UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LUIZ GUILHERME T. RIBEIRO

SISTEMA DE PRESENÇA POR RECONHECIMENTO FACIAL

RIO DE JANEIRO
2022

LUIZ GUILHERME T. RIBEIRO

SISTEMA DE PRESENÇA POR RECONHECIMENTO FACIAL

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Paulo Henrique de Aguiar Rodrigues

RIO DE JANEIRO

2022

CIP - Catalogação na Publicação

Ribeiro, Luiz Guilherme Trindade
RR484s Sistema de presença por reconhecimento facial / Luiz Guilherme Trindade Ribeiro. -- Rio de Janeiro, 2022.
56 f.

Orientador: Paulo Henrique de Aguiar Rodrigues.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2022.

1. Reconhecimento facial. 2. Processamento de imagem. 3. Chamada. 4. dlib. 5. HOG. I. Rodrigues, Paulo Henrique de Aguiar, orient. II. Título.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a), sob a responsabilidade de Miguel Romeu Amorim Neto - CRB-7/6283.

LUIZ GUILHERME T. RIBEIRO

SISTEMA DE PRESENÇA POR RECONHECIMENTO FACIAL

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 09 de Fevereiro de 2022

BANCA EXAMINADORA:

Paulo Henrique de Aguiar Rodrigues
Professor Titular (UFRJ)

Valeria Menezes Bastos
Professora Adjunta (UFRJ)

Silvana Rossetto
Professora Associada (UFRJ)

Dedico este trabalho a todas as pessoas com as quais tive a oportunidade de aprender alguma coisa ao longo da minha vida.

AGRADECIMENTOS

Gostaria de agradecer aos meus pais, José L. R. Filho e Maria T. S. Trindade, por todo amor, dedicação e carinho ao longo de todos os meus anos de vida.

Agradeço a todos os professores com os quais tive o prazer de aprender em toda minha jornada até aqui, pois sem o conhecimento que me foi passado não seria quem hoje sou.

Por fim, mas não menos importante, agradeço a minha esposa Marianna Laviano e a minha Filha Olívia L. Ribeiro pela compreensão e apoio para que o trabalho aqui apresentado pudesse ser desenvolvido.

"Machines take me by surprise with great frequency."

Alan Turing

RESUMO

Sistemas de visão computacional estão se tornando parte do cotidiano das pessoas, trazendo na grande maioria das vezes facilidade e praticidade para suas vidas. Com o avanço de tais tecnologias, sua utilização para a autenticação biometrica através do reconhecimento facial se tornou comumente utilizada em diferentes cenários com finalidades distintas. O objetivo deste trabalho é aprofundar-se nas técnicas e ferramentas utilizadas no reconhecimento facial através do desenvolvimento de um sistema de presenças baseado em reconhecimento facial. Para isso, a biblioteca Python utilizada como base no estudo aqui apresentado foi a *Facial Recognition*, que por sua vez utiliza a dlib, uma biblioteca com foco em *machine learning* e métodos numéricos, reconhecida por apresentar 99,38% de acurácia no em testes de reconhecimento facial utilizando o conjunto de registros presentes no *labaled faces in the wild*, disponibilizado pelo MIT com esta finalidade. O sistema aqui desenvolvido utiliza-se da arquitetura cliente servidor e, apesar funcional, apresenta problemas de desempenho, sobretudo quanto ao processamento de frames.

Palavras-chave: visão computacional; reconhecimento facial; processamento de imagem; chamada;

ABSTRACT

Computer vision systems became part of people's life, bringing ease and practicality to them most of the time. With the enhancement of such technologies, their use for biometric authentication through facial recognition has become commonly used in different scenarios with multiple purposes. The main goal of this work is to delve into the techniques and tools used in facial recognition by implementing and analysing a classroom attendance system based it. To fulfill this purpose the Facial Recognition Python package was used and since it depends on the dlib library, a machine learning and numerical methods toolkit that is recognized for 99.38% accuracy on tests with the labelled faces in the wild dataset provided by MIT. The system developed on this project is implemented using the client server architecture and, even though it is functional, it has performance problems, regarding mainly the frame processing part of it.

Keywords: computer vision; facial recognition; image processing; attendance;

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BCC	Bacharelado em Ciência da Computação
CSS	*Cascade style sheet*
CV	*Computer vision*
FPS	*Frames per second*
GB	Giga bytes
GHz	Giga Hertz
HOG	*Histogram of oriented gradients*
HTML	*Hyper text markup language*
HTTP	*Hyper Text Transfer Protocol*
JSON	*JavaScript object notation*
LTS	*Long term service*
MHz	Mega Hertz
MIT	*Massachuttes Institute of Technology*
MVP	*Minimun viable product*
NoSQL	*No structured query language*
RAM	*Random access memory*
REST	*Representational state transfer*
RGB	*Red, Green and Blue*
SSD	*Solid state drive*
SSL	*Secure socket layer*
SVM	*Support vector machine*
TCP	*Transmission control protocol*
TLS	*Transport layer security*

LISTA DE SÍMBOLOS

θ	Letra grega theta
∇	Letra grega nabla (gradiente)
$\sum$	Somatório
$\forall$	Para todo

SUMÁRIO

1 INTRODUÇÃO

O desejo de criar máquinas com capacidade de imitar comportamentos e habilidades humanas é quase tão antigo quanto a própria computação. A questão levantada por Turing (1950, p. 433, tradução) "Podem máquinas pensar?", abre espaço para uma vasta vertente de discussão acerca de quais qualidades, características e comportamentos são de fato exclusivamente humanos, e quais podem ser replicados por máquinas. Os seres humanos percebem e se relacionam com o meio em que se encontram a partir de seus sentidos e são capazes de interpretar os estímulos capitados por diferentes órgãos sensoriais a fim de criar uma representação fidedigna do ambiente em que estão, para que seja possível conviver e interagir com ele e todos os elementos que fazem parte dele, sejam esses outros seres ou objetos. Um dos principais sentidos utilizados para isso é a visão, que é capaz de criar uma imagem do ambiente a partir de estímulos luminosos.

A proposta de criar máquinas dotadas do sentido da visão teve sua primeira mostra em 1963, com um artigo de Larry Roberts sobre a extração de características de três dimensões a partir de projeções em duas dimensões de objetos. Com o aumento da capacidade de processamento dos computadores desde então, disseminou-se e evoluiu consideravelmente até chegar nos dias atuais, nos quais câmeras fotográficas e filmadoras são capazes de focalizar objetos específicos, sistemas de estacionamento são capazes de capturar e identificar placas de automóveis e aparelhos celulares autorizam ou não o acesso de usuários com base em seus rostos.

O desenvolvimento e aprimoramento da visão computacional tornou possível a resolução de diversos problemas de computação aplicada como os mencionados anteriormente e outros que permeiam outras áreas como biologia, astrologia e física. Em linhas gerais, a visão computacional consegue hoje tanto solucionar questões nas quais tradicionalmente seriam utilizadas pessoas quanto tarefas humanamente impossíveis, dadas as limitações de nosso aparato visual.

De tal forma, este trabalho de conclusão de curso foi desenvolvido com o objetivo principal de mergulhar a fundo na visão computacional, com foco específico em técnicas de reconhecimento facial. Assim, os conhecimentos obtidos durante o Bacharelado em Ciência da Computação nas áreas de arquitetura de sistemas distribuídos, redes de computadores, infraestrutura, aprendizado por máquina e bancos de dados, além de revisitados, foram aprofundados e colocados em prática.

Diversas são as aplicações em que o reconhecimento facial é utilizado na atualidade. Soluções como a prova de vida realizada por pensionistas do INSS, a prevenção a fraudes no transporte público em alguns estados brasileiros e a autenticação e desbloqueio em *smartphones* são alguns dos exemplos de soluções já disponíveis que utilizam tal tecnologia.

A área escolhida para aplicação de tais técnicas neste trabalho foi a da educação. Nela, inúmeros desafios poderiam utilizar-se da visão computacional como recurso para resolução de problemas. A utilização de sistemas capazes de metrificar experiências de aprendizagem para realizar um acompanhamento detalhado dos estudantes por parte dos docentes e intervenção e tomada de decisão a partir de notificações apresentam um grande potencial.

O problema atacado neste trabalho foi o de atribuição de presenças e faltas em sala de aula. Ele foi escolhido pois o autor, enquanto lecionava, identificou que as alternativas convencionais consumiam um tempo considerável, em alguns casos eram pouco confiáveis e, além disso, pouco práticas em termos de análise de dados para tomada de decisão. Com base nisso, uma solução foi desenhada com a capacidade de aferir presenças de forma automática e passiva, para que o processo fosse realizado sem interferência humana e, ao mesmo tempo, os dados pudessem ser utilizados de forma fácil posteriormente pelo docente. O projeto passou por diversas etapas para que pudesse ser realizado com sucesso. A primeira foi a parte de pesquisa, abordada no Capítulo 2, na qual as diferentes alternativas tanto da parte algorítmica quando de bibliotecas já existentes para reconhecimento facial foram estudadas e analisadas cuidadosamente. A etapa seguinte, apresentada no Capítulo 3, foi a de concepção do projeto, na qual a arquitetura da aplicação foi esboçada, as entidades do banco de dados definidas e as escolhas de protocolos, linguagens, bibliotecas, padrões de projeto, banco de dados e infraestrutura realizadas. Em sequência deu-se a implementação do projeto de fato, na qual os conhecimentos obtidos nas etapas anteriores e requisitos especificados foram utilizados e redesenhados quando necessário, presente no Capítulo 4. Logo após isso, foi realizada a coleta de métricas de desempenho do sistema e análise dos dados gerados a partir da execução do sistema desenvolvido operando em diferentes configurações, realizada no Capítulo 5. No Capítulo 6, o último do trabalho aqui apresentado, consta uma análise geral de todo o processo que ocorreu para sua realização e propostas de pontos de melhoria são apresentadas para que além de um MVP a solução desenvolvida possa ser transformada em um produto de aplicação em ambientes de ensino.

2 CONCEITOS BÁSICOS

Este capítulo trará uma visão geral quanto a visão computacional, com enfoque principalmente em reconhecimento facial. Trataremos portanto de esmiuçar os principais conceitos por trás da visão computacional, como a detecção e identificação de objetos e as especificidades relacionadas ao reconhecimento de pessoas a partir de imagens de seus rostos. As técnicas e métodos apresentados dizem respeito às empregadas pelas bibliotecas *Face Recognition*[1] e *dlib*[2], uma vez que estas foram as utilizadas no projeto pelo autor, por apresentarem vasta documentação e informações disponíveis e serem facilmente utilizadas. Além destas, a biblioteca *face-api*[3] também foi inicialmente avaliada mas, como diversos problemas surgiram em sua execução em casos base e exemplos, optou-se por descartá-la e focar nas duas outras supracitadas.

2.1 VISÃO COMPUTACIONAL

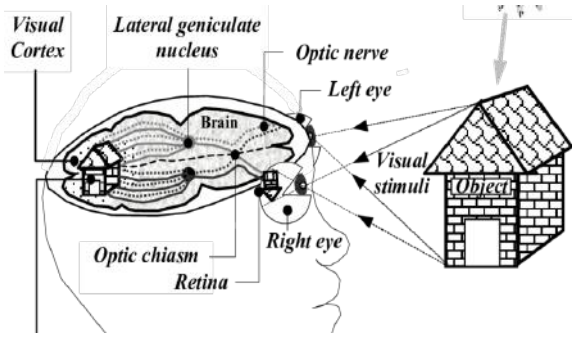
A visão é caracterizada como o processo pelo qual um ser é capaz de receber estímulos luminosos do ambiente em que se encontra e, a partir disso, formar uma representação de um local, tornando possível assim o reconhecimento de objetos e outros seres presentes no cenário. Diferentes espécies foram selecionados pelo meio em que vivem ao longo do tempo por vantagens diversas e, como consequência deste fato, utilizam mecanismos distintos para enxergar, variando no grau de complexidade e detalhamento das imagens criadas no sistema nervoso central dos seres. Como humanos, a grande maioria de nossa espécie é capaz de enxergar e percebe o mundo a partir de imagens. É válido ressaltar, no entanto, que apesar de veloz, o procedimento realizado pelo corpo humano para que seja possível enxergar é de grande complexidade.

O sistema visual é constituído em linhas gerais pelos olhos, pelos caminhos e ligações entre os olhos e o cérebro e partes mais específicas do cérebro encarregadas do processamento da imagem observada como o córtex visual(TALUKDAR; GANGULY, 2017). A Figura 1 representa este processo, que se inicia com a luz passando pela córnea, que é a membrana mais externa do olho, e entrando pela iris, a responsável por regular a quantidade de luz que entra na cavidade ocular. A lente faz com que a imagem capturada seja devidamente focada e assim, um retrato invertido é formado na retina, onde células especializadas (cones e bastonetes) serão responsáveis por, através do nervo ótico, estimular o córtex visual e esse, por sua vez, realiza o processamento da imagem, juntando as informações do ambiente coletadas pelos dois olhos e estimula outras áreas do cérebro

[1] https://github.com/ageitgey/face$_r ecognition$
[2] http://dlib.net/
[3] https://justadudewhohacks.github.io/face-api.js/docs/index.html

para que seja realizada a detecção e identificação de objetos e pessoas (TALUKDAR; GANGULY, 2017).

Figura 1 – Processo de visão humana



Fonte: https://www.researchgate.net/figure/Human-vision-process_fig2_247834199/

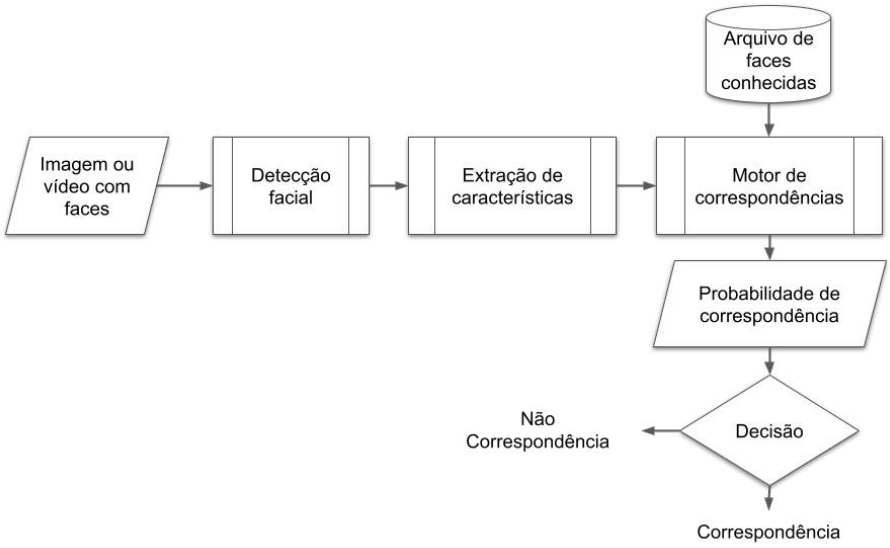
A visão computacional (CV) é um campo da computação que teve início em 1960 com a tese de Ph.D. de Larry Roberts no MIT, que tratava da extração de informações geométricas em três dimensões a partir de vistas perspectivas em duas dimensões de poliedros(ALOIMONOS, 1992). Ao longo do tempo, técnicas de base para aprimoramento da visão computacional como detecção de bordas e fronteiras e segmentação de imagens foram sendo desenvolvidas (HUANG, 1996). Tais técnicas fizeram com que aplicações como direção autônoma, montagem de equipamentos e componetes eletrônicos e inspeções industriais pudessem ser possíveis (HUANG, 1996). Na atualidade, no entanto, a forma como a visão computacional é empregada nessas aplicações difere bastante do modelo de visão presente nos seres humanos. O paradigma chamado *Purposive Vision* defende que os algoritmos de visão computacional devem se orientar por seus objetivos e que na maioria dos casos devem ser qualitativos (ROBERTS, 1965). Por conta disso, apesar das técnicas demostradas neste capítulo serem aplicáveis em cenários diferentes do abordado, os modelos utilizados foram treinados com um propósito específico.

2.2 SISTEMA DE RECONHECIMENTO FACIAL

Um sistema de reconhecimento facial é uma aplicação que utiliza computadores para verificar e identificar pessoas a partir de características extraídas de imagens digitais, comparando-as com características obtidas anteriormente e adicionadas em um registro como banco de dados (SHARMA; SHANMUGASUNDARAM; RAMASAMY, 2016). O objetivo principal de tais aplicações é fornecer uma forma de validação ou verificação biométrica de maneira não intrusiva, sendo possível realizá-la mesmo sem ciência dos usuários. No entanto, apesar da tarefa de reconhecimento facial ser trivial para seres humanos, computacionalmente a atividade é extremamente complexa e envolve diversas etapas a fim de ser realizada com acurácia e precisão. Grande parte da dificuldade está

associada a elementos variáveis na imagem como a distância que rosto a ser reconhecido se encontra da câmera, a resolução da câmera utilizada, o posicionamento do rosto, a luz ambiente e expressões faciais e, além disso, partes da face podem ser sombreadas pelo cabelo ou barba e até mesmo maquiagem (RAWLINSON; BHALERAO; WANG, 2010). O fluxograma presente na Figura 2 a seguir ilustra as etapas do processo de reconhecimento facial.

Figura 2 – Etapas do reconhecimento facial



Fonte: Adaptado de Rawlinson, Tim (2010 p.4)

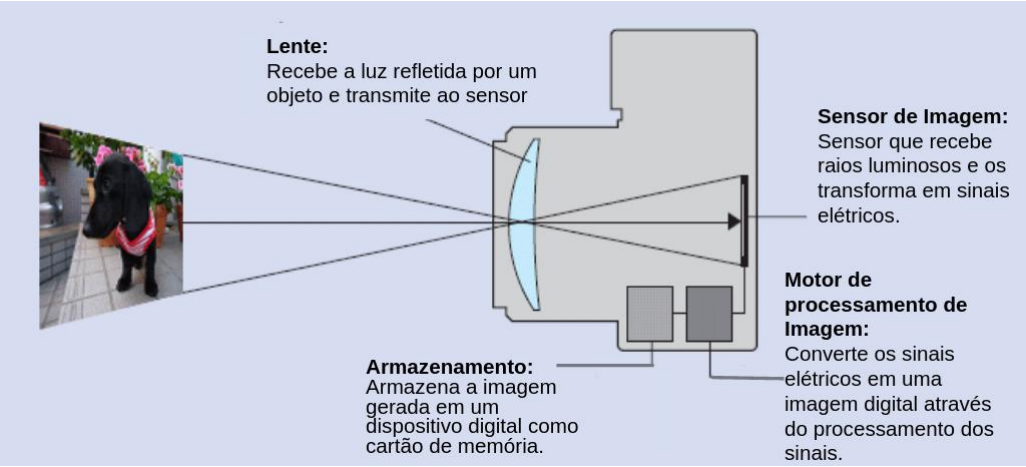
O reconhecimento facial faz parte de uma classe de problemas conhecido como *pattern recognition* e, como é possível verificar na Figura 2, passa por etapas cruciais de localização da face, normalização e extração de características faciais, que devem aparecer em todas as faces e, ao mesmo tempo, serem únicas para comparação com a base de características faciais conhecidas e, assim, identificar uma face presente na imagem (RAWLINSON; BHALERAO; WANG, 2010). Faz-se necessário, portanto descrever e analisar cada uma das etapas do processo, levando em consideração os métodos e algoritmos utilizados especificamente na aplicação desenvolvida.

2.2.1 Captura de imagem

O processo de captura de imagem é o primeiro de todo o fluxo e primordial para o bom desempenho do reconhecimento facial. Tal procedimento pode ser exemplificado no esquema fornecido pela Figura 3. Nele, a câmera digital USB recebe a luz, que é focalizada pela lente e atinge um sensor de silício composto por diversos fotopontos (geralmente denominados de pixels) sensíveis a luz (LODRIGUSS, 2016). A partir deste ponto, o sensor codifica a informação recebida por cada pixel, transformando assim o sinal analógico

em digital, o que resulta um uma imagem. É válido ressaltar que apesar das estruturas utilizadas em cada uma das etapas serem diferentes, o mecanismo pelo qual imagens são obtidas de forma digital se assemelha bastante ao mecanismo já mencionado através do qual seres humanos conseguem enxergar. Além disso, é primordial estabelecer a relação entre uma única imagem obtida a partir de uma câmera e um vídeo. O vídeo se caracteriza por uma série de imagens obtidas ao longo de um intervalo de tempo em sequência cronológica. Na grande maioria das aplicações comerciais a quantidade de imagens obtidas em um intervalo de tempo é caracterizada por FPS *(frames por segundo)*.

Figura 3 – Funcionamento da câmera digital



Fonte: https://av.jpn.support.panasonic.com/support/global/cs/dsc/knowhow/knowhow01.html (modificada)

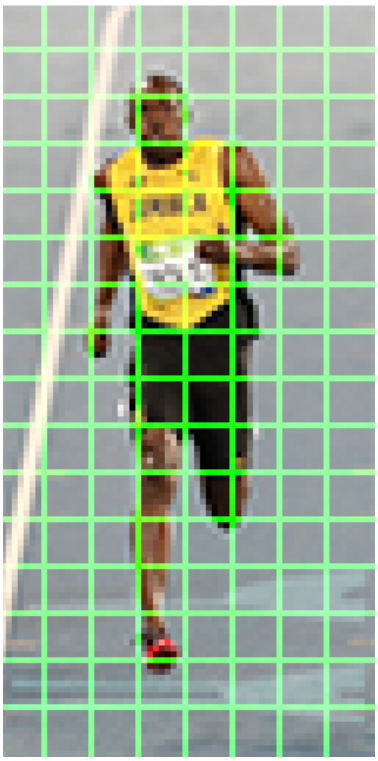
2.2.2 Detecção facial e extração de faces

Uma vez a imagem digitalizada, o próximo passo no processo consiste na detecção e extração dos rostos presentes na amostra (RAWLINSON; BHALERAO; WANG, 2010). No entanto, antes de realizar a extração das faces, com a finalidade de reduzir a quantidade de dados a serem processados, a técnica de conversão de cores, de RGB para tonalidades de cinza é feita. Assim, os dados de coloração da imagem, que são irrelevantes para a detecção facial são descartados e o tempo de processamento da imagem é otimizado (GEITGEY, 2016). Antes da detecção e extração dos rostos presentes na imagem, é preciso comprimir a quantidade informações presentes nela e extrair somente as *features* (características) necessárias para que as faces sejam encontradas. O algoritmo utilizado para tal tarefa é o histograma de gradientes ordenados (HOG). Tal algorimo se destaca por exigir pouco poder de processamento e superar os outros métodos existentes (DALAL; BILL, 2005). A entrada desse método consiste em uma imagem em tons de cinza e sua saída é um vetor de características da imagem fornecida. O intuito desta técnica é, através da variação de intensidade entre pixels vizinhos, fornecer a magnitude e a orientação da variação para

cada região da imagem processada. Levando em consideração que em uma imagem a maior variação de intensidade tende a ocorrer nas bordas dos objetos, o HOG é capaz de indicar fielmente em grande parte dos casos a fronteira entre dois elementos da imagem de forma veloz e sem a necessidade da utilização de poderosos recursos computacionais (MALLIK, 2016).

O primeiro passo do algoritmo consiste nas operações de recorte e escala. Nele, a imagem fornecida para análise é segmentada em diversas porções de acordo com seu tamanho, podendo haver sobreposição entre elas, mas sempre mantendo a relação de 1:2. A partir de um recorte obtido a operação de escala é realizada, transformando-o em uma imagem de 64x128. Após isso o recorte transformado será dividido em porções de 8x8, que serão utilizadas na computação do histograma de gradientes orientados (MALLIK, 2016), como pode ser verificado na Figura 4.

Figura 4 – Imagem dividida em 8x8 pixels

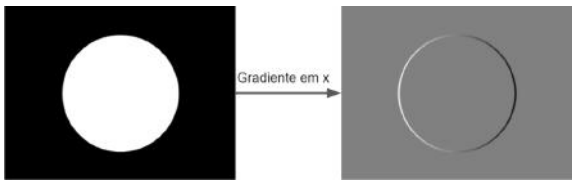


Fonte: https://www.learnopencv.com/histogram-of-oriented-gradients/

De posse dos segmentos de 8x8, como explicitado pelo próprio nome do método, vetores gradientes são extraídos da imagem em relação ao eixo das abscissas (x) e das ordenadas (y). As Figuras 5 e 6 demonstram graficamente o aspecto do gradiente obtido com relação ao eixo x (g_x) e ao eixo y (g_y), respectivamente, de uma imagem constituída de um círculo branco e um fundo preto (imagem à esquerda).

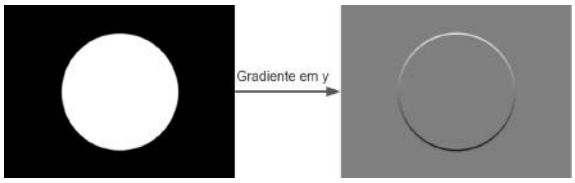
Esse gradiente é calculado através da diferença entre dois pixels adjacentes quanto a suas intensidades, onde zero representa o preto e 1 representa a cor branca. Esse processo

Figura 5 – Gradiente em X (g_x)



Fonte: https://stackoverflow.com/questions/19815732/what-is-the-gradient-orientation-and-gradient-magnitude, modificada

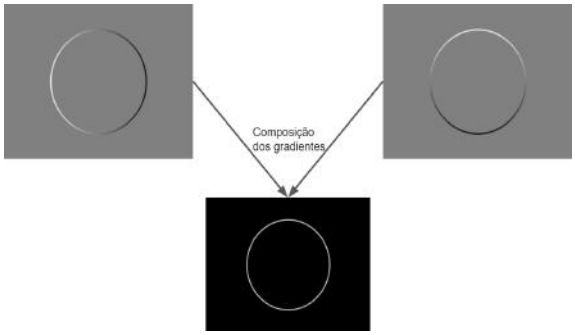
Figura 6 – Gradiente em Y (g_y)



Fonte: https://stackoverflow.com/questions/19815732/what-is-the-gradient-orientation-and-gradient-magnitude, modificada

é feito ao longo de todo o eixo para cada par de pixels da esquerda para a direita em x e de cima para baixo em y. Como resultado disso, é obtida a imagem cinza (à direita) na Figuras 5 e 6. A transição de preto para branco apresenta a cor branca por possuir a maior variação possível e, analogamente, a transição de branco para preto apresenta a cor preta, por apresentar a menor variação. Assim, a imagem apresentada na Figura 7 é o resultado da união da intensidade dos dois gradientes gerados. Vale ressaltar que os valores possíveis para g_x e g_y encontram-se no intervalo de -1 à 1.

Figura 7 – União dos gradientes referentes a X e Y



Fonte: https://stackoverflow.com/questions/19815732/what-is-the-gradient-orientation-and-gradient-magnitude

Com os gradientes calculados em x (g_x) e y (g_y), são obtidas a direção (equação 2.1)

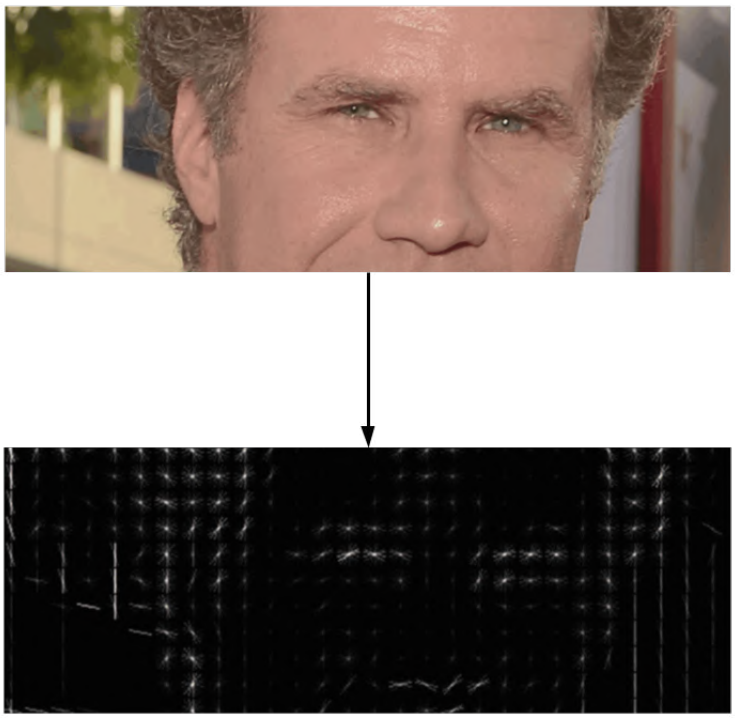
e a intensidade (equação 2.2) associada a cada ponto da imagem.

$$\theta = \arctan \frac{g_y}{g_x} \tag{2.1}$$

$$g = \sqrt{g_x^2 + g_y^2} \tag{2.2}$$

A finalidade principal de dividir a figura dessa forma é condensar e compactar as características da imagem. Levando em consideração um recorte de 8x8 da imagem, a intensidade e direção dos gradientes calculados, temos um total de 128 valores (8x8x2). A Figura 8 evidencia os histogramas de gradientes orientados gerados em cada pedaço de 8x8 pixels de uma imagem e possibilita verificar que as informações presentes na imagem são condensadas, sendo claras principalmente as bordas e fronteiras existentes entre os diferentes elementos.

Figura 8 – Visualização da etapa intermediária do HOG

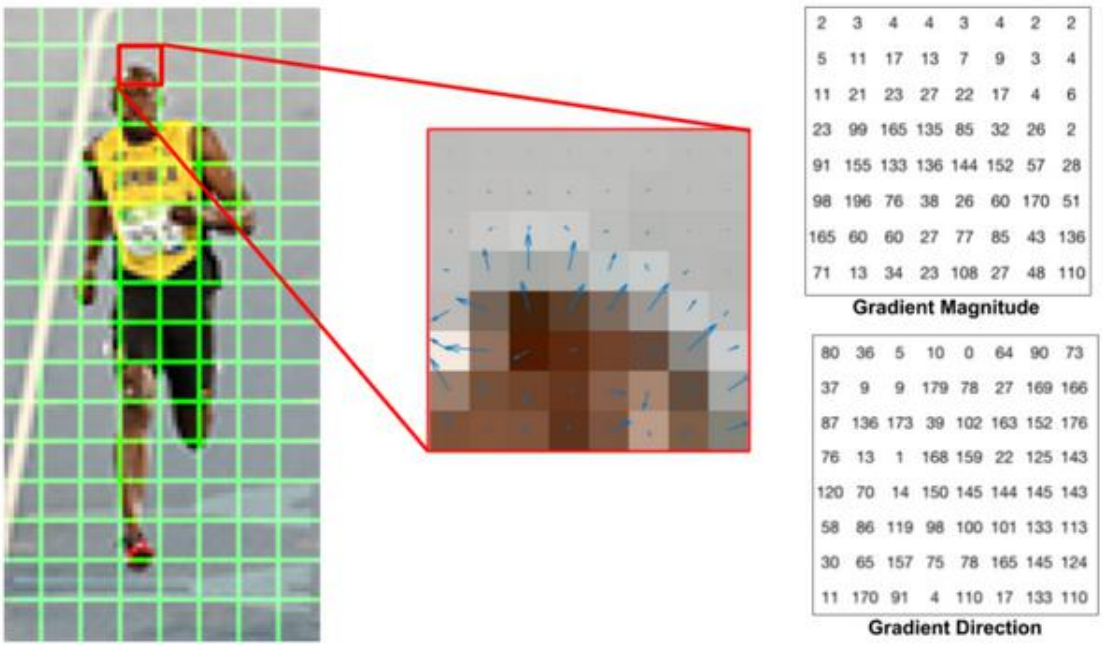


Fonte: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78, modificada

Já a Figura 9 representa visualmente todas as informações contidas na secção da imagem, os pixels, a magnitude (tamanho do vetor) e direção (angulação) do gradiente. Ao final do método esses 128 valores são resumidos em apenas 9 e, além disso, condensando a informação utilizando pedaços da imagem maiores do que pixels, a técnica se torna menos sensível a ruídos (MALLIK, 2016). Ainda na Figura 9 é possivel observar que a direção

dos gradientes varia apenas de 0^o à 180^o. Isso ocorre pois, empiricamente, verificou-se um desempenho melhor do HOG ao utilizar o valor absoluto dos gradientes (sem sinal) (MALLIK, 2016).

Figura 9 – Intensidade e direção de gradientes de porção 8x8



Fonte: https://www.learnopencv.com/histogram-of-oriented-gradients/

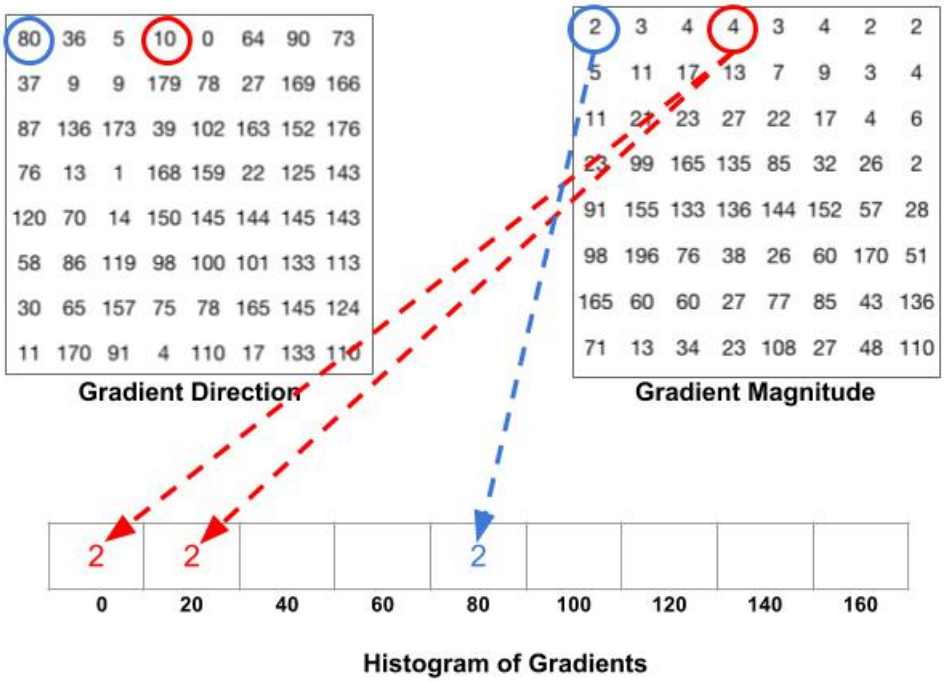
Com a magnitude e direção do gradiente associadas a cada ponto obtida, o próximo passo consiste na elaboração do histograma. Para isso, são criados 9 containers diferentes que são etiquetados de 0 à 160, em intervalos de 20. Esses containers armazenam a soma da intensidade dos gradientes presentes no pedaço de imagem analisado, utilizando para isso a direção dos mesmos. Dessa forma, as magnitudes são proporcionamente divididas e acrescentadas aos containers com etiquetas que descrevem o intervalo no qual a direção do gradiente se encontra, utilizando para isso a equação 2.3 para o cálculo do percentual do valor da magnitude acrescido ao limite inferior e a equação 2.4 para o cálculo do percentual do valor da magnitude acrescido ao limite superior. A Figura 10 ilustra como tal procedimento é realizado. Nela é possível verificar que no caso em que a magnitude apresenta valor 4 para um gradiente de direção 10 a magnitude é igualmente dividida entre os containers 0 e 20 (50% para cada). Isso ocorre pois 10 encontra-se equidistante de 0 e 20, os containers etiquetados com valores limites do intervalo e, segundo o resultado obtido ao aplicar-se 2.3 e 2.4 4 deve acrescentar 50% ao container 0 e 50% ao container 20. Outro caso que se destaca nesta figura é o marcado em azul. Como a direção do gradiente é igual a etiqueta de um container a magnitude é acrescida em sua totalidade

(100%) a ele.

$$C_{low} += \frac{20 - \nabla_{dir_{i,j}}}{20} \tag{2.3}$$

$$C_{high} += 1 - C_{low} \tag{2.4}$$

Figura 10 – Criação do histograma a partir das magnitudes e direções dos gradientes



Fonte: https://www.learnopencv.com/histogram-of-oriented-gradients/

O último passo do algoritmo consiste em normalizar os histogramas a fim de diminuir o impacto das variações de iluminação, que interferem diretamente na magnitude do gradiente, para que o vetor de características final possa ser obtido (MALLIK, 2016). A normalização é feita utilizando porções de 16x16 da imagem ou seja, com um histograma sendo gerado por um bloco de 8x8, na normalização passamos a analisar quatro histogramas ao mesmo tempo. Nesse processo todos os valores dos histogramas são divididos pelo fator apresentado na equação 2.5. Dessa forma, o bloco de 16x16 da imagem é transformado em um vetor de características que contém 36 elementos (4 histogramas com 9 containers) normalizados com diferenças de iluminação e ruídos removidos.

$$V_{norm} = \sqrt{\left(\sum_{\forall} m_0\right)^2 + \left(\sum_{\forall} m_{20}\right)^2 + ... + \left(\sum_{\forall} m_{160}\right)^2} \tag{2.5}$$

O procedimento de normalização é realizado para todos os blocos de 16x16 presentes na imagem, considerando mesmo os que se sobrepõe, e estes são adicionados ao vetor de características final que será utilizado nas etapas subsequentes do reconhecimento facial. Após todo esse processo de transformação, a entrada deixa de ser uma imagem e é criado o vetor de características, no qual todos os blocos de 8x8 pixels adjacentes na imagem passam a ser presentados por um histograma de gradientes orientados, normalizado com base em sua vizinhança 16x16.

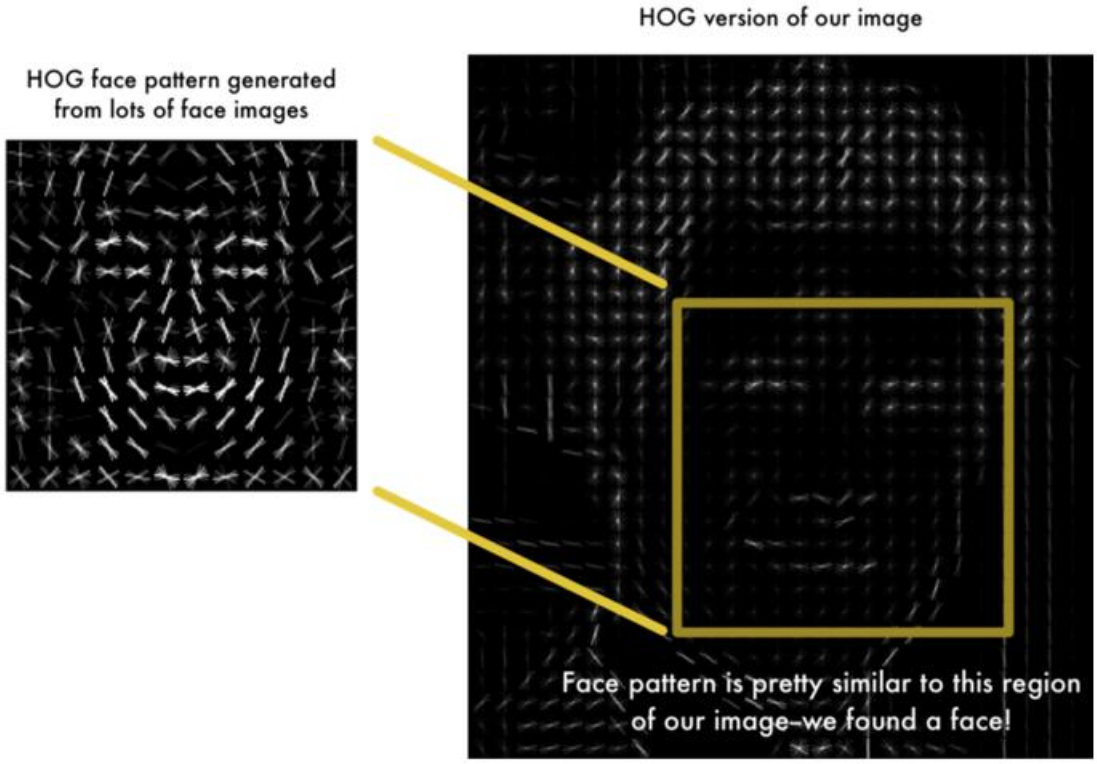
Com a imagem transformada no vetor de características, o próximo passo consiste na identificação extração dos rostos presentes na cena. Essa tarefa é realizada por uma SVM (*Support Vector Machine*) disponibilizada pela biblioteca dlib[4], previamente treinada, que utiliza para isso como entrada diversas faces codificadas presentes em Open faces (AMOS; LUDWICZUK; SATYANARAYANAN, 2016) e a partir do padrão gerado consegue distinguir quais porções da imagem possuem de fato faces e quais não possuem. A Figura 11 fornece um paralelo entre a etapa intermediária do HOG e o padrão considerado como uma face pela SVM e o padrão encontrado em uma imagem codificada. Dessa forma, é possível encontrar todos os rostos presentes em uma imagem, extraí-los e utilizá-los nas etapas subsequentes do reconhecimento facial. É válido ressaltar que, como diferentes raças e etnias encontram-se no conjunto de treino da Open faces, os diferentes traços faciais apresentados por grupos populacionais distintos não interferem no reconhecimento facial. Além disso, características como barba, cabelo, óculos e maquiagem não influenciam no processo de reconhecimento.

2.2.3 Extração de características

A extração de características capazes de identificar um rosto apresenta certa dificuldade para máquinas, principalmente quanto à aparência de uma face. Em um primeiro momento, é intuitivo pensar que para a detecção facial faz-se necessário apenas comparar os vetores de características gerados pelo método HOG e, com base em um percentual de semelhança com os rostos já codificados em um banco de dados, reconhecer os indivíduos presentes na imagem. No entanto, pessoas podem aparecer em imagens em diferentes ângulos e poses, o que torna os vetores de características completamente distintos para uma mesma pessoa. A fim de solucionar essa questão, torna-se necessário encontrar para todas as faces detectadas em uma imagem, independentemente de suas angulações, os marcos faciais[5], características faciais únicas que são utilizadas como descritores capazes de discretizar características faciais. O processo envolve a utilização de uma estimativa inicial dos marcos faciais, gerada a partir dos pontos presentes no conjunto de teste (E0 na Figura 12), um conjunto esparso de pixels (exemplificado na Figura 13) e a intensidade de cada um dos pixels (KAZEMI; KTH, 2014). O método consiste na utilização de

[4] http://dlib.net/
[5] facial landmarks

Figura 11 – Paralelo entre padrão de faces e imagem codificada



Fonte: https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78

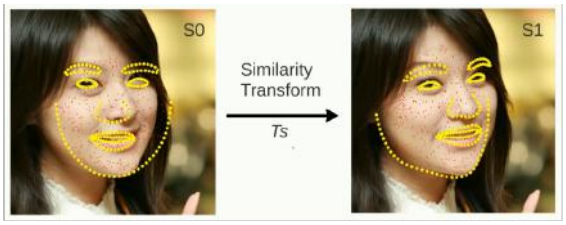
árvores de regressão em série para estimar a posição dos marcos faciais. Com os parâmetros do modelo obtidos através do conjunto de testes (variação dos marcos (Δl), limites de comparação de intensidade de pixels e quais pixels devem ser comparados), cada nó das árvores de regressão compara a intensidade de dois pixels e, a partir dos resultados, comparando-a com o limite de comparação, chega-se a uma folha da árvore. Nessa folha encontram-se as variações de posição que devem ser realizadas em cada um dos marcos faciais. Em cada uma das dez etapas do processo esse mesmo procedimento é realizado para uma série de árvores de regressão e, ao final da rodada, os marcos faciais são atualizados com base na soma de todos os resultados das folhas ($\sum \Delta l$). Com as novas posições dos marcos faciais, o conjunto esparso de pixels passa por uma transformada de similaridade [6]. Nela, a posição do pixel é atualizada com base no marco facial mais próximo dele através de translações e rotações, mantendo assim a mesma distância e ângulo. Esse processo é realizado diversas vezes e desta forma, o método converge (KAZEMI; KTH, 2014) e, assim, obtem-se o vetor de características faciais.

[6] similarity transform

Figura 12 – Estágios da utilização de árvores de regressão em cascata

Fonte: (KAZEMI; KTH, 2014), modificada

Figura 13 – Conjunto esparso de pixel para transformada de similaridade



Fonte: (KAZEMI; KTH, 2014), modificada

2.2.4 Reconhecimento facial

Uma vez as características extraídas da face identificada, a última etapa do processo consiste em reconhecer a pessoa presente na imagem a partir do vetor de características obtido. Para realizar tal tarefa, um classificador deve ser utilizado. Na solução empregada, uma SVM multiclasses é responsável por esta tarefa. A partir do conjunto de faces cadastradas no sistema, a SVM é capaz de classificar o novo vetor de características fornecido como uma pessoa previamente conhecida, minimizando o erro.

3 PROPOSTA DE PROJETO

Esta monografia foi concebida com o intuito de ampliar e sedimentar conhecimentos em diversas áreas da computação com foco principalmente em visão computacional, na qual múltiplas ferramentas e soluções gratuitas e de código aberto encontram-se disponiníveis. Como tal temática não é abordada de forma direta pelas disciplinas ofertadas pelo BCC-UFRJ, este projeto utiliza-se das bases fornecidas ao longo do curso para extrapolá-las e adentrar em uma área específica da computação, que tem se apresentado bastante proeminente e promissora ao longo da última década. A proposta consiste portanto em elaborar e implementar um sistema de presenças baseado em reconhecimento facial, para verificação de comparecimento de estudantes em salas de aula. Tal sistema foi projetado para ser utilizado primariamente em ambientes Unix, levando em consideração que a aplicação deve ser executada pelo docente ou entidade responsável, para que o objetivo principal seja alcançado. A aplicabilidade deste sistema vai ao encontro de uma necessidade observada pelo autor, que em diversas ocasiões enquanto lecionava necessitou de registros históricos de presenças dos alunos e, pela falta de mecanismos e fluxos bem estabelecidos e confiáveis, não obteve dados satisfatórios.

3.1 DETALHAMENTO DA PROPOSTA DE PROJETO

3.1.1 Requisitos

Para que o sistema consiga de fato aferir presenças utilizando reconhecimento facial, é necessário que todos os pontos abaixo sejam implementados:

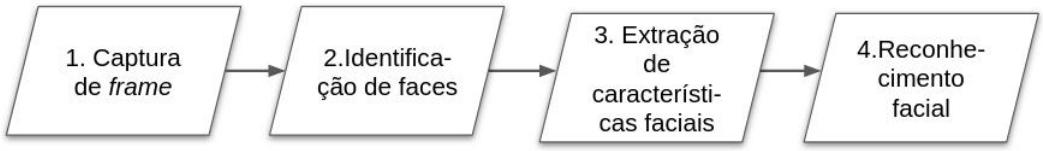
- **Captura de imagem:** O sistema precisa ser capaz de capturar as imagens utilizadas tanto para o cadastro de estudantes (junto com suas informações pessoais), quanto para atribuir presença aos estudantes que se encontrem em um *frame*.
- **Armazenamento de informações**: O sistema precisa ser capaz de armazenar informações de uso geral como nome e matrícula de discentes, assim como seus marcos faciais.
- **Processamento de imagem**: O sistema precisa ser capaz de processar imagens a fim de fornecer como resultado quais as faces presentes no *frame* processado.

3.1.2 Arquitetura lógica

O fluxograma apresentado na Figura 14 exibe as etapas necessárias para que o processo de reconhecimento facial e, consequentemente, a atribuição de prensença a um estudante ocorra. Analogamente, o fluxograma presente na Figura 15 apresenta os passos necessários

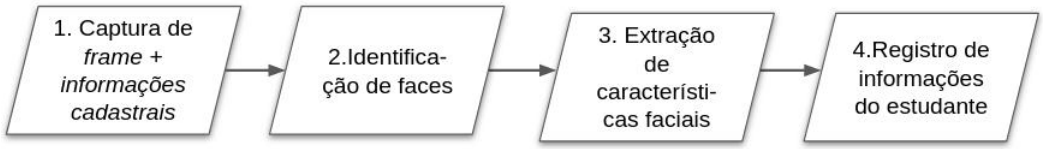
para que o registro de um novo estudante seja realizado junto ao sistema. Inicialmente destaca-se a semelhança existente entre esses dois processos. Em ambos o passo 1 é responsável por gerar os dados utilizados pelo sistema e os demais tem o papel de processar os dados e realizar alterações no estado do aplicação.

Figura 14 – Fluxograma de operações (chamada)



Fonte: Compilação do autor

Figura 15 – Fluxograma de operações (cadastro de estudante)



Fonte: Compilação do autor

Com base nisso, o sistema pode ser dividido em duas frentes diferentes, uma responsável pela geração dos dados e outra responsável por sua utilização e processamento. De tal maneira, um modelo lógico condizente com tais características é o modelo cliente servidor. Nele, o servidor fornece serviços específicos, que na aplicação desenvolvida são os de cadastro de estudantes e reconhecimento facial e, o cliente, fica responsável pela geração dos dados e envio deles ao servidor para que os serviços disponibilizados sejam utilizados. Esse modelo arquitetural apresenta diversas vantagens pois, com a segregação das responsabilidades dessa maneira, o servidor passa a poder atender diversos clientes simultâneamente, os dados são armazenados de forma segregada das partes que utilizam a aplicação, ficando eles no servidor, o que traz maior segurança ao sistema e, além disso, os clientes utilizados podem apresentar poucos recursos computacionais como processamento e memória, uma vez que a parte mais pesada do processo é realizada pelo servidor.

3.1.3 Arquitetura de sistema

Com o sistema dividido em duas frentes, o servidor (*back end*) e o cliente (*front end*), temos que o servidor é a parte responsável por aferir as presenças através do reconhecimento facial, armazenar os dados cadastrados dos alunos para fins de comparação e

validação das presenças, e guardar registros das presenças, levando em consideração as faces detectadas. Os componentes desta parte do sistema são: aplicação contendo as regras de negócio e lógica do sistema, e o sistema de banco de dados. A outra parte constituinte do sistema, o cliente, é responsável por coletar os *frames* a partir da câmera e enviá-los para o servidor para que a imagem possa ser devidamente processada e, a partir disso, as presenças aferidas. Outra atribuição do cliente diz respeito ao registro e cadastro inicial de alunos para que seja possível reconhecer as faces e, posteriormente, atribuir presenças aos estudantes identificados. O componente utilizado como cliente da aplicação nesta implementação é uma página web, acessada através de um navegador. No entanto, dada a arquitetura da aplicação, outras interfaces podem ser desenvolvidas a fim de utilizar a lógica implementada no servidor para reconhecimento facial e atribuição de presenças, uma vez que os mecanismos e protocolos utilizados para a comunicação entre cliente e servidor sejam respeitados.

A comunicação entre cliente e servidor é realizada utilizando dois protocolos distintos. A troca de *frames* entre as duas partes do sistema é feita através de *websockets*, pois esse protocolo mantém a conexão ativa, o que torna o envio de dados do cliente ao servidor e vice-versa mais rápido, uma vez que não é nessário iniciar uma nova comunicação todas as vezes em que ocorre uma troca de mensagem. Já a parte de acesso às páginas do cliente e cadastro de novos estudantes é realizada utilizando o protocolo HTTP, esteja o cliente utilizando TLS ou não. Como abordagem arquitetural utilizada na implementação da rota de cadastro padrão REST é utilizado. Nele, as requisições feitas entre sistemas são realizadas utilizando representações do estado de recursos e, além disso, os diferentes métodos disponibilizados na especificação do protocolo HTTP são utilizados com o intuito de realizar alterações distintas no estado da aplicação. Tal padrão deve ser utilizado pois traz diversos benefícios em termos de estruturação da solução e é amplamente utilizado na indústria.

3.1.4 Fluxo de execução e funcionamento

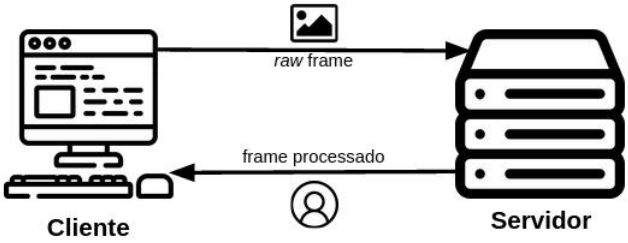
O sistema pode ser executado tanto de forma local, quando o docente executa tanto o cliente quanto o servidor em uma mesma máquina, quanto distribuída, quando um servidor é disponibilizado e docentes podem utilizar clientes para interagir com ele. O fluxo de funcionamento da aplicação, presente na Figura 16, é iniciado com o começo de uma aula e, para isso, é necessário que o servidor esteja em pleno funcionamento. O discente ou entidade responsável deve acessar a interface *web* (1) do programa para que possa ser dado início à chamada. Ao serem concedidas as permissões para a captura de imagens da câmera, o cliente passará a enviar os *frames* capturados para processamento no servidor e o servidor responderá com frames processados (2), que exibem os locais em que rostos foram encontrados e pessoas reconhecidas na imagem, destacando-as visualmente por meio de retângulos envolta dos rostos. Além disso, uma lista contendo matrículas e

nomes dos estudantes que tiveram suas faces identificadas nesta sessão é disponibilizada na página. Caso seja necessário cadastrar estudantes que ainda não tenham seus dados armazenados no sistema, a página dedicada para cadastro de alunos deve ser utilizada. Nela, um formulário é exibido, no qual a matrícula, nome completo, endereço de e-mail devem ser fornecidos e um frame, contendo apenas o rosto que será cadastrado, deve ser capturado para envio ao servidor. Dessa forma o rosto é processado e suas informações são armazenadas no banco de dados junto com as demais preenchidas no formulário e, assim, tais informações podem ser utilizadas novamente na página de chamada.

Figura 16 – Fluxo de funcionamento da aplicação

Fonte: Compilação do autor

Figura 17 – Modelo de arquitetura cliente servidor



Fonte: Compilação do autor

O modelo de funcionamento do sistema é observável na Figura 17. Nela fica evidente as diferentes atribuições de cada uma das partes, com o cliente sendo responsável por

capturar frames com as imagens dos alunos e enviá-las para o servidor e este, por sua vez, realizar todo o processo de transformação da imagem em um vetor de caraterísticas que será comparado aos valores armazenados no banco de dados da aplicação, e com base nisso aferir as presenças.

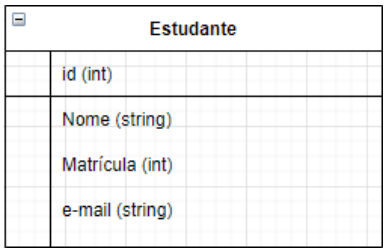
É importante ressaltar que por conta desse fluxo de execução e funcionamento da aplicação quaisquer *frames* capturados são enviados ao servidor para processamento. Dessa forma, mesmo *frames* que não contenham nenhum rosto são processados. Apesar de ser uma maneira de operar ineficiente, tal característica é inerente ao problema proposto uma vez que só é possível reconhecer faces em um *frame* que tenha sido processado. Isso significa que mesmo no caso em que não existem faces para serem reconhecidas é preciso que o *frame* produzido pelo cliente seja processado e analisado pelo servidor da aplicação.

3.1.5 Detalhes de implementação do servidor

O servidor da aplicação é a parte responsável tanto pelo reconhecimento facial quanto pelo gerenciamento dos dados utilizados pelo sistema para aferir presenças. Com a finalidade de disponibilizar o serviço de cadastro de alunos na aplicação, faz-se necessária a implementação de uma interface na qual dados acerca do estudante que será cadastrado no sistema possam ser fornecidos. Para tal, o protocolo HTTP é utilizado, seguindo o padrão de projeto REST. Esta interface é dotada da capacidade de cadastrar os dados fornecidos no sistema de banco de dados utilizado e, quando necessário, retorná-los para que possam ser utilizados no reconhecimento facial. Outra interface disponibilizada pelo servidor compreende o processamento de imagens fornecidas pela aplicação cliente. Dada a natureza do problema solucionado pela aplicação, não é necessária a utilização de mecanismos com alto grau de resolução como *data streams*, na qual *frames* em sequência são enviados ao servidor, sem que esse tenha necessariamente processado os frames anteriores, causando assim um gasto de recursos como memória e processamento desnecessário. Isso ocorre pois basta que um estudante esteja presente em apenas um frame para que sua presença seja detectada e, consequentemente, não seja necessário o processamento de uma grande quantidade de frames. A utilização de uma arquitetura *near real-time* resulta em um resultado satisfatório com um pequeno atraso entre o frame exibido no cliente e o frame processado. Consequentemente, o modelo arquitetural cliente servidor, representado na Figura 17 pode ser adotado, no qual o cliente é responsável por produzir e enviar os *frames* e, ao servidor cabe o processamento e envio do *frame* contendo os rostos detectados. Como mecanismo de troca de frames, o protocolo *websocket* foi utilizado por ser *full duplex*, o que possibilita a troca de mensagens birecionalmente entre cliente e servidor e possuir uma sobrecarga baixa quanto a criação de conexões, uma vez que esta se mantém constantemente ativa. Outra característica da interface da troca de frames é a capacidade de suportar mútiplos clientes operando simultaneamente de forma independente e satisfatória dada a capacidade de processamento disponibilizada pelo servidor.

No que diz respeito às estruturas utilizadas para armazenar os dados cadastrais dos discentes, os atributos presentes na Figura 18 devem ser armazenados com intuitos particulares. O Nome completo deve ser armazenado tanto para acompanhamento e confirmação visual na interface do cliente quanto para identificação do estudante. O número de matrícula é o identificador único do discente com relação à instituição de ensino e é utilizado como indexador em outros sistemas auxiliares. Com ele portanto, faz-se possível o cruzamento de informações provenientes de fontes diversas, como sistemas de gestão acadêmica. O endereço de *e-mail* é armazenado para a eventual necessidade de contato, por parte do docente, com seus alunos. As características faciais do estudante são guardadas para que seja possível, a partir da comparação, determinar se um estudante presente em um *frame* faz parte do conjunto cadastrado a priori.

Figura 18 – Atributos de estudante



Fonte: Compilação do autor

3.1.6 Detalhes de implementação do cliente

O cliente é a parte do sistema responsável pela interface entre o usuário e as principais funcionalidades disponibilizadas pela aplicação. A implementação do cliente utiliza como plataforma de funcionamento um navegador *web* e, desta forma, pode ser acessada a partir do endereço de uma página *web*. As funcionalidades principais desta interface são o fornecimento de mecanismos para o início da chamada em uma aula, a presença de elementos visuais capazes de informar aos discentes a identificação de suas faces e, consequentemente, a atribuição de presenças em uma determinada atividade acadêmica, e a capacidade de cadastro dos dados de um discente juntamente ao sistema *back end* para que a identificação facial possa ser realizada corretamente. Como resultado destes requisitos, a aplicação desenvolvida divide em duas páginas diferentes o cliente. A primeira, é responsável pela chamada em si. Nela, encontram-se os controles para dar início ao procedimento, mecanismos para troca de *frames* entre servidor e cliente, uma lista, na qual encontram-se os nomes dos estudantes que tiveram seus rostos reconhecidos pelo sistema na sessão em execução e uma imagem contendo o resultado do processamento do último *frame* processado pelo servidor e enviado ao cliente, tal qual presente na Figura 19.

Figura 19 – *Frame* com rosto reconhecido

Fonte: Compilação do autor

Dessa forma, a interface disponibiliza mecanismos para gerenciamento, verificação e validação do funcionamento do sistema. No que diz respeito à outra página, esta é responsável pelo envio de dados cadastrais dos estudantes ao servidor da aplicação. Para que essa tarefa seja cumprida, um formulário é disponibilizado, no qual as informações presentes na Figura 18 são introduzidos pelo usuário e, além disso, uma fotografia, contendo apenas o rosto de quem deseja-se cadastrar no sistema, deve ser obtida para envio, processamento e armazenamento das características faciais por parte do servidor.

4 EXPERIMENTO

Este capítulo tem como finalidade descrever a implementação do projeto em sua totalidade, detalhando assim pontos que dizem respeito tanto a utilização de determinados *softwares* ou *hardwares*, quanto a implementação em si. O fato do projeto ter sido dividido em duas frentes distintas, o *front end* (responsável pela interface com usuários e operadores e o *back end* (responsável pelo processamento da caractrísticas faciais e armazenamento dos registros de presença) foi levado em consideração na elaboração deste capítulo. Portanto, as descrições serão segmentadas a fim de tornar mais simples a compreensão das abordagens adotadas.

4.1 PREPARAÇÃO

Nesta seção os componentes de *hardware* e o ambiente de execução da aplicação utilizados no projeto são descritos, fornecendo as devidas justificativas para tais escolhas.

4.1.1 *Hardware*

O equipamento utilizado tanto para o desenvolvimento quanto para os testes da aplicação foi um Lenovo Thinkpad X220, similar ao apresentado na Figura 20 equipado com um processador Intel Core i5-2520M *dual-core* de 2.50GHz e 3MB de memória *Cache*, com 16GB de memória RAM DDR3 à 1333MHz, 512GB de SSD, utilizando a interface SATA3. Tal máquina foi utilizada, pois durante o tempo em que o projeto foi desenvolvido, era o computador pessoal utilizado no cotidiano pelo autor para fins de estudos, pesquisa e trabalho.

Figura 20 – Notebook Lenovo Thinkpad

Fonte: https://www.notebookcheck.info/Lenovo-Thinkpad-X220-4290W1B.55922.0.html

4.1.2 *Software*

Os componentes de software utilizados nessa frente foram: Ubuntu Linux [1], Docker [2], Docker Compose [3], Python [4], MongoDB [5], Servidor web nginx [6], HTML, Bootstrap CSS [7] e JavaScript.

A aplicação e os testes desenvolvidos foram executados utilizando o sistema operacional Ubuntu Linux, na versão 18.04-LTS. Com a finalidade de simplificar o processo de execução das diferentes partes utilizadas no projeto e, ao mesmo tempo, possibilitar a fácil reprodutibilidade do experimento aqui apresentado, o Docker foi utilizado. O Docker é uma aplicação para criação e gerenciamento de *containers* (DOCKER.COM, 2021a), que por sua vez são um conjunto de aplicações e suas dependências, funcionando em um sistema de arquivos isolado do hospedeiro, que possibilita a execução de aplicações de forma rápida, fácil e confiável (DOCKER.COM, 2021c). Como consequência, o Docker abstrai o ambiente hospedeiro em que a aplicação é executada, tornando-a independente do sistema operacional utilizado. Como diversos *containers* são utilizados na aplicação, o utilitário Docker Compose foi empregado, pois ela é uma ferramenta voltada para a definição e gerenciamento de ambientes que utilizam diversos containers (DOCKER.COM, 2021b).

A infraestrutura provisionada pelo Docker consiste na utilização de quatro containers distintos:

- Traefik (versão 2.4.8) [8]
- Mongo (versão 4.4.6) [9]
- Nginx (versão 1.21.0) [10]
- Python (versão 3.8-slim-buster) [11]

O *Traefik* é uma aplicação *open-source* capaz de gerenciar o tráfego de borda destinado ao conjunto de serviços utilizados e disponibilizados no projeto. Ele recebe requisições utilizando diferentes protocolos e as direciona ao serviço de destino adequado, funcionando como um *proxy* reverso. Além disso, por padrão ele gera certificados SSL autoassinados (TRAEFIK.IO, 2021), extremamente relevantes na prevenção de ataques de *man in the*

[1] https://releases.ubuntu.com/18.04.5/
[2] https://docs.docker.com/release-notes/
[3] https://github.com/docker/compose/releases
[4] https://www.python.org/downloads/release/python-380/
[5] https://www.mongodb.com/evolvedmdbfourtwo
[6] https://nginx.org/en/download.html
[7] https://nginx.org/en/download.html
[8] https://hub.docker.com/$_{/}traefik$
[9] https://hub.docker.com/$_{/}mongo$
[10] https://hub.docker.com/$_{/}nginx$
[11] https://hub.docker.com/$_{/}python$

middle. Além disso, a utilização da funcionalidade de certificados SSL disponibilizada pelo *Traefik* torna-se necessária uma vez que o cliente utiliza elementos de *media devices* dos navegadores, que por padrão só podem ser utilizados em contextos seguros (DEVELOPER.MOZILLA.ORG, 2021). O *container* Mongo, por sua vez, disponibiliza um banco de dados MongoDB para armazenamento de dados da parte do *back end* da aplicação. O MongoDB é um banco de dados não relacional (NoSQL), projetado para facilitar o desenvolvimento e crescimento de aplicação (MONGODB.COM, 2021). Esta aplicação foi escolhida por ser facilmente integrável à linguagem utilizada para o desenvolvimento do servidor, tornando o processo de desenvolvimento mais rápido. Isso se deve ao fato da utilização de documentos, estruturas simples e versáteis, para o armazenamento de dados. Já o *container* da aplicação Nginx apresenta como atribuição a disponibilização dos arquivos estáticos (HTML, CSS e JavaScript), utilizados para a construção da interface do cliente da aplicação e responsáveis pelo envio de dados do cliente para o servidor e recebimento dos dados fornecidos pelo servidor. O *container* que utiliza como imagem base a Python:3.8-slim-buster é o responsável por abrigar o servidor da aplicação. Nele, o código desenvolvido, que especifica as regras de negócio e liga as partes distintas do sistema, é executado, disponibilizando os serviços para os clientes utilizando *sockets* TCP. Por se tratar de uma linguagem de alto nível e com uma vasta comunidade de desenvolvedores, o Python possui uma série de bibliotecas e pacotes disponibilizados que auxiliam no processo de desenvolvimento de aplicações.

Torna-se necessário, portanto, ressaltar os principais pacotes disponibilizados pela comunidade de desenvolvedores que tornaram este projeto possível. Dentre eles destacam-se:

- pymongo (versão 3.10.1) [12]
- flask (versão 1.1.2) [13]
- flask-RESTful (versão 0.3.8) [14]
- flask-SocketIO (versão 4.3.0) [15]
- dlib (versão 19.19.0) [16]
- face-recognition-models (versão 0.3.0)[17]

A biblioteca *Pymongo* foi utilizada como interface entre o código implementado em Python e o banco de dados MongoDB, utilizado para armazenar os dados cadastrais e de

[12] https://github.com/mongodb/mongo-python-driver
[13] https://github.com/pallets/flask
[14] https://github.com/flask-restful/flask-restful
[15] https://github.com/miguelgrinberg/Flask-SocketIO
[16] https://github.com/davisking/dlib
[17] https://github.com/ageitgey/face_recognition

presença dos estudantes. Além da conexão entre essas duas partes distintas do sistema, a pymongo também foi utilizada para modelar os documentos responsáveis pelo armazenamento das informações junto ao banco de dados. Já as bibliotecas *Flask, Flask-Restful e Flaks-SocketIO* foram utilizadas em conjunto para a estruturação da arquitetura de rotas do servidor. A *Flask* por padrão fornece um servidor web, capaz de receber e processar uma requisição por vez, o que significa que apenas um cliente do servidor pode ser atendido por vez. A *Flask-RESTfull* disponibiliza a utilização de um padrão declarativo para a criação de rotas utilizando o padrão de projeto REST. Tal mecanismo foi utilizado na parte do sistema responsável pelo cadastro dos dados de um novo estudante, no qual, seguindo REST, o verbo POST do protocolo HTTP foi utilizado, com as informações desejadas sendo enviadas no corpo da requisição. A *Flask-socketIO* por sua vez, foi utilizada para a criação de disponibilização de recursos que utilizem o protocolo *WebSocket*. Esses recursos foram utilizados na parte da aplicação responsável pela chamada, pois tornam possível uma conexão *full-duplex* entre servidor e cliente e com baixo *overhead* inicial. Dessa forma, a troca de frames para processamento e a notificação de presenças pode ser implementada de forma independente e sem a necessidade de um intervalo de tempo específico, podendo assim aproveitar melhor os recursos do servidor e evitar sua sobrecarga. As bibliotecas *dlib* e *face-recognition-models* foram utilizadas na parte de detecção e reconhecimento facial. *Dlib* é um conjunto de ferramentas de *machine learning*, que fornece a implementação de diversos algoritmos de alto desempenho (DLIB.NET, 2021). Esta biblioteca é crucial para o funcionamento da aplicação pois o ferramental utilizado pela biblioteca *face-recognition-models* é fornecido por ela. A *face-recognition-models* então, além de utilizar os métodos de *machine learning* da *dlib*, também contribui com os parâmetros para os modelos utilizados ao longo de todas as etapas para que o reconhecimento facial seja possível. As ferramentas fornecidas por ela obtiveram 99.38% de acurácia com testes utilizando o *dataset labeled faces in the wild* [18] (GEITGEY, 2021).

4.2 DETALHAMENTO DO FUNCIONAMENTO DO EXPERIMENTO

Nesta seção os detalhes de funcionamento de cada uma das partes do sistema, como atribuições e papel são explanados. Vale destacar também a presença da descrição dos mecanismos de comunicação entre as diferentes partes do sistema, presentes também nesta parte do texto.

4.2.1 *Infraestrutura*

A infraestrutura da aplicação utiliza como ponto de acesso o *container* que executa o Traefik. Nele, as portas TCP 80, 443 e 8080 estão habilitadas e podem ser utilizadas. A porta 80 disponibiliza o cliente da aplicação através do protocolo HTTP. Como é necessário

[18] http://vis-www.cs.umass.edu/lfw/

que a aplicação seja executada por meio de um canal seguro (encriptado) as requisições realizadas para essa porta redirecionam para o serviço disponibilizado na porta 443, que consiste no mesmo serviço, mas disponibilizando uma via de comunicação segura entre cliente e servidor, utilizando para isso HTTPS, que por sua vez implementa TLS [19]. A porta 8080 é utilizada para controle e diagnóstico do Traefik, tornando visíveis todas as subredes, roteamentos, serviços e regras de operações utilizados na configuração do programa.

4.2.2 *Cliente*

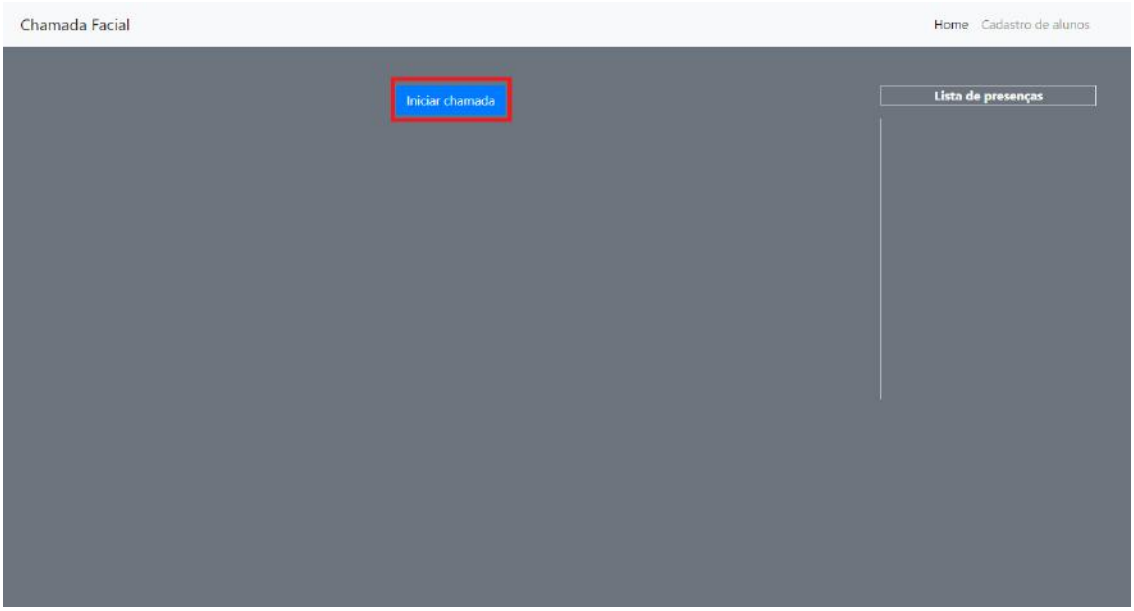
O cliente é disponibilizado no *path* "/app" da máquina em que o ambiente é executado, e é acessível por meio de um navegador web. Nele, as funcionalidades básicas para operação da ferramenta de registro de presenças estão disponíveis. Na tela inicial, um botão é utilizado para iniciar a registro das presenças, como é possível verificar na Figura 21. Uma vez o botão clicado, o vídeo será iniciado, como presente na Figura 22 e a troca de *frames* entre cliente e servidor é iniciada. Os *frames* são capturados pelo cliente, enviados ao servidor, processados e enviados de volta ao cliente. Como o modelo de funcionamento da aplicação adotado só envia um novo frame ao servidor após o último frame enviado ter sido processado e retornado ao cliente, a taxa de frames por segundo exibidos no cliente da aplicação é variável e é função do tempo de processamento dos frames enviados. Tal processamento dá-se no servidor pois esse possui acesso e gerencia os dados presentes no banco necessários para o reconhecimento facial. Além disso, essa arquitetura possibilita a utilização de clientes com pouco poder de processamento para a realização das chamadas.

A troca de frames entre cliente e servidor é feita utilizando *Web Sockets* [20] pois estes possibilitam uma comunicação interativa entre cliente e servidor, extremamente vantajosa para a troca de *frames* realizadas entre as duas partes. A conexão *Web Socket* também é utilizada quando uma face é identificada no *frame* processado pelo servidor. Dessa forma, a lista de presentes (na parte direita da Figura 22) é mantida constantemente atualizada.

Além do mecanismo de presenças, o cliente também conta com o formulário de cadastro de alunos, presente na Figura 23. Nele, um formulário é utilizado e, ao ser submetido com uma foto (que passa pela validação do servidor), os dados do estudante são inseridos no banco de dados e passam a poder serem utilizados para fins de verificação e validação de presença em sala de aula. Este cadastro é realizado através de uma requisição HTTP, utilizando para isso o método POST, uma vez que este é o indicado na arquitetura REST para a criação ou inserção de novos registros junto a um sistema.

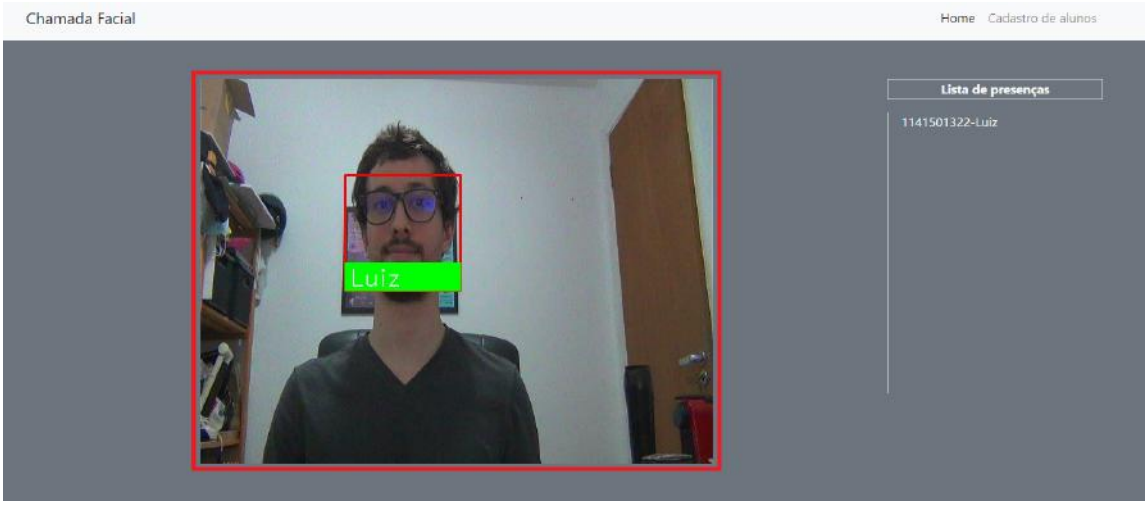
[19] Transport Layer Security
[20] https://developer.mozilla.org/pt-BR/docs/Web/API/WebSockets_API

Figura 21 – Página inicial - iniciar chamada



Fonte: Reprodução própria

Figura 22 – Página inicial - Chamada em funcionamento



Fonte: Reprodução própria

4.2.3 *Servidor*

O servidor da aplicação é responsável pelo processamento dos frames e cadastro de dados de alunos no banco de dados da aplicação. Vale ressaltar que, com a utilização do *framework* Flask, como o *gateway* do servidor *web* disponibiliza apenas uma *thread*, as requisições realizadas pelos clientes são processadas de forma sequencial. No entanto, o processamento dos *frames* enviados pelo cliente é realizado utilizando múltiplos núcleos uma vez que tal funcionalidade é diponibilizada pela *dlib*.

O acesso ao servidor é disponibilizado através do Traefik, que redireciona todas as

Figura 23 – Cadastro de alunos

Fonte: Reprodução própria

requisições que possuem como prefixo do *path* "/apifacerec". O serviço de cadastro de novos estudantes é disponibilizado através do *path* "/student_registry", através do qual um novo registro é adicionado toda vez que uma requisição que segue as seguintes regras é enviada:

- O método HTTP POST é utilizado
- Uma requisição com corpo (*body*) da mensagem do tipo JSON é enviada
- O corpo da mensagem possui as seguintes características:
  - matricula: campo contendo informações em formato de texto
  - email: campo contendo endereço de correio eletrônico da pessoa cadastrada em formato de texto
  - nome: nome completo da pessoa cadastrada em formato de texto
  - foto: foto capturada, com o rosto da pessoa cadastrada, codificada utilizando Base64

Outro mecanismo implementado pelo servidor é o serviço de detecção e reconhecimento de faces cadastradas. Nele, quando um frame é enviado via *Web Socket* para o servidor, este frame é processado, caso uma pessoa que ainda não tenha sido identificada na sessão atual seja identificada, um registro no banco de dados será inserido, constando os dados do estudante e horário em que a presença foi detectada e, simultaneamente, duas requisições serão disparadas para o cliente, uma contendo o novo *frame* processado, com o rosto devidamente destacado, e uma requisição de atualização das presenças apresentadas na página.

5 AVALIAÇÃO DA IMPLEMENTAÇÃO

Este capítulo tem por objetivo apresentar os resultados empíricos de desempenho do sistema desenvolvido, uma vez que a biblioteca utilizada no processo de reconhecimento facial apresenta uma acurácia de 99.83% nos testes realizados com o conjunto de rostos *labaled faces in the wild*[1] (KING, 2022). Tais métricas foram concebidas com a finalidade de descobrir o quão escalável o sistema é e como ele se comporta com diferentes cargas de trabalho. Para isso, o tempo de processamento de um *frame* pelo servidor da aplicação foi observado em diferentes situações, com relação ao número de clientes e faces presentes na imagem.

5.1 EXPERIMENTO PONTA A PONTA

No experimento ponta a ponta foi utilizado além do servidor da aplicação, no qual as métricas foram coletadas, o cliente desenvolvido. Dessa forma, a aplicação é avaliada em sua totalidade, remontando de fato um cenário em que a solução elaborada esteja em pleno funcionamento. A coleta dos dados foi realizada de forma independente para cada um dos cenários avaliados. Os dados coletados para análise foram:

- *socket id*: Identificador único do *websocket* utilizado na troca de frames entre cliente e servidor
- delta de tempo de processamento: tempo decorrido para que o *frame* enviado ao servidor fosse processado
- número de faces: quantidade de rostos de pessoas diferentes presentes no *frame* analisado

Para que os dados fossem condizentes com o conjunto avaliado, as métricas obtidas foram filtradas com base nos identificadores dos *websockets* e número de faces distintas presentes nos *frames*. Além disso, os dados referentes ao tempo de processamento passaram por uma transformação de segundos para milissegundos a fim de se obter uma visualização mais clara dos resultados, e foram gerados histogramas utilizando intervalos de tempo como base para a agrupamento e contagem dos tempos de processamento para representação gráfica. Esse recurso foi utilizado pois a variável do tempo de processamento é uma variável contínua e ao empregar essa técnica a visualização da métrica avaliada torna-se mais clara. Nos testes as diferentes configurações de clientes (aplicação *web*) e propriedades e características de *frame* abaixo foram analisadas.

[1] Conjunto de faces organizado pelo MIT para treinamento de sistemas de reconhecimento facial

- Um único cliente com nenhuma face presente nos *frames*
- Dois clientes com nenhuma face presente nos *frames*
- Um único cliente com uma face presente nos *frames*
- Um único cliente com duas faces presentes nos *frames*
- Dois clientes com uma face presente nos *frames*
- Dois clientes com duas faces presentes nos *frames*

5.1.1 Um único cliente sem faces presentes

Neste cenário de teste todos os *frames* avaliados foram gerados por um único cliente e não possuíam face alguma presente para os procedimentos de detecção e identificação. A Tabela 1 apresenta as métricas obtidas a partir dos dados coletados. Os dados apresentados apontam um comportamento com poucas variações neste cenário, uma vez que o valor do desvio padrão encontrado é relativamente baixo. Tal afirmação é corroborada pelo histograma presente na Figura 24, no qual a grande maioria dos *frames* processados se encontra na faixa entre 20 e 40 ms.

Tabela 1 – Métricas avaliadas para utilização de um cliente e *frames* sem faces

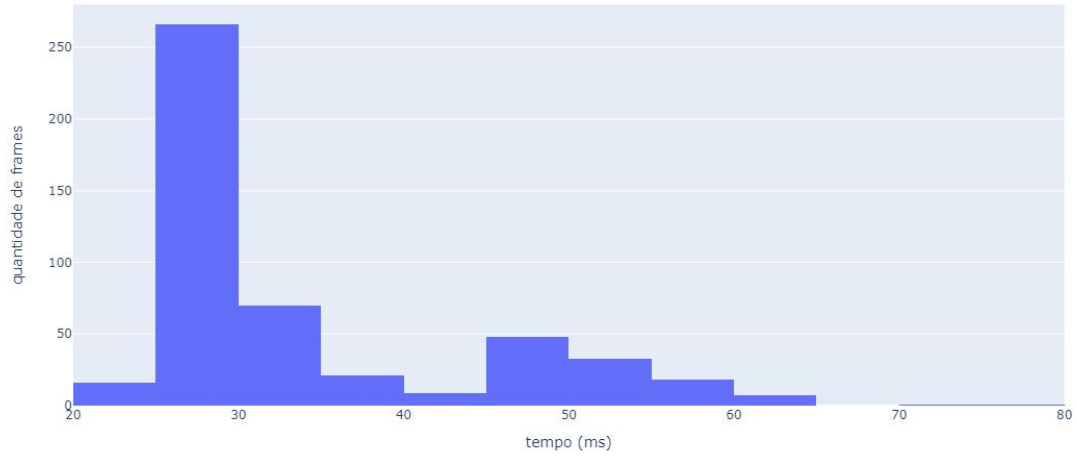
Estatística	Valor
Média	34.20 ms
Variância (σ^2)	108.60 ms^2
Desvio padrão (σ)	10.42 ms

Fonte: Reprodução própria

5.1.2 Dois cliente sem faces presentes

As amostras coletadas nessa sessão tiveram os *frames* sendo gerados a partir da utilização de dois clientes distintos, em execução simultanemente. Outro detalhe deste cenário de testes foi a ausência total de faces presentes nos *frames* capturados e processados. A Tabela 2 apresenta as estatísticas elaboradas com base nos dados coletados. Apesar de uma média de tempo de processamento para cada *frame* ter sido maior que a verificada no cenário anterior, a dispersão das amostras é ainda menor, o que evidencia um comportamento previsível para o processamento e análise de *frames* . A Figura 25 ratifica visualmente tais informações, uma vez que torna clara a grande quantidade de *frames* com tempo de processamento entre 40 ms e 65 ms.

Figura 24 – Histograma do tempo de processamento de frames: Um cliente e nenhuma face



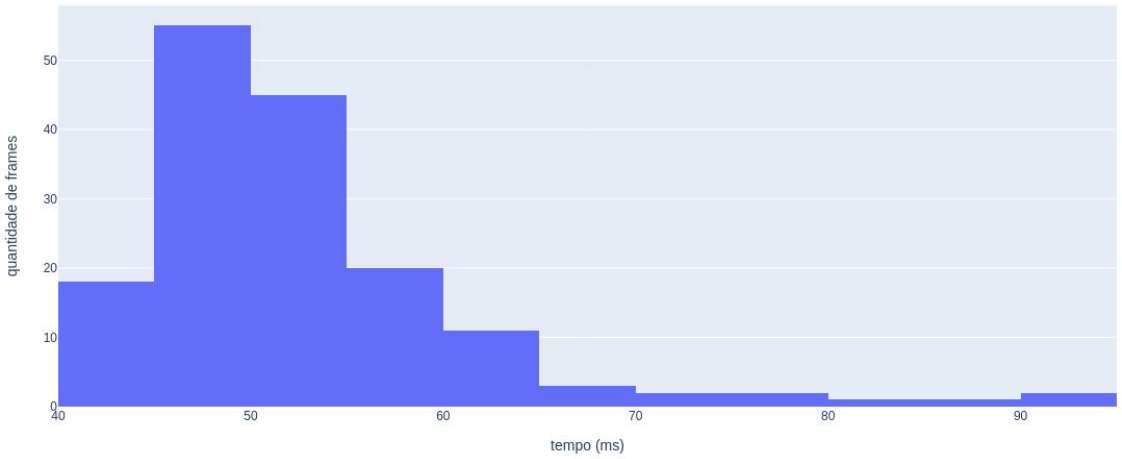
Fonte: Reprodução própria

Tabela 2 – Métricas avaliadas para utilização de um cliente e *frame* sem faces

Estatística	Valor
Média	52.62 ms
Variância (σ^2)	75.28 ms^2
Desvio padrão (σ)	8.68 ms

Fonte: Reprodução própria

Figura 25 – Histograma do tempo de processamento de frames: Dois clientes e nunhuma face



Fonte: Reprodução própria

5.1.3 Um cliente com uma face

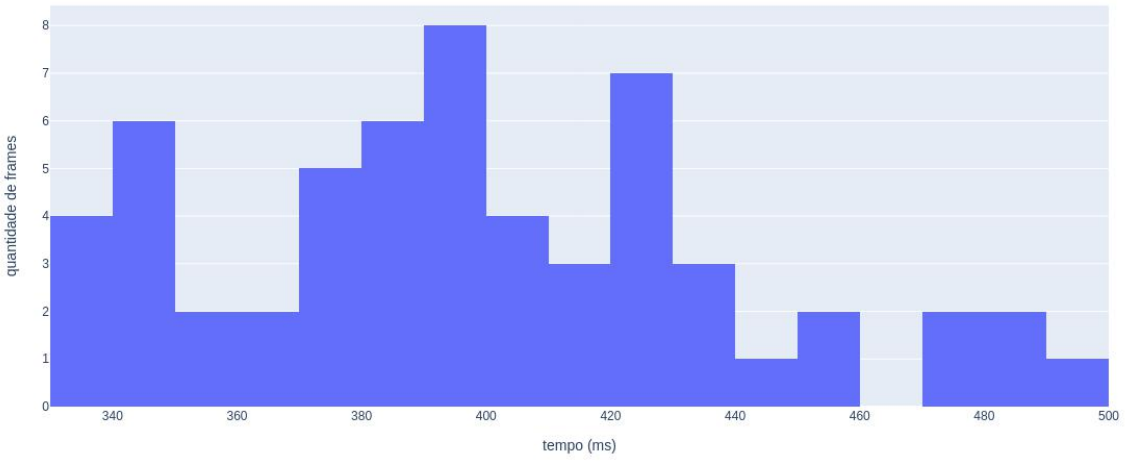
Os dados gerados nessa bateria de testes foram produzidos a partir da utilização do sistema por um único cliente e, além disso, foram contabilizadas apenas métricas geradas quando uma face foi detectada em um *frame*. A Tabela 3 apresenta os resultados obtidos a partir dos testes realizados. Em uma breve comparação com o cenário com apenas um cliente e nenhuma face presente em frames (5.1.1), a média encontrada quando ocorre a detecção e reconhecimento facial é cerca de 11.64 vezes maior, o que além de evidenciar o quão trabalhosa e dispendiosa é a tarefa de reconhecimento facial, demonstra também o impacto no servidor, que neste caso é o ponto de gargalo do sistema, ao processar *frames* com apenas uma face. Com relação ao espalhamento das amostras, o Desvio padrão é evidentemente maior que os dos dois casos analisados previamente mas, em termos de variação da distribuição, permanece relativamente baixo. Esta afirmação é corroborada pela visualização fornecida pelo gráfico presente na Tabela 26, na qual apesar de apresentar valores dispersos, a quantidade de observações concentra-se principalmente na faixa composta pelos intervalos de processamentos entre $340ms$ e $440ms$, com baixa variação nas quantidades de amostras observadas.

Tabela 3 – Métricas avaliadas para utilização de um cliente e uma face

Estatística	Valor
Média	398.20 ms
Variância (σ^2)	1702.08 ms^2
Desvio padrão (σ)	45.26 ms

Fonte: Reprodução própria

Figura 26 – Histograma do tempo de processamento de frames: Um cliente e uma face



Fonte: Reprodução própria

5.1.4 Dois clientes com uma face

Neste caso as amostras utilizadas para geração das métricas de desempenho foram coletadas utilizando dois clientes, com um enviando ao servidor da aplicação *frames* sem face alguma e outro enviando *frames* com uma face para reconhecimento e detecção. Nota-se a partir da interpretação da Tabela 4 que, os resultados obtidos encontram-se na faixa esperada, com a média do tempo de processamento dos frames entre os valores obtidos para o processamento de *frames* com um cliente, com uma face presente e sem nenhuma face presente. Tal valor é esperado nesse cenário pois trata-se de uma composição dos casos anteriormente analisados. Vale ressaltar também outro ponto tornado claro pelo histograma presente na Figura 27, na qual são facilmente identificados dois focos na distribuição. O primeiro com valores entre $0ms$ e $100ms$, correspondendo às amostras processadas que não contavam com face alguma e o outro com tempo de processamento entre $350ms$ e $550ms$, tendo sido gerados pelos *frames* que possuiam uma face em sua constituição. Outra questão que é possível ser observada a partir dos dados analisados diz respeito à dispersão das amostras. O desvio padrão elevado, quando comparado ao tempo médio de processamento é esperado uma vez que, na realidade, as amostras deste cenário foram geradas a partir de dois processos diferentes, que apresentam tempos de processamento completamente distintos. No que diz respeito ao impacto da utilização de dois clientes simultaneamente na infraestrutura do servidor, esta questão fica evidente quando utilizados como referência os casos de testes anteriores, com a utilização de apenas um cliente, nos quais as faixas em que as amostras coletadas se encontravam apresenta valores de intervalo menores do que esta, com maioria entre $25ms$ e $35ms$ no caso em que não haviam faces presentes nos *frames* e $340ms$ e $440ms$ para *frames* com um rosto presente.

Tabela 4 – Métricas avaliadas para utilização de dois clientes e *frame* sem faces

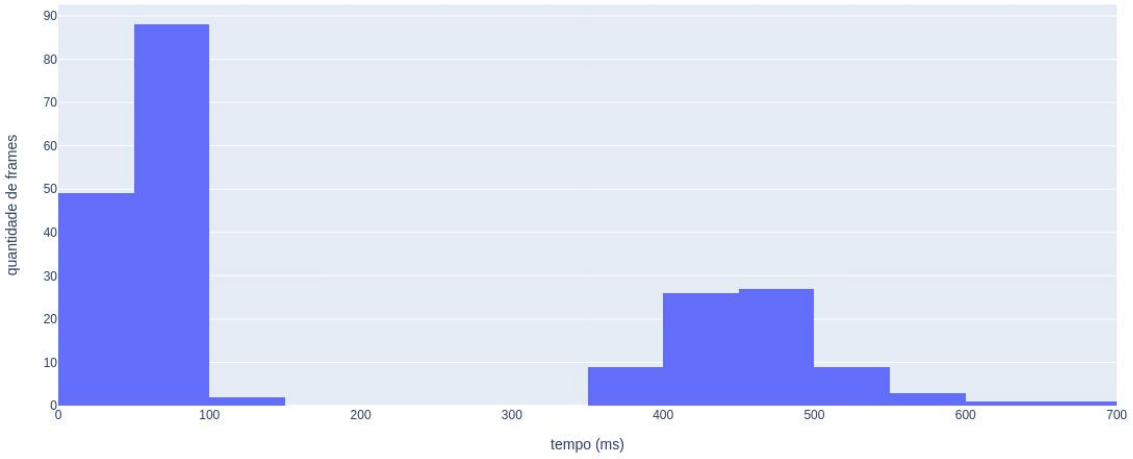
Estatística	Valor
Média	200.07 ms
Variância (σ^2)	38304.67 ms^2
Desvio padrão (σ)	195.72 ms

Fonte: Reprodução própria

5.1.5 Um cliente com duas faces

Neste cenário de testes *frames* contendo duas faces, gerados a partir de um só cliente foram utilizados para coleta dos dados. Com as métricas disponibilizadas pela Tabela 5 é possível verificar um tempo médio de processamento para cada um dos frames bastante elevado quando comparados aos valores anteriormente fornecidos pela Tabela 3 e pela Tabela 4. A comparação entre estes cenários de testes é direta pois em ambos os casos

Figura 27 – Histograma do tempo de processamento de frames: Dois clientes e uma face



Fonte: Reprodução própria

ao menos uma face estava presente em *frames* gerados por ao menos um dos clientes. É válido portanto, ressaltar a comparação entre o cenário com um cliente e uma face, no qual o tempo médio de processamento de um *frame* foi de cerca de $398.20ms$. No caso em que dois rostos encontram-se em todos os *frames* processados, temos um tempo médio de processamento de cerca de $786.25ms$. Tal grandeza é 1.97 vezes maior que a aferida com *frames* contendo apenas uma face, o que empiricamente fornece a impressão de que o tempo de processamento é função linear do número de faces presentes na imagem processada. Quanto ao espalhamento e variabilidade das amostras observadas, tanto a Variância e o Desvio padrão presentes na Tabela 5 quanto o histograma da Figura 28 demonstram a existência de variações nas amostras coletadas de forma pouco acentuada, principamente quando as métricas são comparadas aos valores de borda do intervalo de amostras.

Tabela 5 – Métricas avaliadas para utilização um cliente com duas faces

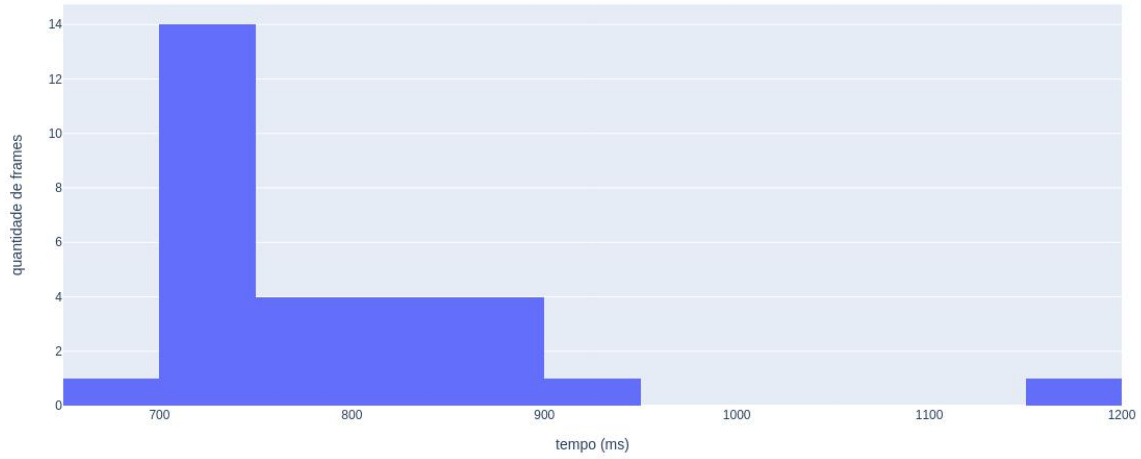
Estatística	Valor
Média	786.25 ms
Variância (σ^2)	9651.93 ms^2
Desvio padrão (σ)	98.24 ms

Fonte: Reprodução própria

5.1.6 Dois clientes com duas faces (uma em cada)

As amostras utilizadas para análise neste cenário foram geradas a partir de dois clientes executando simultaneamente, com uma face presente nas images geradas em ambos e

Figura 28 – Histograma do tempo de processamento de frames: Um cliente e duas faces



Fonte: Reprodução própria

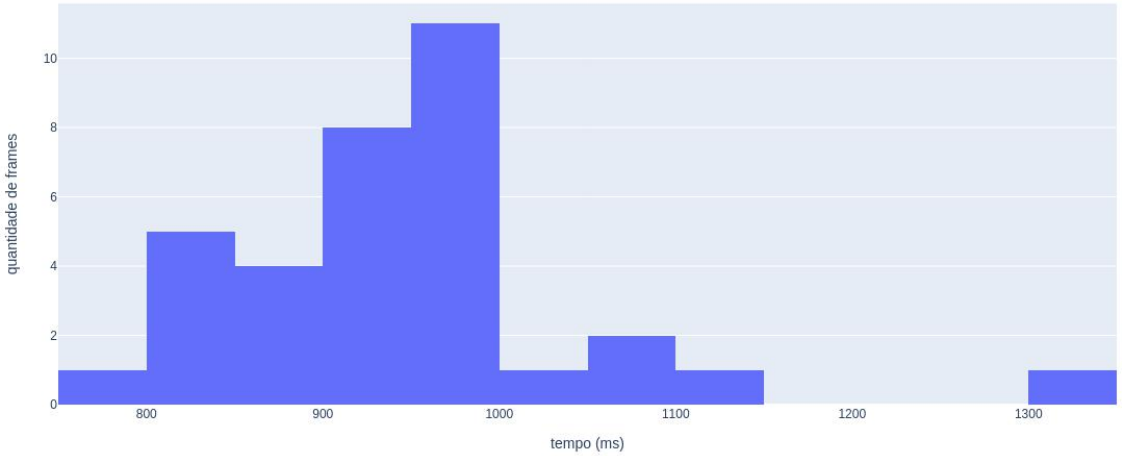
resultaram nas métricas apresentadas na Tabela 6. Desta, é possível observar um tempo de processamento médio dos *frames* de $940.59ms$, tempo muito maior que o de todos os cenários anteriores, até mesmo o cenário com dois rostos presentes nos *frames* processados. O *overhead* causado pela utilização de dois clientes é perceptível e tem um impacto tão grande no tempo de processamento e consequentemente de resposta por conta da forma como o *framework* utilizado para o desenvolvimento do servidor funciona. O Flask trata as requisições de forma sequencial, o que faz com que uma requisição passe a ser processada apenas após o término da requisição anterior. Por conta disso, o tempo decorrido desde o recebimento de um *frame* até o envio ao cliente das informações processadas depende não apenas do processamento do próprio *frame*, mas também do tempo de processamento do *frame* anterior. Como consequência, o espalhamento das amostras de tempo de processamento dos *frames* indicados tanto pela Tabela 6 quanto pelo histograma presente na Figura 29 demonstram a maior variabilidade dentre os diferentes casos de testes analisados.

Tabela 6 – Métricas avaliadas para utilização de dois clientes com duas faces (uma em cada)

Estatística	Valor
Média	940.59 ms
Variância (σ^2)	10394.59 ms^2
Desvio padrão (σ)	101.95 ms

Fonte: Reprodução própria

Figura 29 – Histograma do tempo de processamento de frames: Dois clientes e duas faces (uma em cada)



Fonte: Reprodução própria

5.1.7 Análise geral

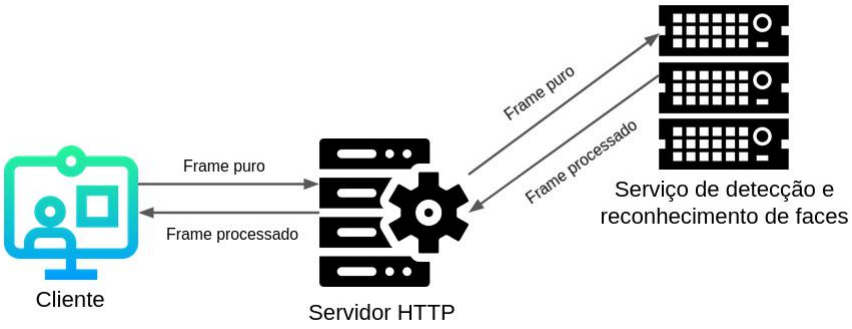
Com base nos dados apresentados fica evidente que a aplicação desenvolvida e utilizada como prova de conceito cumpre com os requisitos apresentados no Capítulo 3, sendo capaz de suportar múltiplos clientes de forma simultânea, apesar de processar apenas uma requisição de cada vez, com o sistema segregado em duas partes e a interface entre elas sendo realizada via chamadas de *API's*. Os testes realizados demonstram que, mesmo utilizando uma infraestrutura com baixo poder de processamento, o processo de reconhecimento facial é computacionalmente possível e viável, e tal ferramenta pode ser utilizada a fim de aferir presenças em sala de aula de forma passiva. Além disso, os experimentos e análises feitas em cima dos dados coletados neles tornam clara a diferença existente no tempo de processamento de *frames* em que faces estão presentes e *frames* nos quais nenhum rosto está presente. Uma análise mais cuidadosa pode ser realizada com casos de testes diferentes, que extrapolem o intuito deste trabalho, mas a partir dos dados apresentados, é possível dizer que o tempo de processamento é linearmente dependente da quantidade de clientes que utilizam o servidor. Outro ponto que deve ser destacado é o aumento expressivo do tempo necessário para o processamento de *frames* com faces. A diferença encontrada no caso isolado de apenas um cliente, com uma face presente foi mais de dez vezes maior do que quando nenhuma face estava no *frame* processado, também com um cliente. No entanto, apesar da disparidade, comparando os casos em que um cliente com uma face e um cliente com duas faces foram testados, pode-se inferir que o tempo de processamento também é função linear do número de faces presentes na imagem analisada.

De tal maneira, pode-se concluir que a parte crítica, e consequentemente o gargalo da aplicação desenvolvida encontra-se no processamento de *frames* por parte do servidor para a realização da detecção e reconhecimento facial. Mesmo o sistema utilizando todos os núcleos de processamento disponibilizados pela máquina na qual os testes foram executados no fluxo de processamento da imagem (KING, 2009), a quantidade de tempo despendida para o processamento de um único *frame*, principalmente quando faces estão presentes na imagem, foi expressiva. Por conta disso, a arquitetura monolítica utilizada para o desenvolvimento da aplicação, que torna impossível escalar apenas a parte crítica do sistema para que um melhor desempenho seja obtido mostra-se ineficaz. Outra questão que impacta negativamente no desempenho da aplicação desenvolvida é o *hardware* utilizado nos testes.

5.1.8 Proposta de arquitetura para melhoria de desempenho da aplicação

Com o intuito de melhorar o desempenho da aplicação, o que se traduz numa latência menor entre a geração e captura de um *frame* pelo cliente, seu processamento pelo servidor e a exibição da imagem processada, diferentes melhorias podem ser implementadas, destacando-se principalmente a estruturação do servidor em microsserviços. Nela, dividindo o sistema em dois, uma parte ficaria responsável pelo recebimento das imagens e outra pelo seu processamento, como podemos verificar na Figura 30. Dessa forma, a parte crítica da aplicação, reponsável pela detecção e reconhecimento das faces pode escalar conforme a demanda. Assim o processamento de diversos *frames* pode ocorrer em paralelo, diminuindo o impacto da carga gerada por um grande número de clientes funcionando simultaneamente.

Figura 30 – Arquitetura de sistema em microsserviços



Fonte: Reprodução própria

6 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo tem por objetivo apresentar uma visão geral dos demais capítulos e as conclusões acerca do projeto desenvolvido. Os possíveis desdobramentos da implementação realizada também serão contemplados, levando em consideração para isso novas funcionalidades que agregam ao funcionamento e utilidades do sistema, e que impactam no desempenho da aplicação desenvolvida.

6.1 CONCLUSÃO

O projeto desenvolvido neste trabalho de conclusão de curso torna explícitas as possibilidades criadas pela visão computacional e processamento e análise de imagens. Isto deu-se através do planejamento, implementação e descrição das técnicas utilizadas em um sistema de presença em sala de aula que utiliza-se de reconhecimento facial.

No Capítulo 2 uma definição inicial de visão foi fornecida com base nos sistemas sensoriais utilizados por seres humanos para ver o ambiente e uma descrição dos desafios e dificuldades de fazer máquinas enxergarem a partir do processamento de imagens e reconhecer padrões foi realizada. Com base em tais desafios foram desenvolvidos os demais pontos do capítulo que endereçam o processamento inicial de uma imagem para que seja mais fácil encontrar objetos nela, técnicas específicas para detecção de faces e reconhecimento de pessoas a partir de suas características faciais. Apesar de este capítulo dar um enfoque grande na parte de reconhecimento facial, as técnicas apresentadas podem ser generalizadas e utilizadas em outros cenários relacionados a processamento de imagens para reconhecimento de padrões.

Já no Capítulo 3 a proposta de um sistema prova de conceito voltado para registro de presença em sala de aula, que utilizasse as funcionalidades de processamento de imagem, detecção e reconhecimento facial, foi feita. Nele, além da proposta, estão presentes também a arquitetura do sistema, seu fluxo de execução, seus requisitos funcionais e funcionalidades.

O Capítulo 4 explora em detalhes o projeto desenvolvido em diferentes aspectos como hardware, infraestrutura, linguagens e bibliotecas e atribuições das diferentes partes do sistema desenvolvido.

No Capítulo 5 uma análise empírica com base em dados coletados a partir de diferentes configurações de funcionamento do sistema é feita e, a partir disso, chega-se a conclusões acerca do desempenho da aplicação e validade e viabilidade da sua utilização no cenário proposto

Utilizando os capítulos anteriores como referência, conclui-se que este trabalho de conclusão de curso cumpre com sua proposta, uma vez que nele foi possível explorar

diferentes nuances da visão computacional e, além disso, os conhecimentos prévios de tais técnicas e tecnologias foram aprofundados e aplicados em um cenário fictício atrevés do desenvolvimento de uma solução prática para um problema real. No entanto, mesmo o objetivo principal tendo sido atingido, o projeto desenvolvido ainda apresenta diversos pontos de melhoria que podem ser explorados futuramente.

6.2 TRABALHOS FUTUROS

Os próximos passos do trabalho aqui apresentado seguem linhas com enfoques diferentes. Na vertente de análise de desempenho da aplicação, com a utilização de uma placa de vídeo dedicada com suporte a CUDA[1], pode ser implementada, uma vez que a dlib, a biblioteca utilizada no processamento de imagens, tem suporte para tal. Esta análise seria valiosa uma vez que a parte mais custosa em termos de processamento para a aplicação poderia ser paralelizada e, a princípio, realizada de forma muito mais rápida. Além deste cenário, uma outra possível alternativa para melhoria de desempenho seria reorganizar a arquitetura do servidor como proposto na subseção 5.1.8. Ao realizá-la, a parte crítica do servidor escalaria de acordo com a demanda. Esta análise destaca-se pois com a realização da coleta de dados e métricas de desempenho pode-se chegar a informações como a quantidade necessária de instâncias da seção crítica para que cada cliente em operação tenha um desempenho mínimo satisfatório. Uma terceira alternativa seria utilizar o protocolo HTTP/2 no lugar do HTTP/1.1 e verificar seu impacto no desempenho da aplicação. Como a versão 2 do protocolo sendo utilizada para a comunicação entre cliente e servidor espera-se que o tempo de envio das imagens entre as duas partes do sistema seja reduzida uma vez que nesta versão o protocolo é binário. Além disso, aliando esta funcionalidade com uma das duas anteriormente sugeridas, é possível utilizar a multiplexação do protocolo para o envio de grupos ou blocos de *frames* ao servidor e esse, por sua vez retorna aquele que for processado mais rapidamente.

Uma outra vertente que poderia ser explorada diz respeito as métricas utilizadas para a avaliação de desempenho do sistema. Uma métrica que pode ser explorada no cenário em que mais de um cliente utilizava o servidor da aplicação ao mesmo tempo é a de tempo de espera para início do processamento do *frame* enviado. Cabe também uma outra análise na qual o impacto da quantidade de pessoas registradas em uma turma impacta no tempo de resposta do servidor, uma vez que com um número maior de pessoas mais comparações precisarão ser realizadas para detecções faciais.

Além disso, para que o projeto desenvolvido deixe de ser um mínimo produto viável e passe a ser um protótipo experimental faz-se necessário incorporar outras funcionalidades ao sistema como identificação da disciplina e docente responsável pela sessão de presenças, Atualização dos dados pessoais dos discentes, assim como a atualização automática

[1] https://developer.nvidia.com/cuda-zone

de seus marcos faciais no banco de dados ao longo do tempo e a exportação de relatórios detalhados de presença para determinados recortes temporais. Uma outra funcionalidade que pode ser implementada e explorada é a integração com sistemas de gestão acadêmica. Com tal integração seria possível manter os registros acadêmicos dos discentes sempre atualizados e, além disso, as informações relativas às presenças e características faciais poderiam ser armazenadas junto aos dados pessoais e informações relativas ao desempenho, geradas pelos estudantes ao logo da jornada acadêmica. Dessa forma, a realização de estudos com base em perfil, características comportamentais e desempenho acadêmico poderiam ser realizados sem grande dificuldades. Outras funcionalidades que agregariam ao sistema dizem respeito a mecanismos alternativos para contabilização e atribuição de presenças em situações em que o sistema não se comporte como o esperado. Com relação a confirmação e comprovação das presenças, mecanismos de notificação dos alunos poderiam ser desenvolvidos, nos quais seriam recebidos comprovantes de presença datados e, além disso, uma visão geral de presenças ao logo do tempo poderia ser fornecida.

REFERÊNCIAS

ALOIMONOS, Y. Special issue on purposive and qualitive active vision, cvgip b: Image understanding. **CVGIP B: Image Understanding**, v. 56, 1992.

AMOS, B.; LUDWICZUK, B.; SATYANARAYANAN, M. **OpenFace: A general-purpose face recognition library with mobile applications**. [S.l.], 2016.

DALAL, N.; BILL, T. Histogram of oriented gradients for human detection. 2005.

DEVELOPER.MOZILLA.ORG. **Media Devices - MDN**. 2021. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices>. Acessado em : 01/07/2021.

DLIB.NET. **dlib**. 2021. Disponível em: <https://dlib.net/>. Acessado em : 04/07/2021.

DOCKER.COM. **Get started page - Docker**. 2021. Disponível em: <https://www.docker.com/get-started>. Acessado em : 23/06/2021.

DOCKER.COM. **Overview of Docker Compose - Docker**. 2021. Disponível em: <https://docs.docker.com/compose/>. Acessado em : 23/06/2021.

DOCKER.COM. **What is a container page - Docker**. 2021. Disponível em: <https://www.docker.com/resources/what-container>. Acessado em : 23/06/2021.

GEITGEY, A. **Modern face recognition with machine learning**. 2016. Disponível em: <https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78>. Acessado em : 10/10/2020.

GEITGEY, A. **face_reconition package description**. 2021. Disponível em: <https://github.com/ageitgey/face_recognition>. Acessado em : 04/07/2021.

HUANG, T. Computer vision: Evolution and promise. **CERN School of Computing**, v. 19, p. 21–25, 1996.

KAZEMI, V.; KTH, J. S. One millisecond face alignment with an ensemble of regression trees. 2014.

KING, D. **dlib blog**. 2022. Disponível em: <http://blog.dlib.net/2017/02/high-quality-face-recognition-with-deep.html>. Acessado em : 10/02/2022.

KING, D. E. Dlib-ml: A machine learning toolkit. **Journal of Machine Learning Research**, v. 10, p. 1755–1758, 2009.

LODRIGUSS, J. **How digital cameras work**. 2016. Disponível em: <https://www.astropix.com/html/i_astrop/how.html>. Acessado em : 08/10/2020.

MALLIK, S. **Histogram of Oriented Gradients**. 2016. Disponível em: <https://www.learnopencv.com/histogram-of-oriented-gradients/>. Acessado em : 20/10/2020.

MONGODB.COM. **MongoDB Docs - MONGO**. 2021. Disponível em: <https://docs.mongodb.com/manual/>. Acessado em : 30/06/2021.

RAWLINSON, T.; BHALERAO, A.; WANG, L. Principles and Methods for Face Recognition and Face Modelling. n. January 2010, p. 53–78, 2010.

ROBERTS, L. **Machine perception of 3D solids**. [S.l.]: MIT Press, 1965. 159-197 p.

SHARMA, S.; SHANMUGASUNDARAM, K.; RAMASAMY, S. Farec - cnn based efficient face recognition technique using dlib. **Proceedings of 2016 International Conference on Advanced Communication Control and Computing Technologies, ICACCCT**, p. 192–195, 2016.

TALUKDAR, P.; GANGULY, K. A study on the simulation of visual prosthesis for the restoration of functional vision to the blind. **IRF International Conference**, v. 37, p. 43–46, 2017.

TRAEFIK.IO. **Docs - Traefik**. 2021. Disponível em: <https://doc.traefik.io/traefik/>. Acessado em : 30/06/2021.

APÊNDICE A – CÓDIGOS

Os códigos desenvolvidos neste trabalho de conclusão de curso encontram-se disponíveis em um repositório público, hospedado no Github, e podem ser encontrados neste link: https://github.com/luizgribeiro/tcc_facerec.