UFABC

Universidade Federal do ABC

Trabalho de Graduação em Engenharia de Informação

Sistema de Reconhecimento Facial em Python para Ambientes Multiplataforma

Ana Paula Sales de Araujo

Santo André - SP
2025

Ana Paula Sales de Araujo

Sistema de Reconhecimento Facial em Python para Ambientes Multiplataforma

Trabalho de Graduação apresentado ao curso de Engenharia de Informação da Universidade Federal do ABC, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia de Informação.

Universidade Federal do ABC – UFABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Programa de Graduação em Engenharia de Informação

Orientador: Prof. Dr. Mario Minami

Santo André – SP
2025

Agradecimentos

A Deus, por me dar força, sabedoria e coragem para continuar, mesmo diante dos desafios mais difíceis.

Aos meus pais, que sempre se esforçaram ao máximo para me oferecer oportunidades que eles mesmos não tiveram. Vindos de uma realidade em que sequer puderam concluir o ensino médio, vocês me mostraram que o conhecimento transforma e que o amor e a luta diária constroem sonhos. Essa conquista é de vocês tanto quanto é minha.

À minha melhor amiga Carol, que foi abrigo nos momentos em que eu duvidei de mim mesma. Sua amizade me fortaleceu quando minhas forças falharam, e sua presença foi essencial para que eu seguisse.

Ao meu namorado Lucas, por acreditar em mim, pela paciência, incentivo e apoio constante em todas as fases dessa jornada. E à minha irmã Amanda, que sempre esteve ao meu lado com palavras de carinho, motivação e orgulho — vocês foram âncoras nos meus dias mais turbulentos.

Aos professores da graduação, que me ensinaram com dedicação e me ajudaram a enxergar o mundo com olhar crítico e construtivo. Em especial, agradeço ao Professor Dr. Mario Minami, pela orientação precisa, paciência e apoio ao longo deste trabalho. Sua contribuição foi essencial para que esta pesquisa se concretizasse e seguirá presente na minha trajetória como profissional e como pessoa.

A todos que, de alguma forma, se fizeram presentes durante a minha trajetória — amigos, colegas, familiares — muito obrigada. Cada gesto de apoio foi essencial para que eu chegasse até aqui.

Este trabalho representa muito mais que a conclusão de um curso — é uma vitória pessoal imensa, construída com esforço, disciplina e resiliência. Uma conquista que carrego com orgulho, por tudo que ela simboliza para mim e para minha história.

Dedico este trabalho aos meus pais, que sob muito sol me fizeram chegar até aqui pela sombra e com água fresca. E também à futura Ana, que irá colher os frutos de cada renúncia e esforço. Que ela saiba que tudo valeu a pena.

Resumo

O presente trabalho propõe o desenvolvimento de um protótipo de reconhecimento facial multiplataforma utilizando técnicas de visão computacional com a linguagem de programação *Python*. O protótipo foi projetado com foco em aplicações de controle de acesso em portarias de condomínios, visando maior agilidade, segurança e facilidade de implantação. Para a detecção das faces, utilizou-se o classificador *Haar Cascade*, enquanto o reconhecimento foi realizado por meio do algoritmo *Local Binary Patterns Histograms* (*LBPH*), conhecido por sua robustez e eficiência em diferentes condições de iluminação. As imagens faciais foram capturadas por *webcam*, processadas e organizadas conforme o identificador do usuário, possibilitando o treinamento de um experimento supervisionado. O protótipo final apresentou uma acurácia média de 87% na identificação correta dos indivíduos, considerando uma base com 75 imagens, podendo ser integrado a soluções reais de monitoramento e segurança. Além disso, o projeto pode ser expandido futuramente com algoritmos mais avançados de *deep learning* para aplicações em larga escala.

Palavras-chave: reconhecimento facial, visão computacional, segurança, *OpenCV*, *LBPH*, *Python*.

Abstract

This work proposes the development of a multiplatform facial recognition prototype using computer vision techniques implemented in Python. The system is designed for access control in environments such as condominium entrances, aiming to improve security, agility, and ease of deployment. Facial detection is performed using the Haar Cascade classifier, and recognition is handled by the Local Binary Patterns Histograms (LBPH) algorithm, known for its robustness and efficiency under different lighting conditions. Facial images are captured via webcam, processed, and organized by user ID, allowing the training of a supervised model. The final system achieved an average accuracy of 87%, in correctly identifying registered individuals, considering a dataset with 75 images, and can be integrated into real-world monitoring and security solutions. Furthermore, the project can be expanded in the future with more advanced deep learning algorithms for large-scale applications.

Keywords: facial recognition, computer vision, security, OpenCV, LBPH, Python.

Sumário

1. Introdução

O reconhecimento facial é uma das tecnologias mais consolidadas e estudadas dentro da área de visão computacional [1], destacando-se por sua capacidade de identificar indivíduos a partir de imagens ou vídeos que contenham seus rostos. Essa tecnologia biométrica é fundamentada no princípio de que cada rosto humano possui características únicas — como a distância entre os olhos, o formato do queixo, o comprimento do nariz, entre outras — que podem ser quantificadas e transformadas em padrões matemáticos. Seu uso tem se expandido de forma significativa em aplicações que vão desde desbloqueio de dispositivos móveis até sistemas de vigilância pública, passando por controle de acesso, autenticação digital e interação homem-máquina [2].

Dentro do campo da visão computacional, o reconhecimento facial representa uma das tarefas mais desafiadoras e, ao mesmo tempo, mais promissoras. Isso se deve ao fato de que ele envolve a combinação de múltiplas etapas do processamento de imagem: detecção de rosto, pré-processamento, extração de características e, finalmente, a classificação. A detecção é comumente realizada utilizando o classificador *Haar Cascade*, proposto por Viola e Jones [3], devido à sua eficiência em tempo real. Já na fase de reconhecimento, são utilizados algoritmos que extraem padrões distintos da face e os comparam com uma base de dados previamente treinada. Entre os métodos mais utilizados estão *Eigenfaces*, *Fisherfaces* e, especialmente, o *Local Binary Patterns Histograms (LBPH)*, que oferece robustez em cenários com variação de iluminação e expressões faciais [4].

A relevância do reconhecimento facial está diretamente relacionada ao seu potencial de aplicação em cenários do mundo real. Em portarias de condomínios, por exemplo, o uso dessa tecnologia pode substituir métodos tradicionais de controle de acesso, como chaves, cartões ou senhas, oferecendo uma alternativa mais segura, automatizada e conveniente. Além disso, a tecnologia pode ser implementada em sistemas de baixo custo, com câmeras comuns e software de código aberto, como o OpenCV — biblioteca amplamente utilizada para desenvolvimento de aplicações de visão computacional, desenvolvida em 1999 por Gary Bradski [5]. Essa acessibilidade tecnológica torna viável a adoção de sistemas inteligentes mesmo em contextos residenciais ou comerciais de pequeno e médio porte.

A pandemia de COVID-19 acelerou a necessidade de diversas indústrias adotarem as oportunidades proporcionadas pela tecnologia de reconhecimento facial. [6]. Experiências sem contato tornaram-se a norma, permitindo que empresas ofereçam a seus clientes e funcionários interações mais seguras em seus serviços. Desde o embarque em aviões até a compra de um hambúrguer, espera-se que o reconhecimento facial seja cada vez mais integrado ao cotidiano das pessoas.

Muitos indivíduos já estão habituados a interações diárias que utilizam reconhecimento facial. Atualmente, grande parte dos smartphones disponíveis no mercado já é equipada com essa tecnologia, permitindo que sejam desbloqueados de forma contínua e sem necessidade de contato físico.

Diante desse cenário, este trabalho justifica-se por propor uma solução local, de baixo custo, multiplataforma e de fácil implementação, utilizando apenas ferramentas de código aberto como *Python* e *OpenCV*. A proposta foi idealizada para funcionar sem dependência de internet, o que minimiza custos operacionais e garante maior autonomia e confiabilidade

em ambientes residenciais. Além disso, ao adotar uma arquitetura modular e flexível, o sistema pode ser facilmente adaptado para diferentes contextos e necessidades. Em cenários mais complexos e com maior demanda por escalabilidade, é possível virtualizar e hospedar o ambiente em nuvens públicas, possibilitando acesso remoto, integração com bancos de dados e maior poder computacional.

Este trabalho propõe, portanto, o desenvolvimento de um sistema experimental de reconhecimento facial multiplataforma com foco no controle de acesso em condomínios. A solução utiliza o algoritmo LBPH para reconhecimento e o classificador *Haar Cascade* para detecção de rostos. O modelo treinado atingiu uma acurácia de aproximadamente 87% na identificação de usuários previamente cadastrados, demonstrando que é possível aliar eficiência, acessibilidade e aplicabilidade prática por meio de uma arquitetura simples e escalável.

2. Objetivos

2.1 Objetivo Geral

Desenvolver um sistema de reconhecimento facial multiplataforma utilizando técnicas de visão computacional com a linguagem Python e a biblioteca OpenCV, com foco na aplicação em controle de acesso residencial, oferecendo uma solução local, de baixo custo, eficiente e passível de expansão para ambientes em nuvem.

2.2 Objetivos Específicos

- Implementar um módulo de captura de imagens faciais via webcam, organizando os dados de forma estruturada para treinamento supervisionado;
- Aplicar o algoritmo Haar Cascade para a detecção de rostos em tempo real;
- Utilizar o algoritmo Local Binary Patterns Histograms (LBPH) para o reconhecimento facial, avaliando sua acurácia em diferentes condições de iluminação;
- Desenvolver uma interface simples e funcional para cadastro e identificação de usuários;
- Garantir que o sistema opere localmente, sem dependência de conexão com a internet, visando minimizar os custos de implantação e aumentar a confiabilidade;
- Avaliar a viabilidade da virtualização do sistema em uma infraestrutura de nuvem pública, como a Oracle Cloud, considerando cenários de maior escala.

3. Fundamentação Teórica

Como mencionado na Introdução e nos Objetivos, a intenção deste trabalho é realizar o reconhecimento facial de indivíduos a partir de características únicas extraídas de suas imagens faciais, utilizando técnicas de visão computacional com o apoio da biblioteca *OpenCV*. Para isso, é essencial compreender como o *OpenCV* atua nesse contexto, oferecendo recursos robustos para a detecção e o reconhecimento de rostos em imagens ou vídeos em tempo real.

A biblioteca fornece implementações eficientes de algoritmos clássicos como o Haar Cascade para detecção facial, além de ferramentas para pré-processamento de imagens e extração de características. No caso deste trabalho, o reconhecimento será feito utilizando o algoritmo *Local Binary Patterns Histograms (LBPH)*, que transforma a imagem do rosto em um vetor de características com base em padrões de textura locais, permitindo a comparação com rostos previamente cadastrados.

A seguir, serão abordados os conceitos principais envolvidos no reconhecimento facial, bem como a forma como a biblioteca *OpenCV* os implementa na prática.

3.1 Reconhecimento Facial: História e Fundamentação

O processo de reconhecimento facial consiste na identificação ou verificação da identidade de uma pessoa por meio da análise e comparação das características únicas de seu rosto. Este processo envolve a captura de uma imagem facial, a detecção da face dentro da imagem, extração das características faciais relevantes, e comparação dessas características com um banco de dados pré-existente para determinar a identidade ou autenticar o indivíduo [9].

Nos primórdios do reconhecimento facial, as primeiras tentativas surgiram na década de 1960, impulsionadas principalmente pela necessidade de automatizar processos de identificação pessoal. As técnicas iniciais eram limitadas pela tecnologia da época, que sofria com baixa capacidade computacional, dificuldades de processamento de imagens e ausência de algoritmos sofisticados. Além disso, a necessidade de controlar as condições ambientais, como iluminação e posicionamento da face, tornava o processo extremamente rígido e pouco prático [10].

Durante as décadas seguintes, avanços tecnológicos permitiram melhorias significativas na área, como o surgimento de algoritmos mais robustos e precisos. Na década de 1990, por exemplo, surgiram técnicas como Eigenfaces, que usavam a Análise de Componentes Principais (PCA) para reduzir a dimensionalidade das imagens faciais, permitindo um reconhecimento mais rápido e eficaz [4]. Entretanto, ainda persistiam dificuldades relacionadas à variação nas expressões faciais, ângulo de captura, iluminação e obstruções parciais, o que mantinha o reconhecimento facial limitado a contextos controlados.

Atualmente, o reconhecimento facial é marcado por avanços expressivos devido ao desenvolvimento de métodos baseados em aprendizado profundo (*Deep Learning*) e redes neurais convolucionais (CNNs). Essas técnicas modernas permitem maior robustez frente a variações naturais em condições de captura, incluindo mudanças de expressão, ângulos diferentes, iluminação variável e obstruções parciais da face [11]. Sistemas modernos conseguem realizar reconhecimentos rápidos e precisos mesmo em contextos desafiadores, ampliando significativamente suas aplicações práticas em áreas como segurança pública, autenticação biométrica,

vigilância, *marketing* personalizado e interação humano-computador.

Neste trabalho, foi utilizada a biblioteca *OpenCV*, combinada com o *classificador Haar Cascad*e, para implementar o sistema de reconhecimento facial. A linguagem utilizada foi *Python*. Essa combinação permite realizar a detecção e identificação facial de forma eficaz e acessível, utilizando métodos já consolidados na literatura científica.

Optou-se por essa abordagem tradicional em detrimento de técnicas baseadas em aprendizado profundo e CNNs devido ao custo computacional elevado dessas últimas. Modelos de *Deep Learning*, especialmente redes neurais convolucionais, requerem uma grande quantidade de dados para treinamento, além de *hardware* especializado — como GPUs de alto desempenho — para alcançar resultados satisfatórios em tempo real [15]. Esses modelos geralmente apresentam alta complexidade computacional, demandando processamento paralelo, consumo elevado de memória e maior tempo de execução, o que os torna inviáveis para aplicações em dispositivos com recursos limitados.

O presente trabalho foi pensado com foco em aplicações embarcadas, priorizando soluções de baixo custo e baixo consumo de *hardware*, o que torna a abordagem com H*aar Cascade e LBPH* mais adequada nesse contexto. Esses métodos são mais leves, não exigem aceleração por *hardware* gráfico e podem ser executados com eficiência em processadores embarcados, como *Raspberry Pi*, facilitando sua integração em sistemas autônomos ou portáteis.

3.2 Python

Python é atualmente uma das linguagens de programação mais utilizadas no mundo [16], sendo amplamente adotada em aplicações de ciência de dados, inteligência artificial e visão computacional. Sua sintaxe simples e legível facilita o desenvolvimento e a manutenção de sistemas complexos, tornando-se uma escolha ideal tanto para iniciantes quanto para desenvolvedores experientes [17]. Além disso, *Python* conta com um ecossistema robusto de bibliotecas específicas para processamento de imagens e reconhecimento facial, como *OpenCV, NumPy, Dlib e face_recognition*. Essa vasta gama de ferramentas permite a implementação rápida e eficiente de algoritmos avançados de detecção e identificação facial. No contexto deste trabalho, *Python* foi utilizado como linguagem principal devido à sua flexibilidade, suporte a bibliotecas especializadas e pela comunidade ativa, que oferece extensa documentação e suporte contínuo.

3.3 Bibliotecas *Open CV*

OpenCV, ou *Open Source Computer Vision Library*, é uma biblioteca de código aberto voltada para a visão computacional e processamento de imagens. Desenvolvida inicialmente pela Intel em 1999 e lançada como *open source* em 2000, a *OpenCV* tem como objetivo principal fornecer uma infraestrutura eficiente para aplicações que envolvem o entendimento visual por parte de computadores [12].

A biblioteca é multiplataforma, compatível com sistemas operacionais como Windows, Linux, macOS, Android e iOS. Além disso, oferece suporte a várias linguagens de programação, incluindo C++, *Python* e *Java*, permitindo que desenvolvedores integrem facilmente suas funcionalidades em diferentes ambientes [12].

Com um extenso conjunto de algoritmos, a *OpenCV* abrange áreas como detecção e reconhecimento facial, rastreamento de objetos, análise de movimento, segmentação de imagens e realidade aumentada. Sua eficiência e desempenho otimizado a tornam adequada para aplicações em tempo real, sendo amplamente utilizada em sistemas de segurança, veículos autônomos, robótica e outras áreas que exigem processamento rápido e preciso de imagens.

Neste trabalho, foi utilizada a linguagem *Python* em conjunto com a biblioteca *OpenCV* para implementar o sistema de reconhecimento facial. Para a detecção facial, utilizou-se o classificador *Haar Cascade* fornecido pela *OpenCV*. Para o treinamento e reconhecimento das diferentes faces, foi aplicado o algoritmo *Local Binary Pattern Histogram (LBPH)*, conhecido por sua eficiência e simplicidade na representação de texturas faciais.

3.4 Classificador Haar Cascade

O classificador *Haar Cascade*, implementado na biblioteca *OpenCV*, é uma técnica baseada em aprendizado de máquina utilizada para detecção de objetos em imagens, sendo amplamente empregada para identificar faces humanas. Essa abordagem foi originalmente proposta por Viola e Jones [3] no artigo *"Rapid Object Detection using a Boosted Cascade of Simple Features"* e se destaca pela sua eficiência e desempenho em tempo real.

A técnica é baseada em características *Haar-like*, que são padrões compostos por áreas claras e escuras que representam bordas, linhas e texturas comuns em rostos humanos, como olhos, nariz e boca. Essas características são extraídas de janelas da imagem de entrada e avaliadas para verificar a presença de uma face.

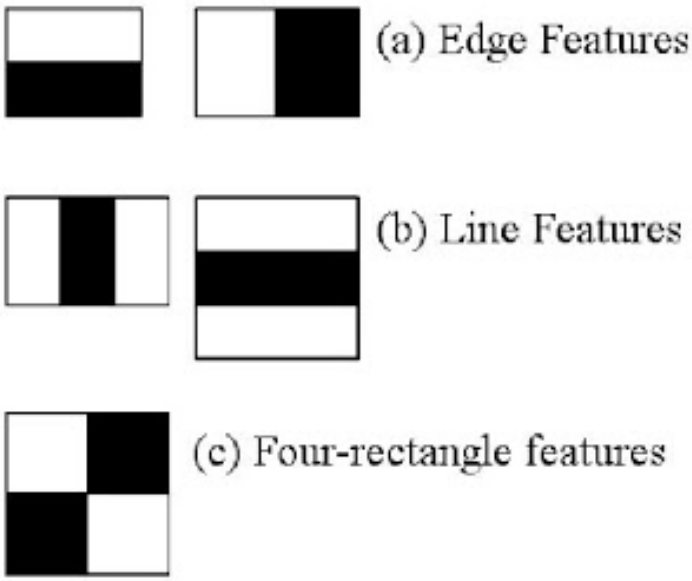

Figura 1: Exemplo de característica de Haar utilizada na detecção facial. Fonte: Documentação do Haar Cascade (OpenCV, 2025) [18].

Inicialmente, o algoritmo necessita de muitas imagens positivas (imagens contendo rostos) e imagens negativas (imagens sem rostos) para treinar o classificador. Após isso, é necessário

extrair características dessas imagens. Para isso, são utilizadas as chamadas características de *Haar*, ilustradas na Figura 1. Elas se assemelham a núcleos de convolução utilizados em redes neurais. Cada característica é representada por um único valor, obtido pela subtração da soma dos pixels sob o retângulo branco pela soma dos pixels sob o retângulo preto [18].

Para a detecção facial utilizando o classificador *Haar Cascade*, uma grande quantidade de características (*features*) é extraída a partir da aplicação de diferentes tamanhos e posições de janelas retangulares sobre a imagem. Mesmo uma pequena janela de 24×24 pixels pode gerar mais de 160.000 características distintas, o que implica em um alto custo computacional para o processamento de imagens.

Cada uma dessas características é obtida pela diferença entre a soma dos pixels situados sob as regiões claras e escuras de um retângulo, o que inicialmente demandaria muitos cálculos. Para tornar esse processo mais eficiente, foi introduzido o conceito de imagem integral (integral image), que permite calcular a soma de qualquer região retangular da imagem utilizando apenas quatro acessos à matriz de pixels, independentemente de seu tamanho. Essa abordagem otimiza significativamente a velocidade de processamento, tornando viável a detecção em tempo real [3].

Entretanto, nem todas as características extraídas são relevantes para a detecção de rostos. Muitas delas não contribuem de forma significativa para a diferenciação entre regiões faciais e não faciais. Por exemplo, certas características destacam o contraste entre a região dos olhos, normalmente mais escura, e as bochechas ou a ponte do nariz, que tendem a ser mais claras — essas são úteis. Já outras, aplicadas a regiões sem padrão definido, são descartadas por não agregarem valor ao processo de identificação.

A seleção das características mais relevantes entre as milhares disponíveis é realizada por meio do algoritmo *AdaBoost*, que escolhe e combina os classificadores fracos mais eficazes para formar um classificador forte. Essa seleção é fundamental para reduzir a complexidade e aumentar a precisão do sistema de detecção.

Para cada característica, é definido um limiar que melhor separa as imagens em suas respectivas classes. O algoritmo *AdaBoost* é utilizado para selecionar as características mais relevantes, atribuindo pesos maiores às imagens mal classificadas a cada iteração, até que se atinja a precisão desejada.

O classificador final é formado por uma combinação ponderada de classificadores fracos, que sozinhos não são eficazes, mas juntos formam um classificador robusto. Embora existam mais de 160.000 características possíveis, o método de Viola e Jones [3] demonstra que cerca de 6.000 são suficientes para atingir uma taxa de acerto superior a 95%.

Para otimizar o desempenho, os autores propuseram o uso de uma Cascata de Classificadores, na qual as características são organizadas em múltiplos estágios. Nos primeiros estágios, apenas algumas características são utilizadas para rapidamente eliminar regiões da imagem que claramente não contêm rostos. Somente as regiões que passam por todos os estágios são classificadas como contendo um rosto. Esse modelo reduz drasticamente o número de cálculos necessários, avaliando, em média, apenas 10 características por subjanela.

3.5 Algoritmo Adaboost

O algoritmo *AdaBoost (Adaptive Boosting)*, desenvolvido por Freund e Schapire (1995), é um método de aprendizado de máquina do tipo ensemble que combina múltiplos classi-

ficadores fracos para formar um classificador forte [7]. Inicialmente, todas as amostras do conjunto de treinamento recebem pesos iguais. A cada iteração, os pesos das amostras mal classificadas são aumentados, e o processo é repetido até alcançar a precisão desejada. No contexto da detecção facial com *Haar Cascade, o AdaBoost* é responsável por selecionar as características mais relevantes e combiná-las de forma eficiente, tornando o sistema mais rápido e preciso ao eliminar características irrelevantes [19].

3.6 Local Binary Pattern Histogram (LBPH)

O *Local Binary Pattern Histogram (LBPH)* constitui um dos algoritmos mais difundidos no campo do reconhecimento facial, notadamente por sua simplicidade conceitual, baixo custo computacional e desempenho satisfatório em ambientes controlados. Fundamenta-se na técnica de *Local Binary Patterns (LBP)*, originalmente proposta para análise de texturas, que foi posteriormente adaptada para aplicações em biometria facial, demonstrando resultados promissores [8].

A principal operação do *LBPH* consiste na análise das relações locais de intensidade luminosa entre pixels adjacentes em imagens em escala de cinza. Para cada pixel da imagem, é considerada uma vizinhança — comumente uma matriz 3×3 — e os valores dos pixels vizinhos são comparados ao valor do pixel central. Se o valor do vizinho for maior ou igual ao do centro, atribui-se o dígito 1; caso contrário, 0 [8]. Essa comparação gera uma sequência binária que, ao ser convertida para um valor decimal, passa a representar a textura local da região analisada.

A seguir, a imagem codificada é dividida em diversas regiões ou células. Para cada célula, calcula-se um histograma de frequência dos padrões *LBP* encontrados. Esses histogramas refletem variações estruturais presentes em partes específicas do rosto, como olhos, nariz e boca. A concatenação de todos os histogramas locais resulta em um vetor unificado de características, que sintetiza a assinatura da face.

Na etapa de reconhecimento, o vetor extraído da imagem consultada é comparado com os vetores previamente armazenados na base de dados. Essa comparação é conduzida por meio de métricas de distância, como as distâncias Euclidiana, *Manhattan* ou *Chi-Square*, sendo atribuída a identidade correspondente ao vetor mais próximo [13]. Para este trabalho, utilizou-se a distância padrão adotada pelo *OpenCV*: *Chi-Square,* cuja fórmula pode ser observada na equação 1:

$$\chi^2(H_1, H_2) = \sum_{i=1}^{n} \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i) + \epsilon} \tag{1}$$

Na equação da distância de *Chi-Square*, os símbolos possuem os seguintes significados:

- $H_1(i)$ representa o valor do bin i do histograma da imagem de entrada;
- $H_2(i)$ representa o valor do bin i do histograma de uma imagem armazenada no banco de dados;
- ϵ é um número muito pequeno (por exemplo, 1×10^{-10}) utilizado para evitar divisão por zero;

- n é o número total de bins (categorias) existentes nos histogramas.

O *LBPH* apresenta diversas vantagens teóricas, destacando-se sua capacidade de operar eficientemente em tempo real, sua tolerância a variações de iluminação e a viabilidade de aplicação em contextos com bases de dados reduzidas. Todavia, a técnica mostra-se sensível a variações de escala, rotações do rosto e oclusões parciais. Apesar dessas limitações, permanece como uma alternativa eficaz e acessível para sistemas de reconhecimento facial em tempo real e com recursos computacionais limitados.

O parâmetro denominado *threshold* (limiar) no algoritmo *LBPH* define o valor máximo aceitável de distância entre o vetor de características da imagem consultada e os vetores armazenados no banco de dados para que o reconhecimento seja considerado válido [14]. Em outras palavras, quanto menor a distância entre os vetores, maior a similaridade entre as imagens comparadas. Caso a distância ultrapasse o valor estabelecido pelo *threshold*, o sistema considera que a face apresentada não pertence a nenhum dos indivíduos previamente cadastrados.

Durante o desenvolvimento deste trabalho, foram realizados testes com diferentes valores de *threshold*, com o objetivo de observar seu impacto no desempenho do protótipo. Foi constatado que, ao reduzir o valor do *threshold*, o sistema experimental tornou-se mais rigoroso, diminuindo a quantidade de falsos positivos, mas aumentando o número de falsos negativos, ou seja, deixando de reconhecer rostos válidos. Por outro lado, ao aumentar o valor do *threshold*, observou-se maior tolerância na identificação, o que resultou em mais reconhecimentos positivos, porém com maior risco de erros e reconhecimento indevido de rostos não cadastrados. Esses experimentos demonstram a importância da calibração cuidadosa desse parâmetro para atingir um equilíbrio entre precisão e sensibilidade do sistema experimental.

A análise dos resultados obtidos foi realizada com o auxílio da matriz de confusão, uma ferramenta fundamental para avaliar o desempenho de classificadores binários. Essa matriz organiza as previsões do sistema em quatro categorias: Verdadeiros Positivos (VP), Falsos Positivos (FP), Verdadeiros Negativos (VN) e Falsos Negativos (FN), permitindo uma visualização clara dos acertos e erros cometidos durante a classificação.

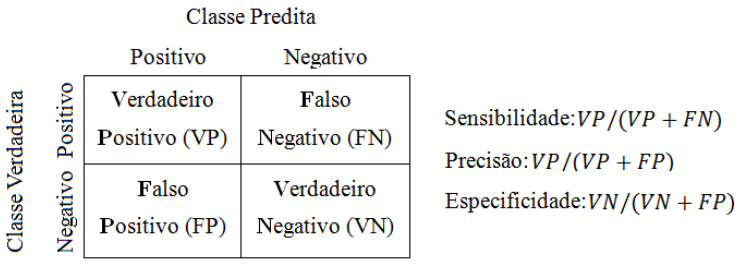

Figura 2: Matriz de confusão utilizada para a análise do desempenho do sistema de reconhecimento facial. Fonte: Filho et al. (2015) [20].

3.7 Métricas de Avaliação de Desempenho

Para a análise do desempenho do sistema de reconhecimento facial desenvolvido, foram adotadas quatro métricas amplamente reconhecidas na literatura: **acurácia**, **precisão**, **re-**

vocação (recall) e **F1-score**. Essas métricas permitem avaliar a capacidade do sistema em identificar corretamente as faces, bem como a taxa de erros cometidos durante a identificação.

- **Acurácia:** representa a proporção de classificações corretas (tanto positivas quanto negativas) em relação ao total de tentativas. A fórmula correspondente é apresentada na Equação 2:

$$\text{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN} \tag{2}$$

  Onde:

  - VP = Verdadeiros Positivos (faces corretamente reconhecidas)
  - VN = Verdadeiros Negativos (não-faces corretamente rejeitadas)
  - FP = Falsos Positivos (não-faces incorretamente reconhecidas como faces)
  - FN = Falsos Negativos (faces reais não reconhecidas)

- **Precisão:** avalia a proporção de reconhecimentos corretos entre todas as tentativas de reconhecimento. A expressão matemática pode ser visualizada na Equação 3

$$\text{Precisão} = \frac{VP}{VP + FP} \tag{3}$$

- **Revocação (Recall/Especificidade):** também conhecida como sensibilidade, representa a proporção de casos positivos corretamente identificados pelo modelo em relação ao total de casos positivos reais.

$$\text{Recall} = \frac{VP}{VP + FN} \tag{4}$$

- **F1-score:** é a média harmônica entre a precisão e o recall, sendo especialmente útil quando há um desequilíbrio entre as classes. Essa métrica busca um equilíbrio entre ambas as medidas.

$$\text{F1-score} = 2 \cdot \frac{\text{Precisão} \cdot \text{Recall}}{\text{Precisão} + \text{Recall}} \tag{5}$$

Essas métricas oferecem uma base sólida para interpretar o desempenho do sistema de forma quantitativa, além de identificar possíveis oportunidades de melhoria em cenários reais de aplicação.

4. Implementação Prática

A implementação do sistema foi organizada em etapas sequenciais, a fim de estruturar de forma clara o fluxo de desenvolvimento. Inicialmente, realizou-se a preparação do ambiente, seguida da criação do banco de imagens por meio da captura das imagens faciais. Em seguida, o modelo foi treinado com base nos dados armazenados, possibilitando a execução da etapa de reconhecimento facial. Por fim, os resultados obtidos foram analisados com o objetivo de avaliar o desempenho do sistema. É importante destacar que a adição de novos rostos ao banco de imagens exige um novo processo de ajuste do modelo, garantindo que ele seja capaz de reconhecer corretamente os novos indivíduos inseridos. O fluxograma pode ser observado pela Figura 3:

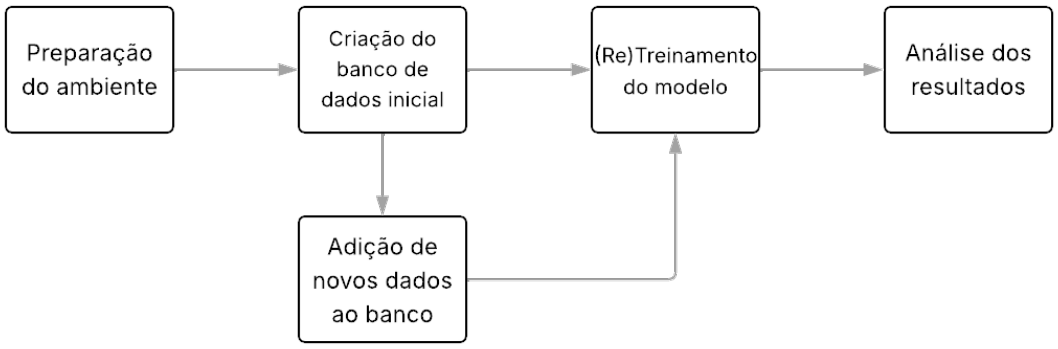

Figura 3: Fluxograma representando as etapas de implementação do sistema de reconhecimento facial. Fonte: Autor.

4.1 Preparação do Ambiente

Conforme abordado anteriormente, a primeira etapa da implementação consistiu na preparação do ambiente de desenvolvimento. Para isso, foi necessário configurar uma estrutura que atendesse a todos os pré-requisitos técnicos exigidos pelo sistema de reconhecimento facial. A linguagem utilizada foi o **Python**, na versão 3.6, juntamente com as bibliotecas **OpenCV** (para captura de imagens, detecção e reconhecimento facial), **NumPy** (para operações matriciais e manipulação de arrays) e **Pandas** (para organização e leitura de dados estruturados).

Para o gerenciamento das credenciais dos usuários (ID e nome), optou-se por uma solução simples e acessível: o uso de um arquivo no formato **CSV**, editável por ferramentas como o Microsoft Excel. Assim, é recomendável que o ambiente de execução possua suporte para esse tipo de arquivo, preferencialmente com o Excel instalado para facilitar edições manuais, caso necessário.

Além disso, foram incorporados dois arquivos essenciais ao funcionamento do sistema: o classificador *Haar Cascade*, `haarcascade_frontalface_default.xml`, utilizado na detecção de rostos, e o arquivo de modelo treinado, `TrainedLBPH.yml`, gerado após o processo de ajuste

com o algoritmo **LBPH (Local Binary Patterns Histograms)**. Ambos os arquivos estão disponíveis para consulta no Apêndice deste trabalho.

O desenvolvimento do sistema foi realizado em um computador com sistema operacional Microsoft Windows 11 Pro, versão 10.0.22631, fornecido pela Microsoft Corporation. A máquina utilizada foi um Dell Latitude 7430, com arquitetura x64 e processador Intel Core da família 6, modelo 154, com frequência aproximada de 2.2 GHz. O ambiente apresentou desempenho adequado para a execução das bibliotecas e frameworks empregados, sendo plenamente capaz de suportar as etapas de detecção, treinamento e reconhecimento facial durante o desenvolvimento da aplicação. Ressalta-se, ainda, que embora o sistema tenha sido implementado em ambiente Windows, sua execução é viável em outros sistemas operacionais, como Linux ou macOS, desde que sejam realizadas as devidas adaptações nas dependências, nos caminhos de arquivos e na compatibilidade das bibliotecas utilizadas. Além disso, o sistema também pode ser executado em máquinas virtuais ou adaptado para ambientes embarcados, desde que respeitados os requisitos mínimos de hardware e software, tornando-o uma solução flexível e multiplataforma. A verificação do ambiente em um sistema operacional windows pode ser feita conforme a Figura 4.

```
python --version
python -m pip install --upgrade pip
pip install opencv-python numpy pandas
```

Figura 4: Comandos para verificação da versão do Python, atualização do *pip* e instalação das bibliotecas necessárias. Fonte: Autor.

Com todos os pré-requisitos devidamente configurados, a etapa seguinte concentrou-se na implementação do módulo de captura de imagens faciais. Essa funcionalidade é responsável por acessar a *webcam* do dispositivo, detectar rostos em tempo real e armazenar as imagens capturadas de forma estruturada, compondo a base de dados que será utilizada nas fases posteriores de treinamento e reconhecimento.

4.2 Captura e Processamento das Imagens Faciais

Para o desenvolvimento dos módulos de captura, treinamento e reconhecimento facial, o código-fonte utilizado como base foi obtido a partir do projeto disponível em Leandro de Lima Carvalho (2021), no repositório: *LBPH Face Recognition* [21]. A partir dessa implementação inicial, o código foi adaptado com as alterações julgadas necessárias, de modo a atender aos objetivos específicos deste trabalho e possibilitar a obtenção dos resultados desejados.

A primeira etapa do código consiste na preparação do ambiente de execução, garantindo a existência dos arquivos e diretórios necessários para o correto funcionamento do sistema. Inicialmente, o *script* verifica a presença do arquivo id-names.csv, que atua como uma base de dados simplificada para armazenar os pares de identificação numérica (ID) e os respectivos nomes dos usuários. Caso esse arquivo já exista no diretório de trabalho, seus dados são carregados utilizando a biblioteca *Pandas*. Caso contrário, é criado um novo *DataFrame* com as colunas "Id"e "Name", e o arquivo é salvo automaticamente no formato .csv.

Na sequência, o sistema verifica se o diretório denominado faces/ já existe. Esse diretório é utilizado para armazenar as imagens faciais capturadas de cada usuário. Se o diretório ainda não estiver presente no ambiente, ele é criado automaticamente. Cada usuário terá uma subpasta dentro desse diretório, nomeada de acordo com seu ID, onde serão armazenadas as imagens capturadas durante o processo de registro facial. Esse procedimento garante a organização dos dados e facilita o acesso durante as etapas de treinamento e reconhecimento facial. O código pode ser observado no bloco de Código 1.

```
if os.path.exists('id-names.csv'):
    id_names = pd.read_csv('id-names.csv')
    id_names = id_names[['id', 'name']]
else:
    id_names = pd.DataFrame(columns=['id', 'name'])
    id_names.to_csv('id-names.csv', index=False)

if not os.path.exists('faces'):
    os.makedirs('faces')
```

Código 1: Inicialização do ambiente de captura.

Após a verificação do passo anterior, o sistema entra em um laço de repetição com o objetivo de validar o ID informado pelo usuário e garantir a consistência das informações registradas. Esse processo é fundamental para evitar conflitos e assegurar que cada usuário possua um identificador único vinculado ao seu nome.

O laço inicia com a exibição de uma mensagem de boas-vindas, solicitando que o usuário informe seu ID. Caso seja a primeira vez que utiliza o sistema, o próprio texto sugere a escolha de um número entre 1 e 300, que funcionará como identificador exclusivo.

Em seguida, o código verifica se o ID informado já está presente no arquivo id-names.csv, que armazena os pares de identificação e nome de cada usuário. Se o ID for encontrado, o sistema recupera o nome associado e solicita ao usuário que confirme sua identidade, respondendo com "S"(sim) ou "N"(não). Essa verificação adicional é importante para evitar que um usuário utilize, intencionalmente ou não, o ID de outra pessoa.

Caso o usuário confirme que é de fato o titular daquele ID, o nome é atribuído à variável name, e o laço é encerrado. Por outro lado, se a confirmação for negativa, o sistema solicita que um novo ID seja informado, retornando ao início do laço de repetição. A implementação pode ser observada no bloco de Código 2.

```
while True:
    print('Bem vindo(a)!')
    print('Por favor, insira o seu ID.')
    print('Se for a primeira vez, escolha um numero entre 1 e 300.')
    id = int(input('ID: '))
    name = ''

    # Caso o ID esteja no CSV (usuario ja registrado)
    if id in id_names['id'].values:
        existing_name = id_names[id_names['id'] == id]['name'].item
            ()
```

```
        print(f"O ID {id} ja esta associado ao nome: {existing_name
            }.")
        confirm = input(f"Voce confirma que e {existing_name}? (s/n)
            : ").strip().lower()

        if confirm == 's':
            name = existing_name
            print(f'Bem-vindo(a) de volta, {name}!')
            break
        else:
            print("Por favor, insira um novo ID.\n")
```

Código 2: Loop para garantir ID válido e confirmação de identidade.

Após a identificação e registro do usuário, o sistema inicia a etapa de captura das imagens faciais por meio da *webcam*. Esse processo é conduzido por um laço de repetição que se mantém ativo enquanto a câmera estiver em uso e até que o usuário finalize a sessão manualmente.

A inicialização ocorre com a chamada `cv.VideoCapture(0)`, que ativa a câmera padrão do dispositivo. Simultaneamente, é carregado o classificador Haar Cascade (`haarface.xml`) por meio da função `cv.CascadeClassifier()`, responsável pela detecção de rostos na imagem. A variável `photos_taken` é utilizada como contador para registrar a quantidade de fotos capturadas com sucesso.

Dentro do laço `while True`, cada quadro capturado pela câmera é obtido por meio da função `camera.read()`. Em seguida, a imagem é convertida para escala de cinza utilizando `cv.cvtColor()`, uma vez que o classificador Haar Cascade opera apenas sobre imagens monocromáticas.

Após essa conversão, a função `detectMultiScale()` é utilizada para localizar os rostos presentes na imagem. Ela retorna as coordenadas de cada face detectada, as quais são utilizadas para desenhar retângulos ao redor das regiões correspondentes.

Para cada rosto identificado, é extraída a região de interesse (ROI), e é calculada a média de brilho com `np.average()`, a fim de evitar o armazenamento de imagens muito escuras. Caso o usuário pressione a tecla `'s'` e o brilho da imagem seja superior a um limiar mínimo (50), a imagem da face é redimensionada para 220x220 pixels e salva no formato `.jpeg`, dentro da pasta correspondente ao ID do usuário. O nome do arquivo é gerado com base no ID e no carimbo de tempo em microssegundos, garantindo unicidade. Parte da lógica utilizada pode ser observada no bloco de Código 3.

```
camera = cv.VideoCapture(0)
face_classifier = cv.CascadeClassifier('haarface.xml')

while True:
    ret, img = camera.read()
    grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(grey)

    for (x, y, w, h) in faces:
        face = grey[y:y+h, x:x+w]
```

```
        if key == ord('s') and np.average(face) > 50:
            face_img = cv.resize(face, (220, 220))
            cv.imwrite(f'faces/{id}/face.{id}.{timestamp}.jpeg',
                face_img)

    if key == ord('q'):
        break
```

Código 3: Captura e salvamento das imagens faciais.

Esse mecanismo garante a captura de um conjunto representativo de imagens para cada usuário, com diferentes expressões, ângulos e condições de iluminação, contribuindo para a robustez do modelo de reconhecimento facial. O funcionamento descrito pode ser visualizado na Figura 5.

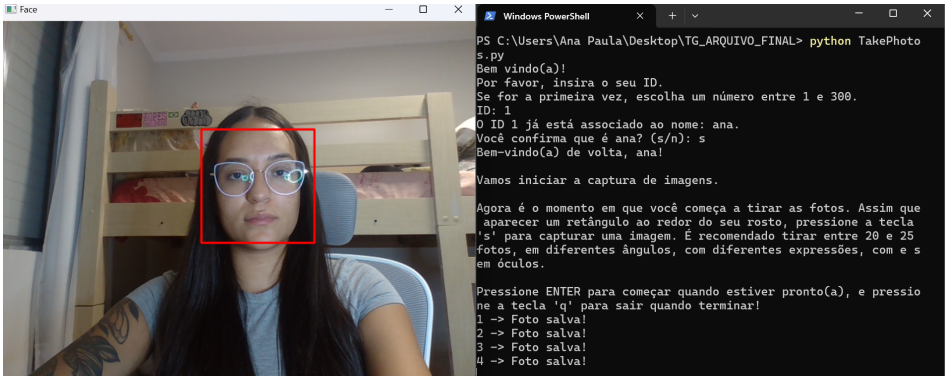

Figura 5: Interface de captura de imagens faciais em tempo real, com destaque para o retângulo de detecção. Fonte: Autor.

O resultado da captura pode ser observado na Figura 6, onde são exibidas as imagens contendo apenas a região facial, já convertidas para escala de cinza.

Figura 6: Resultado da captura: imagens faciais recortadas e convertidas para escala de cinza. Fonte: Autor.

4.3 Fase de Experimentação

A terceira etapa do sistema consiste no ajuste do modelo de reconhecimento facial com base nas imagens previamente capturadas. Essa fase é essencial para que o algoritmo seja capaz de aprender a associar as características faciais de cada usuário ao seu respectivo identificador (ID). Para isso, é utilizado o algoritmo *Local Binary Patterns Histograms* (LBPH), fornecido pela biblioteca *OpenCV*.

Durante o processo, o sistema percorre as pastas contendo as imagens faciais armazenadas na etapa anterior. Embora essas imagens tenham sido originalmente salvas em escala de cinza, o código realiza novamente a conversão para esse formato utilizando a função `cv.cvtColor()`, logo após a leitura com `cv.imread()`. Essa prática é adotada como medida preventiva, pois o OpenCV, por padrão, carrega imagens no formato BGR (três canais), mesmo que estas tenham sido armazenadas como monocromáticas [22].

Cada imagem convertida é então associada ao seu respectivo ID com base na estrutura de diretórios. Ao final do processo de modelagem, o modelo é salvo em um arquivo no formato `.yml`, o qual será utilizado posteriormente na etapa de reconhecimento facial em tempo real. Essa etapa do processo é exemplificada no bloco de Código 4.

```
id_names = pd.read_csv('id-names.csv')
id_names = id_names[['id', 'name']]

lbph = cv.face.LBPHFaceRecognizer_create(threshold=500)

def create_train():
    faces = []
    labels = []
    for id in os.listdir('faces'):
        path = os.path.join('faces', id)
        try:
            os.listdir(path)
        except:
            continue
        for img in os.listdir(path):
            try:
                face = cv.imread(os.path.join(path, img))
                face = cv.cvtColor(face, cv.COLOR_BGR2GRAY)

                faces.append(face)
                labels.append(int(id))
            except:
                pass
    return np.array(faces), np.array(labels)

faces, labels = create_train()
lbph.train(faces, labels)
lbph.save('Classifiers/TrainedLBPH.yml')
print('Treinamento finalizado!')
```

Código 4: Ajuste do modelo de reconhecimento facial utilizando o algoritmo LBPH.

Além disso, durante a criação do modelo de reconhecimento, é possível ajustar o parâmetro de limiar (*threshold*) do algoritmo *LBPH*, o qual define a distância máxima aceitável entre uma imagem de entrada e os dados armazenados para que um rosto seja considerado reconhecido. No código implementado, esse valor é inicialmente definido como 500 por meio do comando `cv.face.LBPHFaceRecognizer_create(threshold=500)`.

A modificação desse valor influencia diretamente o desempenho do sistema: limiares mais baixos tornam o reconhecimento mais restritivo, reduzindo falsos positivos, mas aumentando a chance de falsos negativos. Já valores mais altos tornam o modelo mais permissivo, podendo reconhecer rostos não cadastrados. Dessa forma, durante os testes, ajustes foram realizados com diferentes valores de *threshold* a fim de observar o comportamento do sistema e calibrar o equilíbrio entre precisão e sensibilidade.

A alteração desse limiar permite controlar a sensibilidade do sistema. Valores menores tornam o algoritmo mais restritivo, reduzindo o risco de falsos positivos, porém aumentando a possibilidade de não reconhecer rostos válidos (falsos negativos). Por outro lado, valores maiores tornam o reconhecimento mais permissivo, aumentando a taxa de acertos, mas também o risco de identificar erroneamente rostos não cadastrados.

Durante o desenvolvimento do sistema, foram realizados testes empíricos com diferentes valores de `threshold`, a fim de avaliar o comportamento do modelo frente a variações na iluminação, posição e expressões faciais. Esse ajuste foi feito diretamente na criação do modelo, sendo possível refinar o desempenho do sistema conforme o contexto de aplicação. O treinamento pode ser observado na Figura 7.

```
PS C:\Users\Ana Paula\Desktop\TG_ARQUIVO_FINAL> & "C:\Users\
Ana Paula\AppData\Local\Programs\Python\Python36\python.exe"
 Train.py
Treinamento iniciado.
Treinamento finalizado!
PS C:\Users\Ana Paula\Desktop\TG_ARQUIVO_FINAL>
```

Figura 7: Resultado do processo de treinamento. Fonte: Autor.

4.4 Reconhecimento Facial

Após a fase de aprendizagem do modelo LBPH e o armazenamento do classificador treinado, realiza-se a etapa de reconhecimento facial. Esta fase é responsável por identificar rostos previamente cadastrados no sistema, utilizando a *webcam* como fonte de captura contínua de imagens.

O processo de reconhecimento facial tem início com a leitura do arquivo `id-names.csv`, que contém os pares de identificação (`id`) e nomes dos indivíduos registrados no banco de imagens. Em seguida, o classificador Haar Cascade (`haarcascade_frontalface_default.xml`) é carregado para realizar a detecção de rostos em tempo real.

O modelo previamente treinado é recuperado por meio da função `FaceRecognizer_create`, atribuindo-se o mesmo valor de `threshold` utilizado na fase de treinamento. Em seguida, o classificador é carregado com o método `read()`, utilizando o arquivo `TrainedLBPH.yml`, que armazena os dados do modelo.

A *webcam* é ativada com a função `cv.VideoCapture(0)`, e um laço principal é iniciado. A cada iteração, um novo frame é capturado, convertido para escala de cinza com `cv.cvtColor()`, e processado pelo Haar Cascade com o método `detectMultiScale()`, que identifica as regiões faciais no quadro atual.

Cada região facial detectada é então extraída, redimensionada para `220x220` pixels e passada ao método `predict()` do classificador LBPH. Esse método retorna dois valores: o rótulo previsto (`label`) e o nível de confiança (`trust`) associado à predição. Caso o rótulo esteja presente no arquivo `id-names.csv`, o nome correspondente é recuperado e exibido sobre a imagem capturada, juntamente com um retângulo delimitando a região facial detectada.

O sistema permanece em execução até que a tecla `q` seja pressionada. Nesse momento, os recursos da câmera são liberados com `camera.release()` e todas as janelas abertas são encerradas com `cv.destroyAllWindows()`. A lógica pode ser observada no bloco de Código 5.

```
id_names = pd.read_csv('id-names.csv')
id_names = id_names[['id', 'name']]

faceClassifier = cv.CascadeClassifier('Classifiers/haarface.xml')

lbph = cv.face.LBPHFaceRecognizer_create(threshold=500)
lbph.read('Classifiers/TrainedLBPH.yml')

camera = cv.VideoCapture(0)

while cv.waitKey(1) & 0xFF != ord('q'):
    _, img = camera.read()
    grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    faces = faceClassifier.detectMultiScale(grey, scaleFactor=1.1,
       minNeighbors=4)

    for x, y, w, h in faces:
        faceRegion = grey[y:y + h, x:x + w]
        faceRegion = cv.resize(faceRegion, (220, 220))

        label, trust = lbph.predict(faceRegion)

        # Verifica se o rotulo existe no CSV
        match = id_names[id_names['id'] == label]

        if not match.empty:
            name = match['name'].item()
        else:
```

```
            name = "Rosto nao cadastrado"

        # Desenha retangulo e nome (ou mensagem de erro)
        cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
        cv.putText(img, name, (x, y + h + 30), cv.
           FONT_HERSHEY_COMPLEX, 1, (0, 0, 255))

    cv.imshow('Recognize', img)

camera.release()
cv.destroyAllWindows()
```

Código 5: Código de reconhecimento facial em tempo real utilizando LBPH.

O funcionamento pode ser observado na Figura 8.

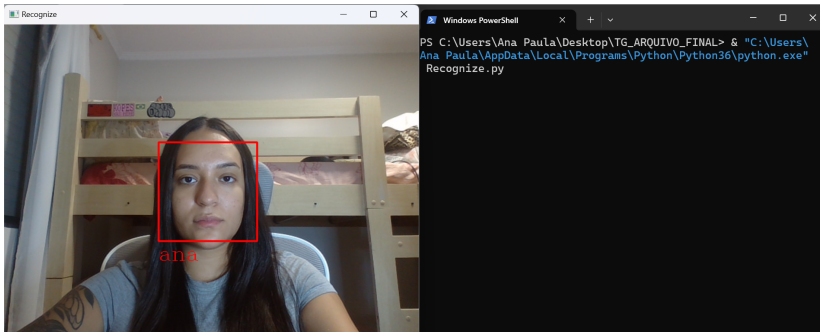

Figura 8: Resultado do processo de reconhecimento. Fonte: Autor.

4.5 Métricas de Desempenho

Para a verificação das métricas de desempenho do sistema de reconhecimento facial, foi elaborado um código específico para a captura de imagens destinadas exclusivamente à fase de teste. As imagens obtidas por meio desse processo não são utilizadas no refinamento do modelo, o que assegura uma avaliação imparcial e condizente com a capacidade de generalização do classificador frente a dados inéditos. Para evitar qualquer interferência com a base de dados original, os arquivos de teste são armazenados em uma estrutura de diretórios distinta daquela empregada durante o treinamento.

Além disso, foi implementado um segundo código com o objetivo de executar os testes de acurácia do modelo. Esse script percorre as imagens da base de teste, realiza as predições com o modelo previamente treinado e compara os resultados com os rótulos reais. Como saída, são fornecidas métricas de avaliação como acurácia, precisão, revocação (recall) e F1-score, além de uma matriz de confusão que permite uma análise detalhada do desempenho do sistema em cada classe.

5. Resultados e Discussões

5.1 Resultados

Antes de proceder com a análise quantitativa por meio das métricas de desempenho, esta subseção tem por objetivo apresentar visualmente alguns casos representativos observados durante os testes realizados com o sistema de reconhecimento facial. Esses exemplos ilustram o comportamento do experimento em diferentes cenários práticos, permitindo uma melhor compreensão das situações em que o classificador acerta ou comete erros.

5.1.1 Experimento 01. Testes realizados com *Threshold* de 500

Conforme mencionado anteriormente, no algoritmo *LBPH*, o processo de reconhecimento consiste em comparar a imagem da face capturada com os vetores de características previamente armazenados durante o treinamento. A comparação é feita utilizando a distância *Chi-Square* entre os histogramas das imagens, que é a métrica padrão empregada pelo OpenCV quando nenhum método de distância é explicitamente definido.

O valor de limiar (*threshold*) representa a distância máxima aceitável para que uma correspondência seja considerada válida. Em outras palavras, se a distância entre o vetor da face capturada e o vetor mais próximo no banco de dados for inferior ou igual ao limiar, o sistema aceita essa previsão como correta e retorna o rótulo correspondente. Caso contrário, a face é considerada desconhecida.

Durante os testes realizados com o limiar de 500, foram identificados diversos casos de falsos positivos, nos quais o sistema reconheceu incorretamente um indivíduo como sendo outro previamente cadastrado. Como exemplo representativo, apresenta-se uma situação em que o experimento classificou erroneamente uma pessoa como pertencente a uma identidade já existente no banco de dados. Apesar de se tratarem de indivíduos distintos, ambos apresentavam características visuais semelhantes, como pele parda, cabelos longos e o uso de óculos, o que pode ter contribuído para a confusão do classificador. Além disso, a iluminação indireta e a baixa nitidez da imagem testada afetaram a qualidade da extração de características, favorecendo a ocorrência do erro. Esse resultado indica que o *threshold* adotado (500) pode estar elevado demais para o nível de variabilidade presente na base de dados, tornando o sistema mais suscetível a reconhecimentos indevidos. Esse comportamento pode ser observado nas Figuras 9 e 10.

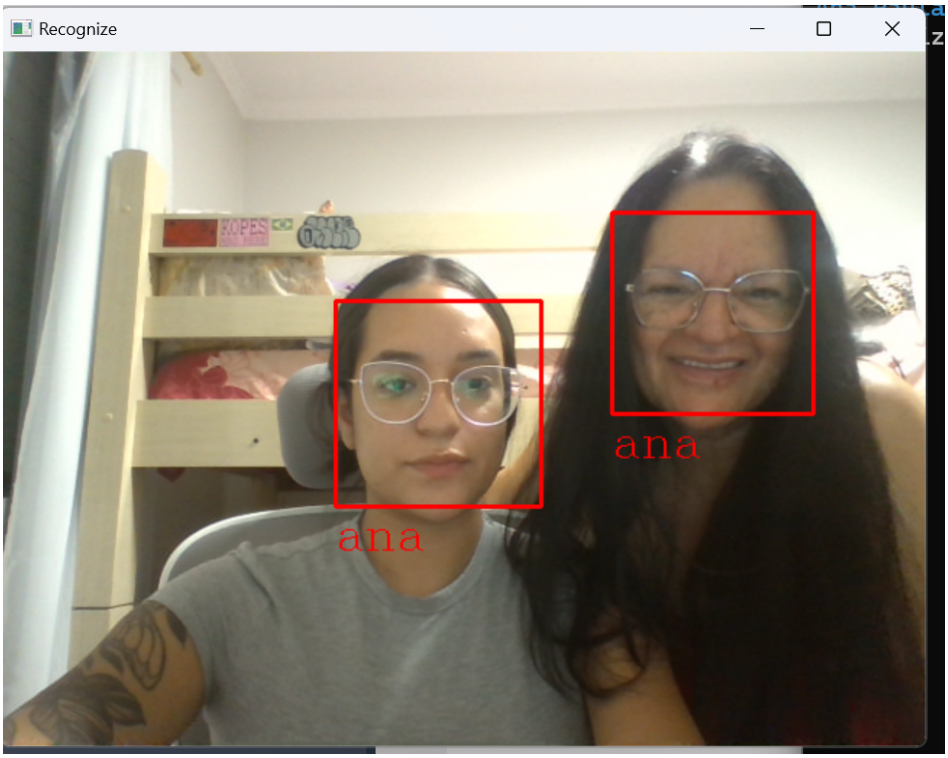

Figura 9: Exemplo de falso positivo. Fonte: Autor.

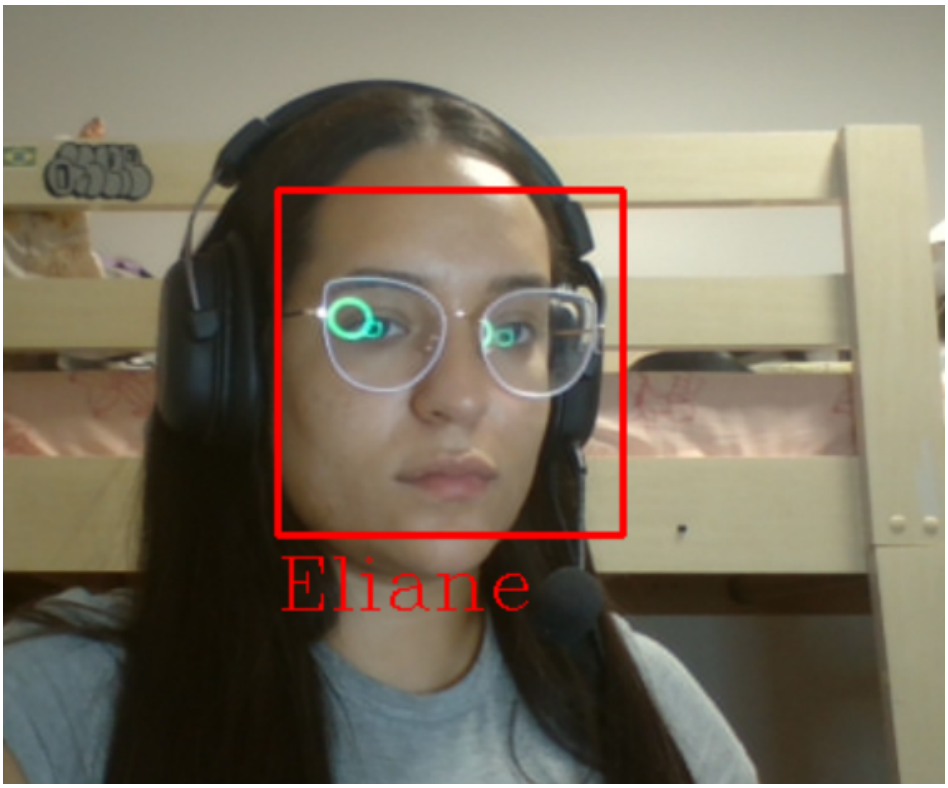

Figura 10: Exemplo de falso positivo, reconheceu incorretamente. Fonte: Autor.

Outro exemplo relevante de falso positivo pode ser observado na Figura 11, na qual o sistema realiza **duas identificações simultâneas incorretas** da mesma pessoa como "Ana", nome associado a uma identidade previamente cadastrada. Nesse caso, além do retângulo principal demarcando corretamente a região da face, o sistema também classificou erroneamente uma **região do pescoço** como sendo uma nova ocorrência da mesma pessoa.

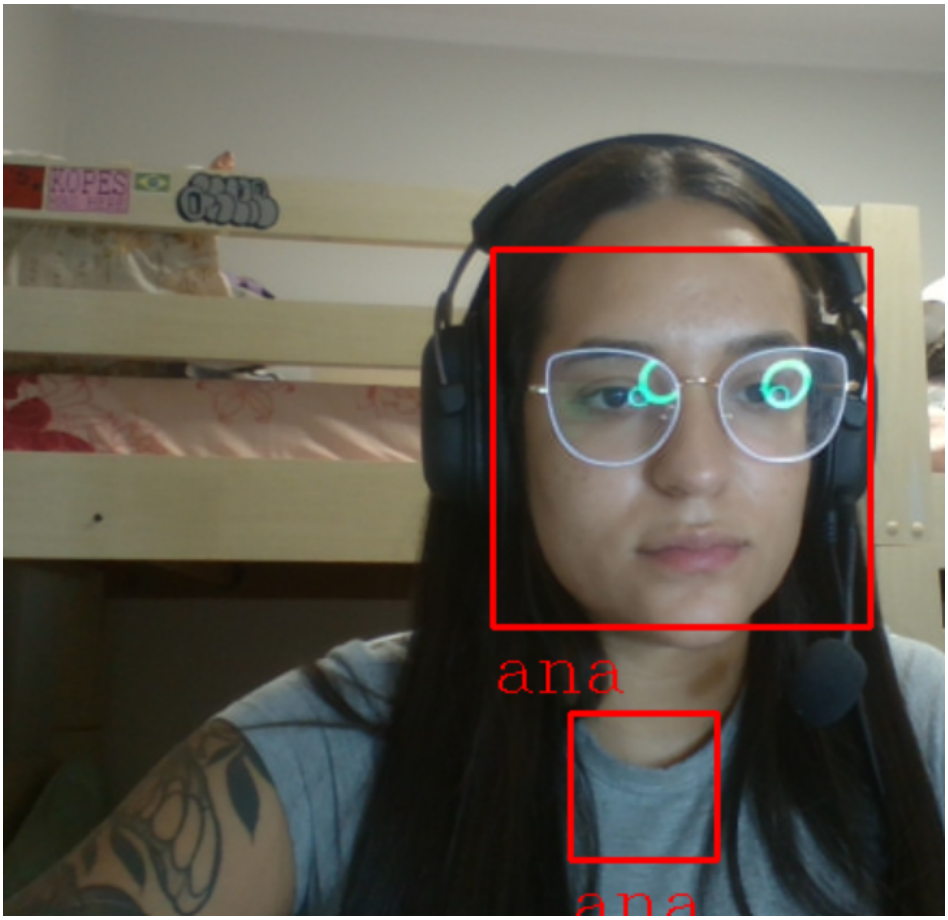

Figura 11: Exemplo de falso positivo: o sistema reconheceu a mesma pessoa duas vezes, incluindo uma região incorreta no pescoço como sendo um rosto. Fonte: Autor.

Esse tipo de comportamento evidencia falhas na segmentação da região facial, indicando que o classificador *Haar Cascade* pode, em determinadas condições, detectar regiões que *não correspondem a faces reais*, mas que, quando submetidas ao experimento *LBPH*, acabam sendo aceitas devido à **semelhança estatística dos histogramas locais**.

Essa ocorrência reforça a necessidade de ajustes no sistema, especialmente no que diz respeito a:

- Refinamento do classificador Haar (por exemplo, ajustando o parâmetro `minNeighbors` ou aplicando validações geométricas adicionais);
- Redução do threshold para tornar o experimento mais seletivo;

- Complementação do pipeline com etapas adicionais de validação (como verificação geométrica da posição relativa dos olhos ou simetria facial).

Esse resultado demonstra que, mesmo quando o reconhecimento nominal ocorre, ele pode ser tecnicamente inválido se não estiver associado a uma **região facial legítima**.

Outro cenário em que o sistema apresentou limitação significativa foi em ambientes com baixa luminosidade. A Figura 12 ilustra um caso no qual a face capturada em um ambiente escuro foi incorretamente reconhecida como sendo da usuária "Eliane" — embora se trate de outro indivíduo previamente cadastrado: "Ana".

Esse erro evidencia a sensibilidade do algoritmo *LBPH* à qualidade da imagem de entrada. Em condições de pouca luz, a quantidade de detalhes texturais extraídos da face é reduzida, comprometendo a geração de um vetor de características confiável. Como consequência, o algoritmo pode associar indevidamente o padrão capturado a outro vetor presente no experimento treinado, especialmente quando o threshold está ajustado para um valor mais permissivo, como 500.

Figura 12: Exemplo de falso positivo sob iluminação insuficiente: o sistema reconheceu incorretamente o rosto como sendo da usuária "Eliane". Fonte: Autor.

Apesar dos desafios mencionados anteriormente, como sensibilidade à iluminação, falsos positivos e reconhecimentos indevidos, o sistema foi capaz de identificar corretamente indivíduos em condições mais favoráveis. A Figura 13 mostra um exemplo em que o reconhecimento foi bem-sucedido, com a identificação correta da usuária "ana". Nesse caso, observa-se que o ambiente apresentava uma iluminação adequada e a imagem capturada possuía boa definição e contraste facial.

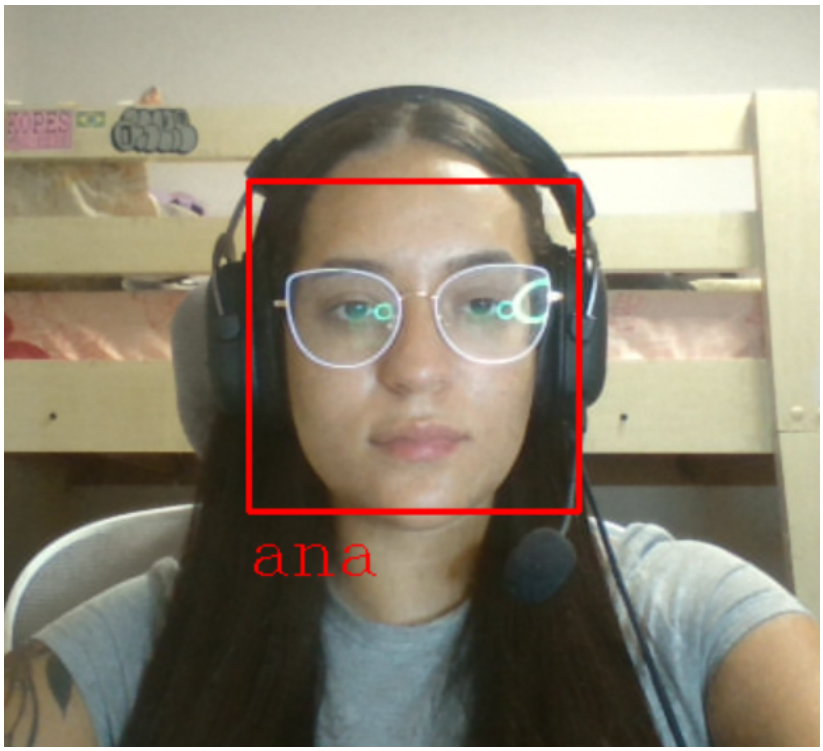

Figura 13: Reconhecimento facial correto em ambiente iluminado, com identificação precisa da usuária “Ana”. Fonte: Autor.

5.1.2 Avaliação de Desempenho do Experimento 01

Para a avaliação da acurácia do experimento de reconhecimento facial utilizando o algoritmo *LBPH* (*Local Binary Patterns Histograms*), foram utilizados dois conjuntos de imagens:

- **30 imagens de pessoas não cadastradas**, pertencentes ao grupo de “desconhecidos”, com o objetivo de testar a capacidade do sistema em rejeitar rostos não treinados;
- **45 imagens de pessoas cadastradas**, divididas igualmente entre três usuários: Ana, Eliane e João, totalizando 15 imagens de teste para cada indivíduo. Ressalta-se, ainda, que as imagens dos usuários cadastrados utilizadas para teste não foram as mesmas utilizadas para o treinamento.

A Figura 14 apresenta a matriz de confusão obtida, enquanto a Tabela 1 exibe o relatório de classificação com métricas de precisão (*precision*), revocação (*recall*) e F1-score para cada classe.

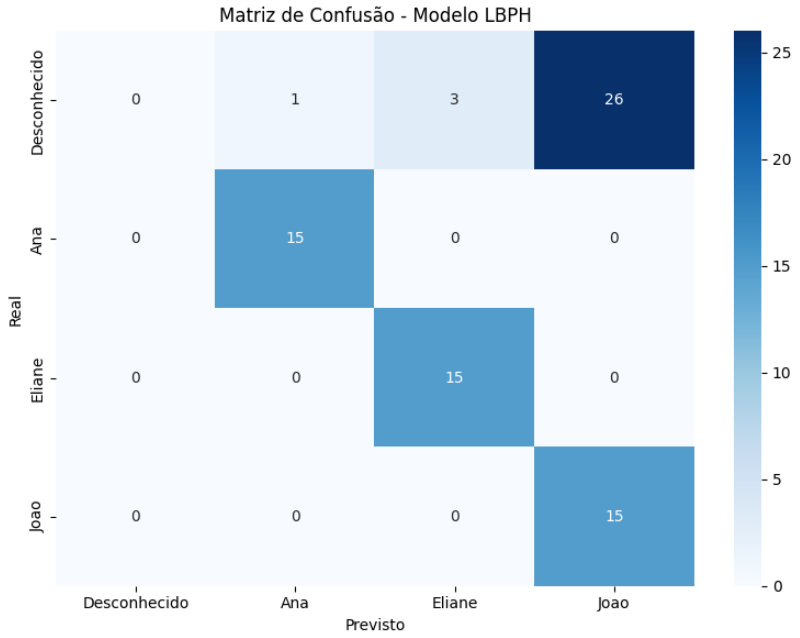

Figura 14: Matriz de Confusão — experimento LBPH, limiar 500. Fonte: Autor.

Classe	**Precisão**	**Revocação**	**F1-Score**	**Suporte**
Desconhecido	0.00	0.00	0.00	30
Ana	0.94	1.00	0.97	15
Eliane	0.83	1.00	0.91	15
João	0.37	1.00	0.54	15
Acurácia		0.60		
Média Macro	0.53	0.75	0.60	75
Média Ponderada	0.43	0.60	0.48	75

Tabela 1: Relatório de Classificação — *Threshold* 500. Fonte: Autor.

Observa-se que o experimento obteve **revocação (recall) de 100% para todas as classes cadastradas**, ou seja, todas as imagens dos usuários Ana, Eliane e João foram corretamente reconhecidas como pertencentes às suas respectivas classes. Isso demonstra que o experimento está altamente sensível a qualquer traço característico presente nos rostos previamente cadastrados, o que elimina completamente a ocorrência de falsos negativos neste cenário.

No entanto, esse comportamento também indica uma **tendência excessivamente permissiva do sistema**. Isso é evidenciado pelo desempenho da classe *Desconhecido*, que apresentou precisão, revocação e F1-score igual a 0. Dos 30 rostos não cadastrados utilizados no teste, nenhum foi corretamente identificado como desconhecido. Ao contrário, 26 dessas imagens foram classificadas incorretamente como sendo do usuário João, enquanto outras foram atribuídas erroneamente às classes Ana e Eliane.

Esse cenário caracteriza a presença significativa de **falsos positivos**, o que compromete

a confiabilidade do sistema em ambientes nos quais a rejeição de indivíduos não autorizados é uma exigência crítica.

A **acurácia total do experimento foi de 60%**, com uma média macro ponderada de F1-score igual a 0,60. A discrepância entre a precisão das classes cadastradas (por exemplo, Ana com 0,94) e a classe *Desconhecido* (0,00) evidencia a necessidade de ajustes no experimento.

5.1.3 experimento 02. Testes realizados com *Threshold* de 5

Com o objetivo de testar a capacidade do sistema em rejeitar rostos não cadastrados, foi realizado um novo teste reduzindo o valor do parâmetro *threshold* para 5. O resultado dessa configuração está representado na Figura 15.

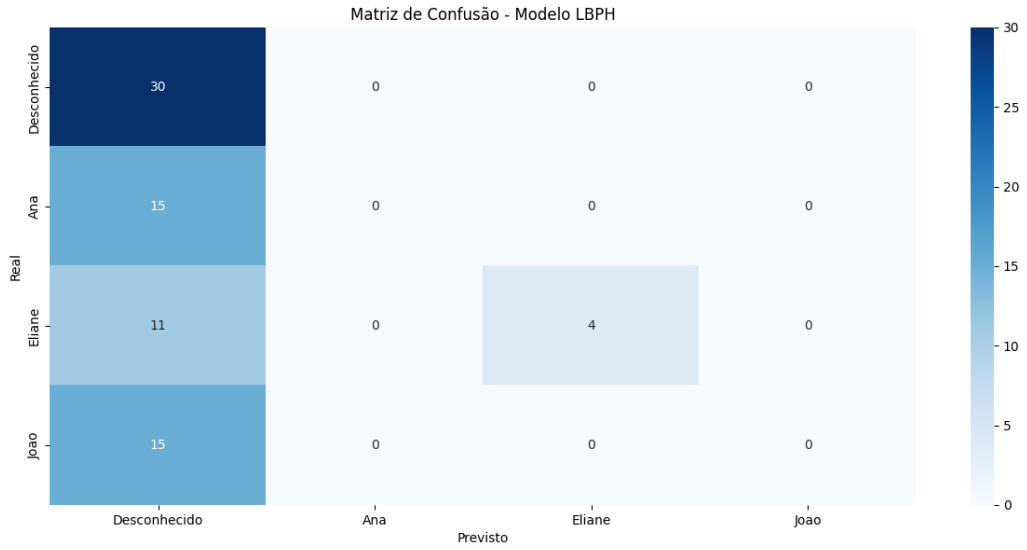

Figura 15: Matriz de Confusão — *Threshold* = 5. Fonte: Autor.

Nesta configuração, o parâmetro *threshold* foi ajustado para o valor 5, tornando o sistema extremamente rigoroso quanto à aceitação de uma correspondência. Com esse limiar, apenas classificações com altíssima confiança são aceitas, o que impacta diretamente o desempenho do experimento.

5.1.4 Avaliação de Desempenho do experimento 02

A análise desses resultados indica que, com `threshold` igual a 5, o experimento tornou-se excessivamente rigoroso, classificando quase todas as imagens — inclusive aquelas de usuários cadastrados — como *Desconhecido*. Isso resultou em uma alta taxa de **falsos negativos**, onde apenas 4 rostos cadastrados foram reconhecidos corretamente.

Classe	Precisão	Revocação	F1-Score	Suporte
Desconhecido	0.42	1.00	0.59	30
Ana	0.00	0.00	0.00	16
Eliane	1.00	0.27	0.42	15
João	0.00	0.00	0.00	15
Acurácia Total		44,74%		
Média Macro	0.35	0.32	0.25	
Média Ponderada	0.36	0.45	0.32	

Tabela 2: Relatório de Classificação — *Threshold* = 5. Fonte: Autor.

A matriz de confusão apresentada na Figura 16 demonstra que, com essa configuração, o sistema classificou corretamente todas as 30 imagens de indivíduos não cadastrados como *Desconhecido*, alcançando revocação de 100% para essa classe e evitando qualquer falso positivo. No entanto, o desempenho sobre os usuários cadastrados foi bastante comprometido: das 45 imagens de rostos conhecidos, apenas 4 foram corretamente reconhecidas (todas da classe *Eliane*), enquanto 41 foram incorretamente rejeitadas como desconhecidas.

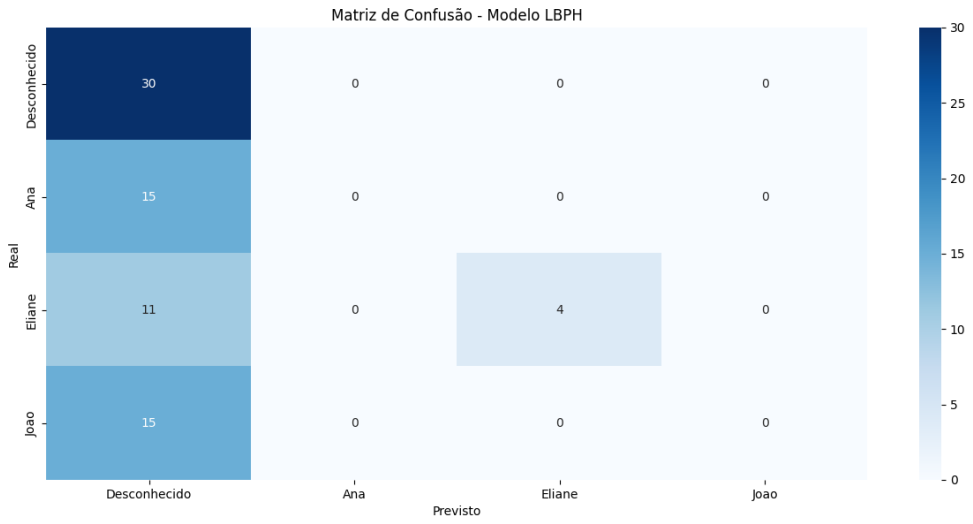

Figura 16: Matriz de Confusão — *Threshold* = 5. Fonte: Autor.

A revocação de 100% da classe *Desconhecido* reflete esse comportamento: todas as imagens, mesmo as pertencentes aos usuários cadastrados, foram incorretamente rejeitadas. Consequentemente, a acurácia geral do sistema caiu para 44,74%, com F1-scores nulos para as classes Ana e João, que são conhecidas. As Figuras 17 e 18 ilustram o cenário de falso negativo, descrito anteriormente:

Figura 17: Falso negativo — *Threshold* = 5. Fonte: Autor.

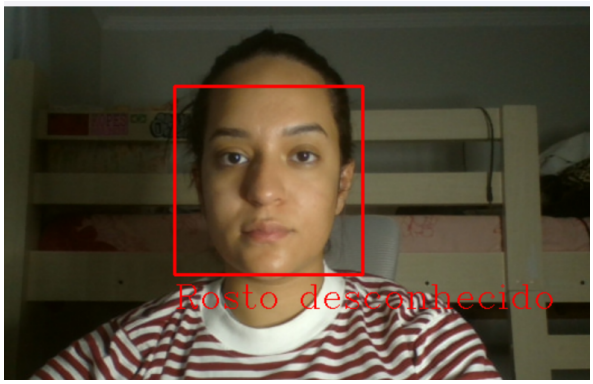

Figura 18: Falso negativo — *Threshold* = 5. Fonte: Autor.

Esse cenário mostra que, embora a redução do `threshold` tenha melhorado a rejeição de rostos desconhecidos, ela comprometeu severamente a capacidade do sistema de reconhecer rostos válidos. Isso demonstra a importância de calibrar esse parâmetro de forma balanceada, de modo a garantir um desempenho adequado tanto na aceitação quanto na rejeição de rostos.

5.1.5 Experimento 03. Testes realizados com *Threshold* de 50

Após a realização de diversos testes com diferentes valores de `threshold`, o valor 50 se mostrou o mais equilibrado entre reconhecimento e rejeição. Esse limiar proporcionou um bom desempenho tanto na identificação correta de usuários cadastrados quanto na rejeição de rostos desconhecidos, como evidenciado na Tabela 3 e na Figura 19.

5.1.6 Avaliação de Desempenho do Experimento 03

Classe	**Precisão**	**Revocação**	**F1-Score**	**Suporte**
Desconhecido	0.79	1.00	0.88	30
Ana	1.00	0.67	0.80	15
Eliane	0.88	0.93	0.90	15
João	1.00	0.73	0.85	15
Acurácia Total		86,67%		
Média Macro	0.92	0.83	0.86	
Média Ponderada	0.89	0.87	0.86	

Tabela 3: Relatório de Classificação — *Threshold* = 50. Fonte: Autor.

A Figura 19 apresenta a matriz de confusão obtida. O sistema atingiu acurácia de **86,67%**, com reconhecimento correto de **77,78% dos rostos cadastrados** e rejeição total dos desconhecidos (**100% de revocação na classe *Desconhecido***). Foram identificados apenas 8 falsos negativos — casos em que o experimento não reconheceu corretamente um usuário válido — e dois falsos positivos.

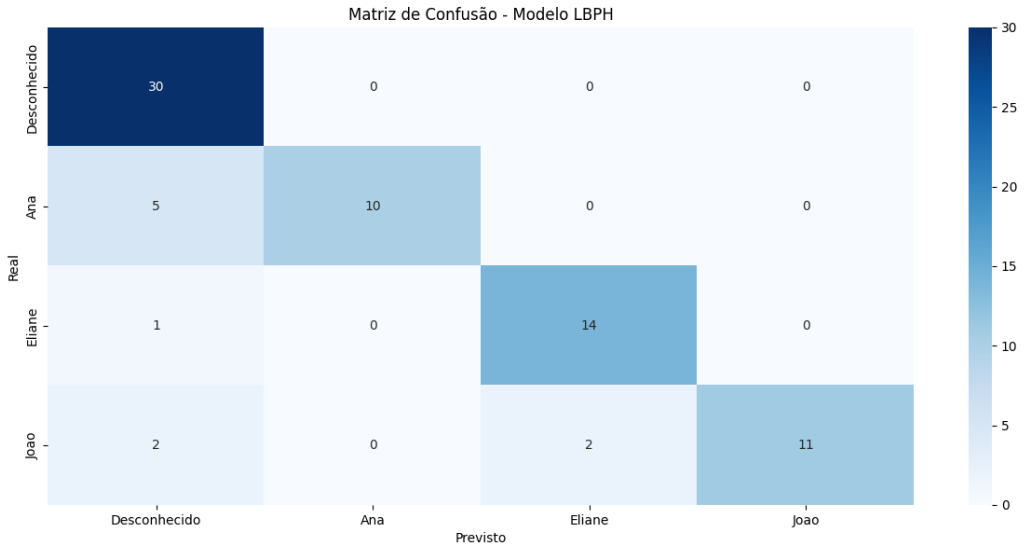

Figura 19: Matriz de Confusão — *Threshold* = 50. Fonte: Autor.

Um exemplo de verdadeiro positivo pode ser observado na Figura 20:

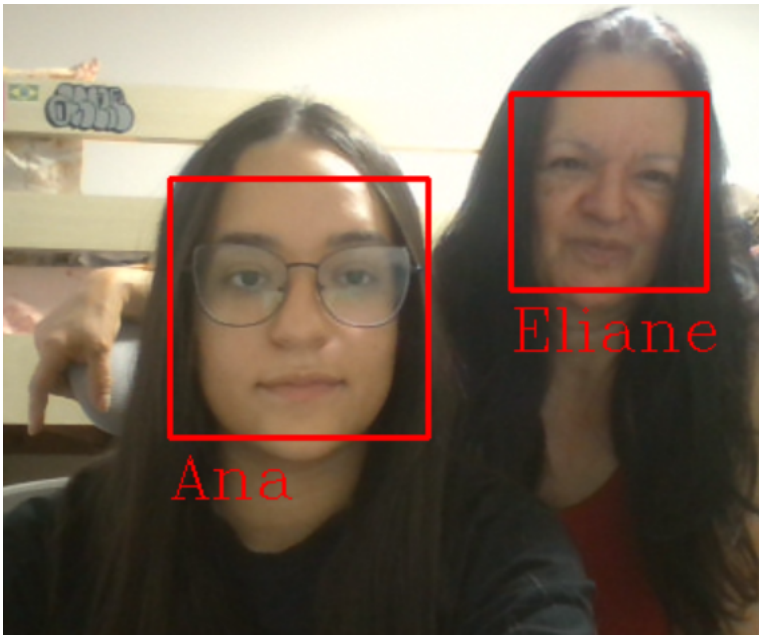

Figura 20: Verdadeiro Positivo - *Threshold* = 50. Fonte: Autor.

E um exemplo de verdadeiro negativo, pode ser observado na Figura 21:

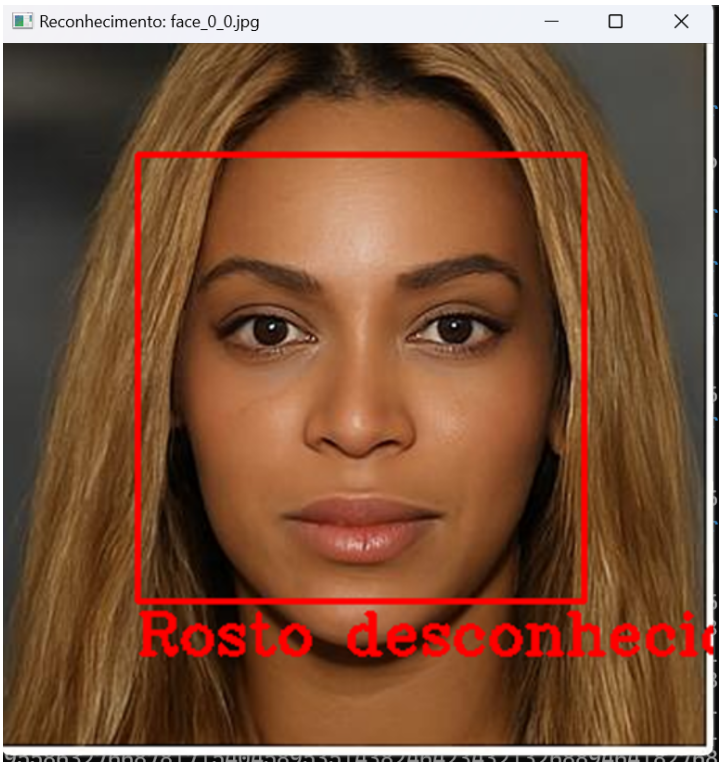

Figura 21: Verdadeiro Negativo - *Threshold* = 50. Fonte: Autor.

5.2 Resumo Comparativo dos Resultados

A Tabela 4 apresenta um resumo consolidado dos principais resultados obtidos para os três experimentos testados, variando o parâmetro `threshold` nos valores de 500, 5 e 50. Esta análise evidencia a sensibilidade do desempenho do sistema frente às alterações neste parâmetro, fundamental no controle da distância máxima permitida entre vetores de características faciais para que uma predição seja considerada válida.

Threshold	**Acur. Total**	**Acur. Conhecido**	**Acur. Desconhecido**	**FN**
500	60,00%	100,00%	0,00%	0
5	44,74%	8,70%	100,00%	41
50	**86,67%**	**77,78%**	**100,00%**	**8**

Tabela 4: Resumo comparativo dos resultados com diferentes valores de *threshold*

Experimento 01 — *Threshold* = 500: apresentou ótimo desempenho na identificação de indivíduos cadastrados, atingindo 100% de revocação nas três classes conhecidas (Ana, Eliane e João). No entanto, a ausência de qualquer rejeição para rostos não cadastrados revelou um problema crítico: o sistema classificou erroneamente todos os indivíduos desconhecidos como usuários válidos, resultando em 30 falsos positivos. Este comportamento indica que o valor de limiar estava excessivamente alto, tornando o experimento muito permissivo e comprometendo a segurança do sistema, especialmente em cenários onde a autenticação correta é essencial.

Experimento 02 — *Threshold* = 5: adotou uma abordagem oposta, sendo extremamente rigoroso na aceitação de uma correspondência. O sistema rejeitou corretamente todos os 30 indivíduos desconhecidos, demonstrando 100% de revocação para essa classe. Contudo, apenas 4 imagens de rostos válidos foram reconhecidas corretamente, o que gerou 41 falsos negativos. A acurácia total caiu para 44,74%, evidenciando que um *threshold* muito baixo impede o reconhecimento mesmo de usuários válidos, afetando diretamente a usabilidade do sistema.

Experimento 03 — *Threshold* = 50: revelou-se a configuração mais equilibrada entre os extremos anteriores. O sistema obteve acurácia total de 86,67%, com 77,78% de revocação nas classes conhecidas e 100% de acerto na rejeição de indivíduos desconhecidos. Foram registrados 8 falsos negativos e 2 falsos positivos, ambos decorrentes de classificações incorretas entre classes cadastradas — mais especificamente, imagens reais do usuário João foram identificadas como pertencentes à usuária Eliane. Apesar dessas confusões pontuais, essa configuração ainda representa um excelente compromisso entre segurança e desempenho, sendo a mais adequada entre as três, para aplicações práticas em contextos controlados, como sistemas de controle de acesso ou identificação restrita.

Com o auxílio do *Matplotlib*, gerou-se um gráfico para melhor visualização dos resultados. A Figura 22 ilustra o comportamento do sistema ao longo dos diferentes *thresholds* testados.

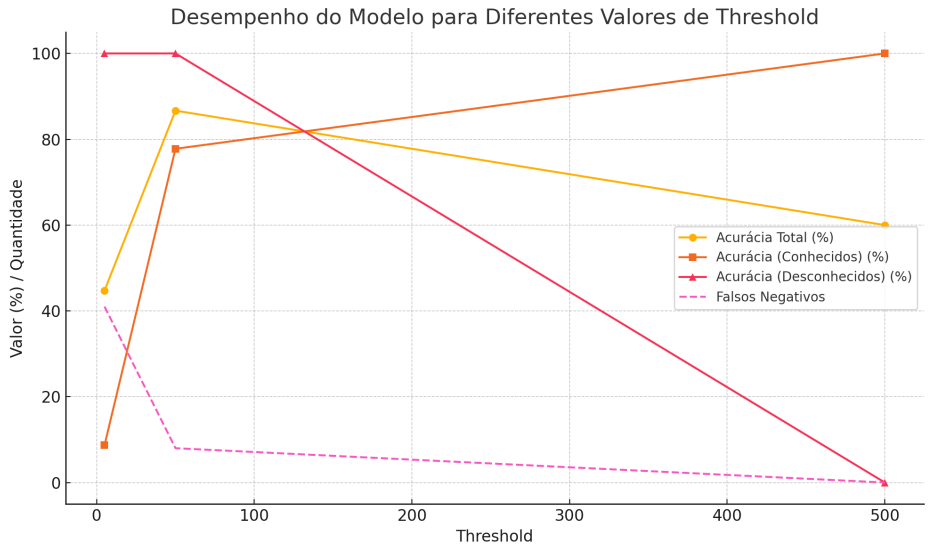

Figura 22: Comparação de desempenho para diferentes valores de *threshold*

No gráfico, cada linha representa uma métrica de avaliação:

- **Acurácia Total (%)** — mostra o percentual geral de acertos do sistema, considerando tanto rostos cadastrados quanto desconhecidos. O pico é atingido com *threshold* = 50.
- **Acurácia (Conhecidos) (%)** — representa a taxa de reconhecimento correto dos usuários previamente cadastrados. Observa-se que valores muito baixos de *threshold* comprometem esse reconhecimento, ao passo que valores muito altos favorecem acertos, mas com o risco de aceitar também rostos indevidos.

- **Acurácia (Desconhecidos) (%)** — indica a capacidade do sistema de rejeitar corretamente rostos não cadastrados. A curva é constante e máxima (100%) para *thresholds* mais baixos, mas cai para zero quando o limiar é alto, como no caso de 500.
- **Falsos Negativos** — métrica inversa à acurácia dos conhecidos. Um alto número de falsos negativos indica que o sistema está rejeitando indevidamente usuários válidos, o que ocorre drasticamente com *threshold* = 5.

Nota-se que a configuração com `threshold` = 50 apresenta o melhor equilíbrio entre as curvas: garante alta acurácia tanto na aceitação de usuários legítimos quanto na rejeição de rostos não cadastrados, com número reduzido de falsos negativos e ausência de falsos positivos.

6. Conclusão

O desenvolvimento do sistema de reconhecimento facial apresentado neste trabalho demonstrou, de forma prática e fundamentada, que é plenamente viável construir uma solução eficiente, acessível e multiplataforma utilizando exclusivamente ferramentas de código aberto, como Python e OpenCV. Operando de forma autônoma, localmente, e com recursos computacionais modestos, o sistema obteve uma acurácia média de 87%, evidenciando não apenas a robustez da abordagem adotada, mas também seu potencial de aplicação em cenários reais e cotidianos.

A arquitetura proposta, baseada na combinação do classificador Haar Cascade para detecção facial e do algoritmo Local Binary Patterns Histograms (LBPH) para reconhecimento, mostrou-se particularmente eficaz em ambientes que demandam simplicidade, leveza e independência de infraestrutura avançada. Com foco em aplicações como o controle de acesso em portarias de condomínios e pequenos estabelecimentos comerciais, a solução desenvolvida oferece uma alternativa concreta, de baixo custo e de fácil manutenção, especialmente atrativa para contextos com limitações de conectividade ou ausência de suporte técnico especializado.

Durante a fase de testes, o sistema foi capaz de identificar corretamente usuários previamente cadastrados mesmo diante de variações sutis em expressões faciais e ângulos de captura. Apesar das limitações naturais da abordagem — como a sensibilidade a condições de iluminação adversas e a ocorrência de falsos positivos em limiares de decisão mais permissivos — os resultados foram amplamente satisfatórios. A realização de experimentos com diferentes valores de *threshold* demonstrou, inclusive, a importância da calibração criteriosa desse parâmetro para equilibrar precisão e sensibilidade.

Além disso, o sistema foi construído de forma modular e expansível, facilitando tanto sua adaptação a outros ambientes quanto sua evolução futura. O uso de arquivos `.csv` para armazenar dados de usuários, embora funcional para fins experimentais, abre espaço para a substituição por bancos de dados relacionais mais robustos e seguros em contextos produtivos. Da mesma forma, a adoção de câmeras de melhor resolução e fontes de luz artificial poderia ampliar significativamente a taxa de reconhecimento e a confiabilidade do sistema em ambientes menos controlados.

Vale destacar que, mesmo com as restrições impostas pelo ambiente experimental, o sistema se mostrou funcional, escalável e promissor, apontando para uma excelente relação entre simplicidade arquitetural e desempenho obtido. O fato de alcançar 87% de acurácia com um modelo leve, executado localmente e sem aceleração por hardware gráfico, demonstra a maturidade técnica do projeto e valida a escolha dos algoritmos utilizados.

Para além do mérito técnico, este trabalho também se destaca por sua contribuição social e acadêmica. Ao propor uma solução que une acessibilidade tecnológica, independência de infraestrutura e aplicabilidade prática, ele reforça o papel da engenharia como agente transformador da sociedade. A democratização do acesso a tecnologias biométricas, muitas vezes restritas a grandes corporações, torna-se possível com iniciativas como esta, alinhando-se aos princípios de inovação responsável, inclusão digital e uso ético da inteligência artificial.

Em síntese, os resultados superaram as expectativas iniciais e indicam um caminho fértil para o uso do reconhecimento facial em aplicações reais, especialmente aquelas que exigem baixo custo e alta confiabilidade. Este trabalho representa não apenas uma realização

acadêmica significativa, mas também um ponto de partida sólido para o desenvolvimento de soluções ainda mais completas, seguras, éticas e impactantes no futuro — com potencial de beneficiar diretamente a sociedade por meio da tecnologia.

Referências

[1] LI, S. Z.; JAIN, A. K. (Ed.). *Handbook of Face Recognition*. 2. ed. London: Springer, 2011.

[2] Zhao, W.; Chellappa, R.; Phillips, P. J.; Rosenfeld, A. *Face recognition: a literature survey*. ACM Computing Surveys (CSUR), v. 35, n. 4, p. 399–458, 2003.

[3] Viola, P.; Jones, M. *Rapid object detection using a boosted cascade of simple features*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2001.

[4] Turk, M.; Pentland, A. *Eigenfaces for recognition*. Journal of Cognitive Neuroscience, v. 3, n. 1, p. 71–86, 1991.

[5] OpenCV. *Introduction to OpenCV.js*. Disponível em: `https://docs.opencv.org/3.4/df/d0a/tutorial_js_intro.html`. Acesso em: 5 abr. 2025.

[6] A. C.; LÜTGE, C. *Vigilância e Relações de Poder – O Uso de Tecnologias de Reconhecimento Facial e Identificação Biométrica a Distância em Espaço Público e Impactos na Vida Pública*. Revista Direito Público, [S. l.], v. 18, n. 100, 2022.

[7] SCHAPIRE, R. E. *Explaining AdaBoost*. In: SCHÖLKOPF, B.; LUO, Z.; VOVK, V. (eds.). *Empirical Inference*. Berlin, Heidelberg: Springer, 2013. Disponível em: `https://doi.org/10.1007/978-3-642-41136-6_5`. Acesso em: 9 abr. 2025.

[8] Ahonen, T.; Hadid, A.; Pietikäinen, M. *Face description with local binary patterns: application to face recognition*. IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 28, n. 12, p. 2037–2041, 2006.

[9] Yang, M. H.; Kriegman, D. J.; Ahuja, N. *Detecting faces in images: a survey*. IEEE Transactions on Pattern Analysis and Machine Intelligence, v. 24, n. 1, p. 34–58, 2002.

[10] Brunelli, R., & Poggio, T. *Face recognition: Features versus templates. IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(10), 1042-1052., 1993*

[11] Taigman, Y.; Yang, M.; Ranzato, M. A.; Wolf, L. *DeepFace: Closing the gap to human-level performance in face verification*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, p. 1701–1708, 2014.

[12] OpenCV. *Open Source Computer Vision Library*. Disponível em: `https://opencv.org/`. Acesso em: 4 abr. 2025.

[13] OpenCV. *Histogram Comparison*. Disponível em: `https://docs.opencv.org/3.4/d8/dc8/tutorial_histogram_comparison.html`. Acesso em: 8 abr. 2025.

[14] OpenCV. *Basic Thresholding Operations*. Disponível em: `https://docs.opencv.org/4.x/db/d8e/tutorial_threshold.html`. Acesso em: 12 abr. 2025.

[15] LEE, Seonho; PHANISHAYEE, Amar; MAHAJAN, Divya. *Forecasting GPU Performance for Deep Learning Training and Inference*. arXiv preprint arXiv:2407.13853v3, 2024. Disponível em: https://arxiv.org/abs/2407.13853v3. Acesso em: 12 abr. 2025.

[16] TIOBE Software. *TIOBE Index for April 2025*. Disponível em: https://www.tiobe.com/tiobe-index/. Acesso em: 5 abr. 2025.

[17] IEEE Spectrum. *Top Programming Languages 2024*. Disponível em: https://spectrum.ieee.org/top-programming-languages-2024. Acesso em: 5 abr. 2025.

[18] OpenCV. *Cascade Classifier Tutorial*. Disponível em: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html. Acesso em: 4 abr. 2025.

[19] SCIKIT-LEARN. *sklearn.ensemble.AdaBoostClassifier*. Disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html. Acesso em: 2 abr. 2025.

[20] Filho, G.; Ueyama, J.; Faiçal, B.; Guidoni, D.; Villas, L. (2015). *ResiDI: Um Sistema de Decisão Inteligente para Infraestruturas Residenciais via Sensores e Atuadores Sem Fio*.

[21] CARVALHO, Leandro de Lima. *LBPH Face Recognition*. 2021. Disponível em: https://github.com/leodlca/lbph-face-recognition. Acesso em: 13 abr. 2025.

[22] OpenCV. *Image file reading and writing – imread*. OpenCV Documentation, 2023. Disponível em: https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html#ga61f7b59a3c2f7b380be494c6f59b0c4d. Acesso em: 13 abr. 2025.

Apendice A – Implementação do Sistema

Este apêndice apresenta os códigos desenvolvidos para a implementação do sistema de reconhecimento facial, incluindo as etapas de captura de imagens, treinamento do modelo, reconhecimento facial em tempo real e avaliação de desempenho.

A.1 Captura de Imagens

O código abaixo realiza o cadastro do usuário, detecção facial via webcam e armazenamento das imagens em pasta correspondente ao seu ID.

```
import os
import numpy as np
import cv2 as cv
import pandas as pd
from datetime import datetime

if os.path.exists('id-names.csv'):
    id_names = pd.read_csv('id-names.csv')
    id_names = id_names[['id', 'name']]
else:
    id_names = pd.DataFrame(columns=['id', 'name'])
    id_names.to_csv('id-names.csv', index=False)

if not os.path.exists('faces'):
    os.makedirs('faces')

while True:
    print('Bem vindo(a)!')
    print('Por favor, insira o seu ID.')
    print('Se for a primeira vez, escolha um numero entre 1 e 300.')
    id = int(input('ID: '))
    name = ''

    if id in id_names['id'].values:
        existing_name = id_names[id_names['id'] == id]['name'].item
            ()
        print(f"O ID {id} ja esta associado ao nome: {existing_name
            }.")
        confirm = input(f"Voce confirma que e {existing_name}? (s/n)
            : ").strip().lower()
        if confirm == 's':
            name = existing_name
            print(f'Bem-vindo(a) de volta, {name}!')
            break
        else:
            print("Por favor, insira um novo ID.\n")
```

```
    elif os.path.exists(f'faces/{id}'):
        print(f"O ID {id} ja esta em uso. Escolha outro.")
    else:
        name = input('Por favor, insira o seu nome: ')
        os.makedirs(f'faces/{id}')
        id_names = pd.concat([id_names, pd.DataFrame([{'id': id, '
           name': name}])], ignore_index=True)
        id_names.to_csv('id-names.csv', index=False)
        print(f'Usuario {name} cadastrado com sucesso!')
        break

print("\nVamos iniciar a captura de imagens.\n")
print("Assim que aparecer um retangulo ao redor do seu rosto,
   pressione a tecla 's' para capturar uma imagem.")
input("\nPressione ENTER para comecar, e pressione a tecla 'q' para
   sair quando terminar.")

camera = cv.VideoCapture(0)
face_classifier = cv.CascadeClassifier('Classifiers/haarface.xml')
photos_taken = 0

while True:
    ret, img = camera.read()
    if not ret:
        print("Erro ao acessar a camera.")
        break

    grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    faces = face_classifier.detectMultiScale(grey, scaleFactor=1.1,
       minNeighbors=5, minSize=(50, 50))
    key = cv.waitKey(1) & 0xFF

    for (x, y, w, h) in faces:
        cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
        face_region = grey[y:y + h, x:x + w]
        brightness = np.average(face_region)

        if key == ord('s'):
            if brightness > 50:
                face_img = cv.resize(face_region, (220, 220))
                img_name = f'face.{id}.{datetime.now().microsecond}.
                   jpeg'
                cv.imwrite(f'faces/{id}/{img_name}', face_img)
                photos_taken += 1
                print(f'{photos_taken} -> Foto salva!')
            else:
                print("Imagem muito escura. Foto nao salva.")
```

```
    cv.imshow('Face', img)

    if key == ord('q'):
        break

camera.release()
cv.destroyAllWindows()
```

A.2 Treinamento do Modelo

Este trecho de código realiza o treinamento do modelo LBPH a partir das imagens capturadas, associando os rostos aos respectivos identificadores numéricos.

```
import os
import pandas as pd
import numpy as np
import cv2 as cv

id_names = pd.read_csv('id-names.csv')
id_names = id_names[['id', 'name']]

lbph = cv.face.LBPHFaceRecognizer_create(threshold=50)

def create_train():
    faces = []
    labels = []
    for id in os.listdir('faces'):
        path = os.path.join('faces', id)
        try:
            os.listdir(path)
        except:
            continue
        for img in os.listdir(path):
            try:
                face = cv.imread(os.path.join(path, img))
                face = cv.cvtColor(face, cv.COLOR_BGR2GRAY)

                faces.append(face)
                labels.append(int(id))
            except:
                pass
    return np.array(faces), np.array(labels)

faces, labels = create_train()

print('Treinamento iniciado.')
lbph.train(faces, labels)
```

```
lbph.save('Classifiers/TrainedLBPH.yml')
print('Treinamento finalizado!')
```

A.3 Reconhecimento Facial

O código abaixo realiza o reconhecimento facial em tempo real, detectando rostos com Haar Cascade e classificando-os com base no modelo LBPH treinado.

```
import pandas as pd
import numpy as np
import cv2 as cv

id_names = pd.read_csv('id-names.csv')
id_names = id_names[['id', 'name']]

faceClassifier = cv.CascadeClassifier('Classifiers/haarface.xml')

lbph = cv.face.LBPHFaceRecognizer_create(threshold=500)
lbph.read('Classifiers/TrainedLBPH.yml')

camera = cv.VideoCapture(0)

while cv.waitKey(1) & 0xFF != ord('q'):
    _, img = camera.read()
    grey = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    faces = faceClassifier.detectMultiScale(grey, scaleFactor=1.1,
       minNeighbors=4)

    for x, y, w, h in faces:
        faceRegion = grey[y:y + h, x:x + w]
        faceRegion = cv.resize(faceRegion, (220, 220))

        label, trust = lbph.predict(faceRegion)

        match = id_names[id_names['id'] == label]

        if not match.empty:
            name = match['name'].item()
        else:
            name = "Rosto desconhecido"

        cv.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
        cv.putText(img, name, (x, y + h + 30), cv.
           FONT_HERSHEY_COMPLEX, 1, (0, 0, 255))

    cv.imshow('Recognize', img)
```

```
camera.release()
cv.destroyAllWindows()
```

A.4 Avaliação de Desempenho

O código a seguir realiza a avaliação quantitativa do modelo treinado com LBPH. Ele compara as predições do modelo com os valores reais em duas bases de teste (rostos conhecidos e desconhecidos), gera a acurácia total e por tipo, calcula falsos positivos e negativos, e exibe a matriz de confusão e o relatório de classificação.

```
import os
import cv2 as cv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, confusion_matrix,
   classification_report

threshold = 50
model_path = 'Classifiers/TrainedLBPH.yml'
unknown_label = -1

id_names = pd.read_csv('id-names.csv')[['id', 'name']]

lbph = cv.face.LBPHFaceRecognizer_create()
lbph.read(model_path)

bases = {
    "faces_acuracia": "conhecido",
    "faces_desconhecidos": "desconhecido"
}

true_labels = []
predicted_labels = []
origem_base = []

for base_dir, tipo in bases.items():
    for item in os.listdir(base_dir):
        item_path = os.path.join(base_dir, item)

        if os.path.isfile(item_path):
            img = cv.imread(item_path, cv.IMREAD_GRAYSCALE)
            if img is None:
                print(f"[ERRO] Nao foi possivel ler: {item_path}")
                continue
```

```
            img_resized = cv.resize(img, (220, 220))
            predicted_label, trust = lbph.predict(img_resized)
            if trust > threshold:
                predicted_label = unknown_label

            true_label = unknown_label
            true_labels.append(true_label)
            predicted_labels.append(predicted_label)
            origem_base.append(tipo)

            print(f"[INFO] {tipo.upper()} | Real: {true_label} |
               Previsto: {predicted_label} | Trust: {trust:.2f} | {
               item_path}")

        elif os.path.isdir(item_path):
            for img_name in os.listdir(item_path):
                img_path = os.path.join(item_path, img_name)
                img = cv.imread(img_path, cv.IMREAD_GRAYSCALE)
                if img is None:
                    print(f"[ERRO] Nao foi possivel ler: {img_path}"
                       )
                    continue

                img_resized = cv.resize(img, (220, 220))
                predicted_label, trust = lbph.predict(img_resized)
                if trust > threshold:
                    predicted_label = unknown_label

                if tipo == "desconhecido":
                    true_label = unknown_label
                else:
                    try:
                        true_label = int(item)
                    except ValueError:
                        print(f"[ERRO] Pasta '{item}' nao e um ID
                           valido.")
                        continue

                true_labels.append(true_label)
                predicted_labels.append(predicted_label)
                origem_base.append(tipo)

                print(f"[INFO] {tipo.upper()} | Real: {true_label} |
                    Previsto: {predicted_label} | Trust: {trust:.2f}
                    | {img_path}")

dados = pd.DataFrame({
```

```
    'real': true_labels,
    'previsto': predicted_labels,
    'tipo': origem_base
})

acuracia_total = accuracy_score(dados['real'], dados['previsto'])
print("\nAcuracia total: {:.2f}%".format(acuracia_total * 100))

for tipo in dados['tipo'].unique():
    subset = dados[dados['tipo'] == tipo]
    acuracia_tipo = accuracy_score(subset['real'], subset['previsto'
       ])
    print(f"Acuracia ({tipo}): {acuracia_tipo * 100:.2f}%")

falsos_positivos = dados[(dados['tipo'] == 'desconhecido') & (dados[
   'previsto'] != unknown_label)]
falsos_negativos = dados[(dados['tipo'] == 'conhecido') & (dados['
   previsto'] == unknown_label)]

print(f"\nFalsos positivos: {len(falsos_positivos)}")
print(f"Falsos negativos: {len(falsos_negativos)}")

rotulos = sorted(list(set(true_labels + predicted_labels)))
rotulo_nomes = []

for r in rotulos:
    if r == unknown_label:
        rotulo_nomes.append("Desconhecido")
    else:
        nome = id_names[id_names['id'] == r]['name'].values
        rotulo_nomes.append(nome[0] if len(nome) > 0 else f"ID {r}")

cm = confusion_matrix(true_labels, predicted_labels)
print("\nMatriz de Confusao:\n", cm)

print("\nRelatorio de Classificacao:\n", classification_report(
   true_labels, predicted_labels, target_names=rotulo_nomes))

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=
   rotulo_nomes, yticklabels=rotulo_nomes)
plt.xlabel('Previsto')
plt.ylabel('Real')
plt.title('Matriz de Confusao - Modelo LBPH')
plt.tight_layout()
plt.show()
```