.TensorSpec(shape=IMG_SIZE+(3,), dtype=tf.float32),
tf.TensorSpec(shape=IMG_SIZE+(3,), dtype=tf.float32)), tf.TensorSpec(shape=(), dtype=tf.float32))
eval_dataset = tf.data.Dataset.from_generator(lambda:
create_evaluation_pair_generator(validation_paths),
output_signature=output_signature_eval).batch(2000).take(1).cache()

for (img1_batch, img2_batch), labels_batch in eval_dataset:
    embeddings1 = embedding_model_eval.predict(img1_batch)
    embeddings2 = embedding_model_eval.predict(img2_batch)
    distances = np.sqrt(np.sum(np.square(embeddings1 - embeddings2), axis=1))
    labels = labels_batch.numpy()

thresholds = np.arange(0.1, 1.8, 0.01)
accuracies = [np.mean(((distances < t) == labels).astype(float)) for t in thresholds]
best_threshold_index = np.argmax(accuracies)
best_threshold = thresholds[best_threshold_index]
best_accuracy = accuracies[best_threshold_index]

print("\n" + "="*50); print("         AVALIAÇÃO FINAL DE PERFORMANCE (TRIPLET)");
print("="*50)
print(f"Acurácia Máxima no Conjunto de Validação: {best_accuracy:.4f} ({best_accuracy*100:.2f}%)")
print(f"Limiar Ideal para esta Acurácia: {best_threshold:.2f}"); print("="*50)

# ========================================================================
```

```
# 6.1. SALVANDO O MODELO DE EMBEDDING FINAL
# ==============================================================================
print("\n--- PASSO 6.1: SALVANDO O MODELO DE EMBEDDING FINAL ---")
final_embedding_model_path = os.path.join(os.path.dirname(MODEL_CHECKPOINT_PATH),
'final_embedding_model.h5')
embedding_model_eval.save(final_embedding_model_path)
print(f"Modelo de embedding final (para uso em produção) salvo com sucesso em:
{final_embedding_model_path}")

# ==============================================================================
# 7. GRÁFICOS
# ==============================================================================
print("\n--- PASSO 7: GERANDO GRÁFICOS DE ANÁLISE ---")
plt.figure(figsize=(12, 5))
plt.plot(history.history['loss'], label='Loss de Treinamento')
plt.plot(history.history['val_loss'], label='Loss de Validação', linewidth=2)
if 'val_loss' in history.history:
    best_epoch = np.argmin(history.history['val_loss']) + 1
    plt.axvline(best_epoch - 1, color='r', linestyle='--', label=f'Melhor Época: {best_epoch}')
plt.title('Loss de Treino vs. Validação por Época', fontsize=16)
plt.xlabel('Época')
plt.ylabel('Triplet Loss')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 5))
plt.plot(thresholds, accuracies, marker='.')
plt.axvline(best_threshold, color='r', linestyle='--', label=f'Melhor Limiar: {best_threshold:.2f}')
plt.axhline(best_accuracy, color='g', linestyle='--', label=f'Melhor Acurácia: {best_accuracy*100:.2f}%')
plt.title('Acurácia vs. Limiar de Decisão', fontsize=16)
plt.ylabel('Acurácia')
plt.xlabel('Limiar')
plt.legend()
plt.grid(True)
plt.show()
```

Server do sistema web de reconhecimento facial:

```
import os
import uuid
import base64
import shutil
import cv2
import numpy as np
import math
from fastapi import FastAPI, Request, Body
from fastapi.responses import HTMLResponse, RedirectResponse, JSONResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
import tensorflow as tf
from tensorflow.keras.models import load_model

#================================================================================
# 1. CONFIGURAÇÃO E CARREGAMENTO
#================================================================================

# --- PARÂMETROS GERAIS ---
IMG_SIZE = 96
TRIPLET_MODEL_PATH = "best_model_Original.h5"
THRESHOLD = 1.03

# --- CAMINHOS ---
CASCADE_PATH = cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
PEOPLE_DIR = "pessoas"

os.makedirs(PEOPLE_DIR, exist_ok=True)


# --- FUNÇÃO CUSTOMIZADA PARA CARREGAR O MODELO ---
def triplet_loss(y_true, y_pred):
    embedding_dim = 128
    margin = 0.5
    anchor, positive, negative = y_pred[:, :embedding_dim], y_pred[:,
embedding_dim:2*embedding_dim], y_pred[:, 2*embedding_dim:]
    positive_dist = tf.reduce_sum(tf.square(anchor - positive), axis=-1)
    negative_dist = tf.reduce_sum(tf.square(anchor - negative), axis=-1)
    loss = tf.maximum(0.0, positive_dist - negative_dist + margin)
    return tf.reduce_mean(loss)

custom_objects = {"triplet_loss": triplet_loss}

# --- CARREGA MODELO E CASCADE ---
try:
    print("Carregando o modelo Triplet completo...")
    if not os.path.exists(TRIPLET_MODEL_PATH):
        raise FileNotFoundError(f"Arquivo do modelo não encontrado: {TRIPLET_MODEL_PATH}")
    triplet_model = load_model(TRIPLET_MODEL_PATH, custom_objects=custom_objects)

    print("Extraindo o modelo de backbone (gerador de embedding)...")
    embedding_model = triplet_model.get_layer('EmbeddingModel')

    print("Carregando Haar Cascade para detecção de rosto...")
    face_cascade = cv2.CascadeClassifier(CASCADE_PATH)
    if face_cascade.empty():
        raise RuntimeError("Falha ao carregar Haar Cascade")
except Exception as e:
```

```
    raise RuntimeError(f"Erro na inicialização: {e}")

#=============================================================================
# 2. FUNÇÕES AUXILIARES E BANCO DE DADOS EM MEMÓRIA
#=============================================================================

def get_embedding(face_bgr: np.ndarray) -> np.ndarray:
    face_rgb = cv2.cvtColor(face_bgr, cv2.COLOR_BGR2RGB)
    face_resized = cv2.resize(face_rgb, (IMG_SIZE, IMG_SIZE))
    face_normalized = face_resized.astype("float32") / 255.0
    embedding = embedding_model.predict(face_normalized[None], verbose=0)[0]
    return embedding

db_reference_images = []

def reload_db_from_storage():
    global db_reference_images
    db_reference_images = []
    print("Recarregando banco de dados de referência a partir da pasta 'pessoas'...")
    user_ids = [d for d in os.listdir(PEOPLE_DIR) if os.path.isdir(os.path.join(PEOPLE_DIR, d))]
    for user_id in user_ids:
        user_dir = os.path.join(PEOPLE_DIR, user_id)
        for img_name in os.listdir(user_dir):
            img_path = os.path.join(user_dir, img_name)
            img = cv2.imread(img_path)
            if img is not None:
                embedding = get_embedding(img)
                db_reference_images.append((user_id, img_name, embedding))
    print(f"[INFO] {len(db_reference_images)} imagens de referência carregadas para {len(user_ids)}
usuários.")
    return len(user_ids)

# Carrega o banco de dados na inicialização
user_count = reload_db_from_storage()
print(f"[INFO] Usando limiar de reconhecimento fixo: {THRESHOLD}")

#=============================================================================
# 3. SETUP DA APLICAÇÃO FASTAPI
#=============================================================================
app = FastAPI(title="Reconhecimento Facial")
app.mount("/static", StaticFiles(directory="static"), name="static")
templates = Jinja2Templates(directory="templates")

#=============================================================================
# 4. ROTAS DA INTERFACE (PÁGINAS HTML) - ATUALIZADO
#=============================================================================

# ——— ROTA RAIZ ————
@app.get("/", response_class=HTMLResponse)
async def home_page(request: Request):
    """Serve a página inicial/explicativa do sistema."""
    return templates.TemplateResponse("index.html", {"request": request})


# ——— PÁGINAS ESTÁTICAS ————
@app.get("/recognize", response_class=HTMLResponse)
def recognize_page(request: Request):
    """Serve a página de reconhecimento facial."""
    user_count = len(os.listdir(PEOPLE_DIR))
    return templates.TemplateResponse("recognize.html", {"request": request, "user_count": user_count})
```

```
@app.get("/not_registered", response_class=HTMLResponse)
def not_registered_page(request: Request):
    return templates.TemplateResponse("not_registered.html", {"request": request})

@app.get("/register", response_class=HTMLResponse)
def register_page(request: Request):
    # ### ATUALIZAÇÃO ###: Passa o número de usuários cadastrados para o template.
    user_count = len(os.listdir(PEOPLE_DIR))
    return templates.TemplateResponse("register.html", {"request": request, "user_count": user_count})

@app.get("/welcome", response_class=HTMLResponse)
def welcome_page(request: Request, user_id: str = "Convidado"):
    # ### ATUALIZAÇÃO ###: Pode passar o user_id para personalização.
    return templates.TemplateResponse(
        "welcome.html",
        {"request": request, "user_id": user_id}
    )

# ——— PÁGINA DE AJUDA ——————————
@app.get("/help", response_class=HTMLResponse)
async def help_page(request: Request):
    return templates.TemplateResponse("help.html", {"request": request})


#=============================================================================
# 5. ROTAS DA API
#=============================================================================

@app.post("/recognize")
async def api_recognize(data: dict = Body(...)):
    test_images_b64 = data.get("images")
    if not test_images_b64 or not isinstance(test_images_b64, list) or len(test_images_b64) != 3:
        return JSONResponse({"success": False, "error": "Requisição deve conter uma lista de 3 imagens."},
status_code=400)

    if not db_reference_images:
        return JSONResponse({"success": True, "decision": "Nenhum usuário cadastrado. Faça um cadastro
primeiro."}, status_code=200)

    test_embeddings = []
    for img_b64 in test_images_b64:
        try:
            _, b64_data = img_b64.split(",", 1)
            img_bytes = base64.b64decode(b64_data)
            np_arr = np.frombuffer(img_bytes, np.uint8)
            bgr_image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
            if bgr_image is None: continue
            gray_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray_image, 1.1, 5, minSize=(60, 60))
            if not len(faces): continue
            x, y, w, h = max(faces, key=lambda r: r[2] * r[3])
            face_crop = bgr_image[y:y+h, x:x+w]
            test_embeddings.append(get_embedding(face_crop))
        except Exception:
            continue

    if not test_embeddings:
        return JSONResponse({"success": True, "decision": "Nenhum rosto detectado nas tentativas."},
status_code=200)
```

```
    best_overall_dist = float('inf')
    best_overall_user_id = None
    for test_emb in test_embeddings:
        for ref_user_id, ref_img_name, ref_emb in db_reference_images:
            dist = np.linalg.norm(test_emb - ref_emb)
            if dist < best_overall_dist:
                best_overall_dist = dist
                best_overall_user_id = ref_user_id
    if best_overall_dist < THRESHOLD:
        decision = "Usuário reconhecido"
        exists = True
        user_id = best_overall_user_id
    else:
        decision = "Usuário não reconhecido."
        exists = False
        user_id = None

    return JSONResponse({
        "success": True, "exists": exists, "user_id": user_id,
        "distance": float(best_overall_dist), "decision": decision
    }, status_code=200)


@app.post("/register")
async def api_register(data: dict = Body(...)):
    image_list_b64 = data.get("images")
    if not image_list_b64 or not isinstance(image_list_b64, list) or len(image_list_b64) != 3:
        return JSONResponse({"success": False, "error": "Requisição deve conter exatamente 3 imagens."},
status_code=400)

    face_crops = []
    for img_b64 in image_list_b64:
        try:
            _, b64_data = img_b64.split(",", 1)
            img_bytes = base64.b64decode(b64_data)
            np_arr = np.frombuffer(img_bytes, np.uint8)
            bgr_image = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
            gray_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2GRAY)
            faces = face_cascade.detectMultiScale(gray_image, 1.1, 5, minSize=(60, 60))
            if not len(faces): continue
            x, y, w, h = max(faces, key=lambda r: r[2] * r[3])
            face_crops.append(bgr_image[y:y+h, x:x+w])
        except Exception:
            continue

    if len(face_crops) != 3:
        return JSONResponse({"success": False, "error": f"Rosto não detectado em todas as 3 imagens.
Apenas {len(face_crops)} rostos válidos encontrados."}, status_code=400)

    new_embeddings = [get_embedding(crop) for crop in face_crops]
    avg_new_emb = np.mean(new_embeddings, axis=0)
    best_existing_dist = float('inf')
    best_existing_id = None
    if db_reference_images:
        for ref_user_id, _, ref_emb in db_reference_images:
            dist = np.linalg.norm(avg_new_emb - ref_emb)
            if dist < best_existing_dist:
                best_existing_dist = dist
                best_existing_id = ref_user_id
```

```
if best_existing_id and best_existing_dist < THRESHOLD:
    return JSONResponse({
        "success": True, "exists": True, "user_id": best_existing_id,
        "decision": "Este usuário já parece estar cadastrado.", "distance": float(best_existing_dist)
    }, status_code=200)

user_id = uuid.uuid4().hex
user_dir = os.path.join(PEOPLE_DIR, user_id)
os.makedirs(user_dir)

for i, crop in enumerate(face_crops):
    cv2.imwrite(os.path.join(user_dir, f"{i}.jpg"), crop)

reload_db_from_storage()

return JSONResponse({
    "success": True, "exists": False, "user_id": user_id,
    "decision": f"Novo usuário cadastrado com sucesso!"
}, status_code=201)
```

Sistema de Reconhecimento Facial utilizando Esp32-Cam:

```
#include <WiFi.h>
#include <ArduinoHttpClient.h>
#include "esp_camera.h"

// --- DEFINIÇÕES DA CÂMERA ---
#define CAMERA_MODEL_AI_THINKER
#include "camera_pins.h"

// ===========================================================
// 1. CONFIGURAÇÕES DO USUÁRIO (MODIFIQUE AQUI)
// ===========================================================

const char* ssid = "NOME_DA_SUA_REDE_WIFI";
const char* password = "SENHA_DA_SUA_REDE_WIFI";

// <<< IMPORTANTE: Coloque o IP do seu computador onde o app.py está rodando >>>
const char* server_address = "192.168.1.10";
const int server_port = 8000;

const int FOTOS_NECESSARIAS = 3; // Quantidade de fotos para cadastro/reconhecimento
const int INTERVALO_FOTOS_MS = 1500; // Tempo em ms entre as fotos
const int INTERVALO_RECONHECIMENTO_S = 10; // Reconhecer automaticamente a cada 10
segundos

//============================================================

WiFiClient wifi;
HttpClient client = HttpClient(wifi, server_address, server_port);

void setup() {
  Serial.begin(115200);
  Serial.println("\n\n=== Sistema de Reconhecimento Facial Dual-Mode ===");

  // --- INICIALIZA A CÂMERA ---
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
```

```
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_VGA;
  config.jpeg_quality = 12;
  config.fb_count = 2; // Aumentar para 2 para mais estabilidade

  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Falha na inicialização da câmera: 0x%x\n", err);
    while (true);
  }
  Serial.println("Câmera inicializada.");

  // --- CONECTA AO WI-FI ---
  WiFi.begin(ssid, password);
  Serial.print("Conectando ao Wi-Fi ");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConectado!");
  Serial.print("Endereço IP da ESP32-CAM: ");
  Serial.println(WiFi.localIP());

  Serial.println("\n================================================");
  Serial.println("INSTRUÇÕES:");
  Serial.println("- O reconhecimento automático começará em breve.");
  Serial.println("- Para iniciar um novo CADASTRO, envie a letra 'c' pelo Monitor Serial.");
  Serial.println("================================================");
}

// Função auxiliar para capturar múltiplas imagens
String capture_multiple_images_as_base64_array() {
  String image_array_json = "[\n";
  for (int i = 0; i < FOTOS_NECESSARIAS; i++) {
    Serial.printf("Capturando foto %d de %d...\n", i + 1, FOTOS_NECESSARIAS);
    camera_fb_t* fb = esp_camera_fb_get();
    if (!fb) {
      Serial.println("Falha ao capturar imagem.");
      return ""; // Retorna string vazia em caso de falha
    }

    // Adiciona a imagem codificada em base64 ao array JSON
    image_array_json += "\"";
    image_array_json += base64::encode(fb->buf, fb->len);
    image_array_json += "\"";

    esp_camera_fb_return(fb);

    if (i < FOTOS_NECESSARIAS - 1) {
      image_array_json += ",\n"; // Adiciona vírgula entre os elementos
      delay(INTERVALO_FOTOS_MS);
    }
  }
  image_array_json += "\n]";
  return image_array_json;
}
```

```
// Função para enviar os dados para o servidor e imprimir a resposta
void send_request_and_print_response(String endpoint, String image_array_json) {
    String json_payload = "{\"images\": " + image_array_json + "}";

    Serial.println("Enviando para o servidor...");
    client.beginRequest();
    client.post(endpoint);
    client.sendHeader("Content-Type", "application/json");
    client.sendHeader("Content-Length", json_payload.length());
    client.write((const byte*)json_payload.c_str(), json_payload.length());
    client.endRequest();

    int statusCode = client.responseStatusCode();
    String response = client.responseBody();

    Serial.println("\n--- Resposta do Servidor ---");
    Serial.print("Status code: ");
    Serial.println(statusCode);
    Serial.print("Resposta: ");
    Serial.println(response);
    Serial.println("----------------------------");
}

// Função específica para realizar o CADASTRO
void realizar_cadastro() {
  Serial.println("\n### INICIANDO PROCESSO DE CADASTRO ###");
  String image_array = capture_multiple_images_as_base64_array();
  if (image_array.length() > 0) {
    send_request_and_print_response("/api/register", image_array);
  } else {
    Serial.println("Cadastro cancelado devido a falha na captura.");
  }

// Função específica para realizar o RECONHECIMENTO
void realizar_reconhecimento() {
  Serial.println("\n### INICIANDO PROCESSO DE RECONHECIMENTO ###");
  String image_array = capture_multiple_images_as_base64_array();
  if (image_array.length() > 0) {
    send_request_and_print_response("/api/recognize", image_array);
  } else {
    Serial.println("Reconhecimento cancelado devido a falha na captura.");
  }
}

void loop() {
  // Verifica se o usuário enviou um comando de cadastro
  if (Serial.available() > 0) {
    char command = Serial.read();
    if (command == 'c' || command == 'C') {
      realizar_cadastro();
    }
  }

  static unsigned long last_recognition_time = 0;
  if (millis() - last_recognition_time > (INTERVALO_RECONHECIMENTO_S * 1000)) {
```

```
  realizar_reconhecimento();
  last_recognition_time = millis();
}

static bool first_run = true;
if (first_run) {
    Serial.println("\nEnvie 'c' para CADASTRAR ou 'r' para RECONHECER.");
    first_run = false;}
if (Serial.available() > 0) {
  char command = Serial.read();
  if (command == 'c' || command == 'C') {
      realizar_cadastro();
  } else if (command == 'r' || command == 'R') {
      realizar_reconhecimento();
  }  }

delay(100); // Pequeno delay para estabilidade}
```

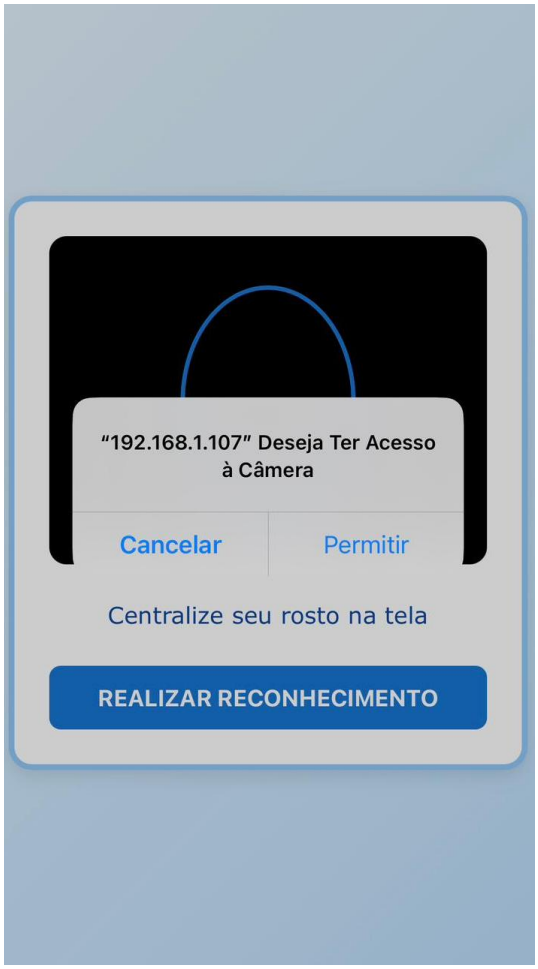
ANEXOS

Telas do sistema web de reconhecimento facial:

Figura 8. Tela Index.

Fonte: A autora, 2025.

Figura 9. Solicitação de acesso à câmera.



Fonte: A autora, 2025.

Figura 10. Tela Recognize.

Fonte: A autora, 2025.

Figura 11. Tela Not_registered.

Fonte: A autora, 2025.

Figura 12. Tela Register.

Fonte: A autora, 2025.

Figura 13. Tela Welcome.

Fonte: A autora, 2025.