INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO AMAZONAS

CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Eduardo Amorim de Araújo

SISTEMA DE RECONHECIMENTO FACIAL APLICADO À GESTÃO DE SEGURANÇA EM CONDOMÍNIOS

Manaus, Amazonas – Brasil

2019

EDUARDO AMORIM DE ARAÚJO

SISTEMA DE RECONHECIMENTO FACIAL APLICADO À GESTÃO DE SEGURANÇA EM CONDOMÍNIOS

Trabalho de Conclusão de Curso apresentado à banca examinadora do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Amazonas – IFAM Campus Manaus Centro, como requisito para o cumprimento da disciplina TCC2 – Desenvolvimento de Software.

Orientação: Prof.ª Dr.ª Joyce Miranda dos Santos.

Manaus, Amazonas – Brasil

2019

FOLHA DE APROVAÇÃO

EDUARDO AMORIM DE ARAÚJO

Esta monografia de Conclusão de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciências e Tecnologia do Amazonas, Campus Manaus Centro, foi julgada e Aprovada pela Banca Examinadora:

Aprovada em 28 de Junho de 2019

_____

Prof.ª Dra. Joyce Miranda dos Santos

Orientadora

_____

Prof.º MSc. Emmerson Santa Rita

Examinador

_____

Prof.º Dr. Jucimar Brito de Souza

Examinador

DEDICATÓRIA

“Disciplina é a ponte entre metas e realizações” (Jim Rohm)

AGRADECIMENTOS

Agradeço a Deus que me deu forças para concluir este projeto de forma satisfatória. Agradeço a minha mãe pelo apoio incondicional em todos os momentos difíceis da minha trajetória acadêmica, ao meu irmão e minha namorada por sempre me motivar e apoiar.

Agradeço ao Instituto Federal de Educação, Ciência e Tecnologia do Amazonas - IFAM, por ter me dado o privilégio de estudar em uma excelente instituição de ensino, oferecendo sempre professores capacitados e laboratórios de qualidade. Agradeço em especial a Prof.ª Dra. Joyce Miranda dos Santos que teve papel fundamental na realização desse TCC.

RESUMO

Devido ao grande número de assaltos a condomínios, soluções alternativas têm sido investigadas com o objetivo de inibir a entrada de pessoas não autorizadas nesse tipo de ambiente. Com o avanço tecnológico na área de biometria, o reconhecimento facial tem sido uma alternativa bastante aplicada para este fim. Este trabalho se propôs a investigar e utilizar técnicas de Aprendizagem Profunda com uso de redes neurais para a tarefa de reconhecimento facial dentro do contexto de controle de acesso a condomínios. Para isso, foi desenvolvida uma aplicação web capaz de gerenciar blocos, apartamentos e moradores do condomínio integrado a um módulo de visão computacional responsável por retornar o resultado do processo de reconhecimento. A partir dos experimentos realizados, foram obtidas algumas métricas para avaliar o desempenho do método de reconhecimento aplicado no trabalho, que reportaram resultados satisfatórios para o fim proposto.

Palavras Chaves: *reconhecimento facial, biometria, Aprendizagem Profunda, redes neurais.*

ABASTRACT

The wide report of the condom in the condomances, this work in the solution to connect in the entry of personal people are not allowed, using an mechanism to facial, the end of human identity, and their their knowledge. With the technological code in the area of biometrics, facial treatment has been one of the most widespread for this purpose. This work should be used with Learning techniques with the use of neural networks and an end of its results with facial recognition. A web application capable of protecting blocks, apartments and residents of the condominium was developed. For the communication of the computer vision module with a web application, the Apache Kafka Messaging Server, responsible for the recognition process was used. Some rates were calculated for the performance of the facial method, returning a number of false positives and negative errors, but for a purpose proposed for this work, the results were satisfactory.

Keywords: face recognition, biometry, deep learning, neural networks.

LISTA DE FIGURAS

Figura 1- Índice de assaltos 11
Figura 2- Representando Deep Learning 18
Figura 3- Representação OpenFace 19
Figura 4- Arquitetura, trabalho relacionado 21
Figura 5- Arquitetura REST 25
Figura 6- Arquitetura básica Apache Kafka 27
Figura 7- Patitions Apache Kafka 28
Figura 8- Diagrama de Caso de Uso 29
Figura 9- Modelagem do banco de dados 30
Figura 10- Diagrama de Classe 30
Figura 11- Tela inicial 31
Figura 12- Tela de listagem de blocos 32
Figura 13- Tela de listagem de apartamentos 32
Figura 14- Informações do morador I 33
Figura 15- Tela de Cadastro 33
Figura 16- Tela de interação do módulo de visão com webcam 34
Figura 17- Tela Com Validação 34
Figura 18- Tela sem validação 35
Figura 19- Arquitetura Kafka 36
Figura 20- Arquitetura geral do módulo de cadastro 37
Figura 21- Arquitetura geral do módulo de cadasto II 37
Figura 22- Exemplo endpoints 39
Figura 23- Código *back-end* acessando Apache Kafka 40
Figura 24- Exemplo Dataset 40
Figura 25- Técnica utilizada no treinamento da rede 41
Figura 26- Gerando encodings 42
Figura 27- Method face_recognition 43
Figura 28- Arquitetura de alto nível do módulo de reconhecimento facial 44
Figura 29- Arquitetura de alto nível do módulo de reconhecimento II 44
Figura 30 – Amostra de imagens do *Dataset* utilizado nos experimentos 45
Figura 31- Dataset de cadastro original 46
Figura 32- Dataset de cadastro recortado 46
Figura 33- Métricas obtidas sem CUDA, tabela I linha I 16
Figura 34- Métricas obtidas sem CUDA, tabela I, linha II 16
Figura 35- Métricas obtidas com CUDA, tabela I, linha IV 16
Figura 36- Mátricas obtidas com CUDA, tabela I linha V 16
Figura 37 - Enviar objeto para o broker 17
Figura 38- Criando vetores com os IDs, previsão e os tempos 17
Figura 39- Enviar e consumir dados do broker 18
Figura 40- Adicionar atributos de tempo e ID previsto e enviar pro broker 18

LISTA DE TABELAS

Tabela 1- Resultados de trabalhos relacionados 22
Tabela 2- Tabela de endpoints 38
Tabela 3- Procedimentos de teste 47
Tabela 4- Comparação entre imagens de teste e cadastro 48
Tabela 5- Testes sem/com CUDA 11
Tabela 6- Resultado de testes com distância de 1m 11
Tabela 7- Resultado de testes com distância de 3 m 12
Tabela 8- Descrição Caso de Uso 15

Sumário

1 INTRODUÇÃO 11
1.2 Objetivos 14
1.2.1 Objetivo geral 14
1.2.2 Objetivo específico 14
2 FUNDAMENTAÇÃO TEÓRICA 15
2.1 Processamento digital de imagens (PDI) 15
2.2 Biometria 15
2.3 Aprendizagem de Máquina 16
2.4 Aprendizagem Profundo 17
2.5 OpenFace 18
2.6 Classificador K-Neareste Neighbors 20
3 TRABALHOS RELACIONADOS 21
4 FERRAMENTAS E TECNOLOGIAS 23
4.1 OPENCV 23
4.2 Python 23
4.3 Angular 2x 24
4.4 Representational State Transfer (REST) 24
4.5 UML 25
4.6 Bootstrap 25
4.7 MySQL 26
4.8 Java – Spring Boot 26
4.9 Apache Kafka 27
4.10 Dlib 28
5 PROPOSTA DE SOLUÇÃO 29
5.1 Introdução 29
5.2 Estrutura da Proposta 29

5.3 Modelagem do Sistema 29
5.3.1 Diagrama de caso de uso 29
5.3.2 Modelagem do banco de dados 30
5.3.3 Diagrama de classe 30
6 IMPLEMENTAÇÃO DA PROPOSTA 31
Neste capítulo será apresentado o detalhamento sobre a implementação do sistema proposto neste trabalho. 31
6.1 Telas da Aplicação 31
6.2 Arquitetura da Aplicação 35
6.2.1 Aplicação Front-End 35
6.2.2 Servidor de mensageria 35
6.2.1 Aplicação Back-end 36
6.2.4 Módulo de Visão computacional 41
7 EXPERIMENTOS 45
8 CONCLUSÃO 13
8.1 TRABALHOS FUTUROS 13
REFERÊNCIAS 14

1 INTRODUÇÃO

Os condomínios fechados sempre foram vistos como uma solução segura de moradia em grandes cidades. Esses espaços são vistos como um modelo de moradia tranquilo, oferecendo aos seus moradores conforto e proteção. Entretanto, o aumento do número de assaltos têm sido cada vez mais frequente. Dados da Secretaria de Segurança Pública(SSP) apontam que o número de roubos e furtos em condomínios no Estado de São Paulo cresceu 56% no ano de 2018, conforme apresentado na Figura 1. Foram 1300 crimes contra prédios, registrados entre janeiro e abril de 2018, contra 832 no mesmo período no ano anterior.

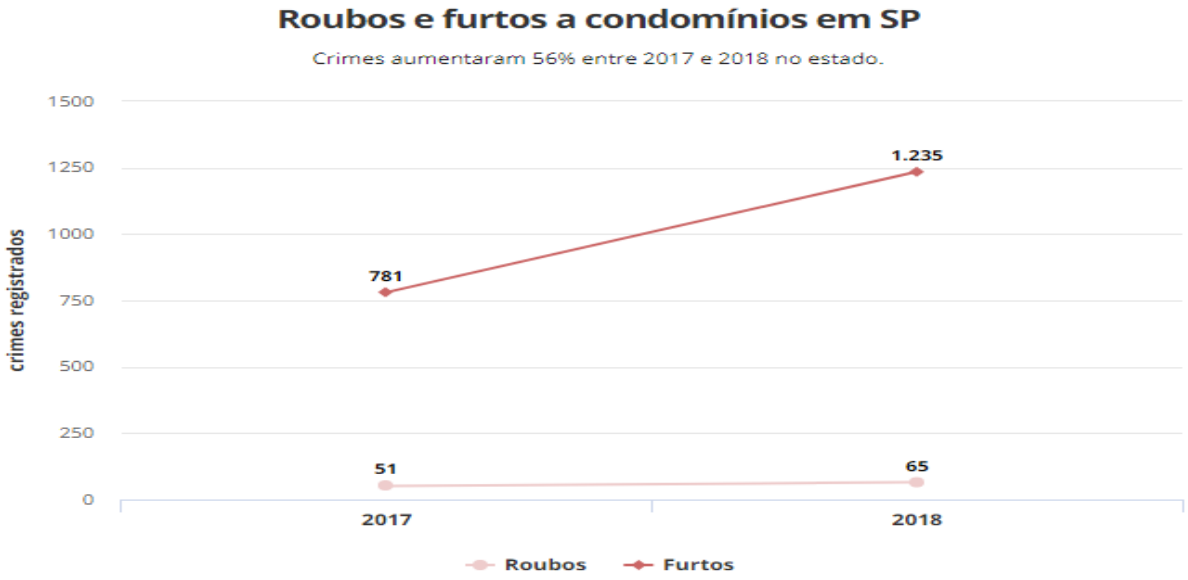


Figura 1- Índice de assaltos

Quadrilhas mais especializadas, aproveitam-se das falhas de segurança existentes e invadem os condomínios para realizar roubos. Para se obter a segurança e privacidade desejada se faz necessário identificar e conhecer os principais riscos e ameaças para a segurança do condomínio. Uma das maiores vulnerabilidades de um condomínio é o controle de acesso dos moradores.

Existem algumas soluções que possibilitam atender à necessidade de restringir o acesso somente para pessoas autorizadas, tais como o uso de painel de senha, cartões magnéticos e controle de acesso biométrico.

Controle de acesso biométrico é um método automatizado de verificação ou reconhecimento da identidade de uma pessoa com base em algumas características fisiológicas,

tais como impressões digitais ou características faciais LIN (2000). Entre os vários métodos de identificação biométrica, os métodos de fisiologia (impressão digital, face, DNA) são os mais estáveis. O motivo é que segundo LIN (2000), as características fisiológicas são muitas vezes não alteráveis, exceto por lesão grave.

Segundo LIN (2000), o reconhecimento facial é um dos poucos métodos biométricos que possuem os méritos de alta precisão. Ao longo dos anos, diversos algoritmos de reconhecimento facial foram desenvolvidos, mas ainda não há nenhum que reconheça faces com a mesma eficiência que o ser humano reconhece, isto é, que reconheça faces em qualquer ambiente, vistas de qualquer ângulo, e não importando a expressão facial. Porém, em condições controladas, existem sistemas que superam o desempenho humano, sendo até mesmo, capazes de diferenciar gêmeos monozigóticos, JAFRI e ARABNIA (2005). Constantemente, vem surgindo novas formas para realizar o reconhecimento facial de maneira mais eficiente, aprendizagem de máquina é uma delas.

De robôs a motores de busca, a aprendizagem de máquina (*Machine Learning*), têm sido cada vez mais integrados à vida cotidiana das pessoas. Embora os conceitos de *Machine Learning* tenha suas raízes em pesquisas realizadas na década de 1960, ela está ganhando um novo impulso no mercado com a era da *Big Data*, onde um grande volume de dados está mais acessível. A tecnologia de aprendizagem de máquina é considerada um subcampo da Inteligência Artificial (IA), que trabalha com a ideia de que as máquinas podem aprender sozinhas a partir da análise de um conjunto de dados. Para que isso seja possível, são criados algoritmos que utilizam análises estatísticas aprimoradas sobre as informações que recebem, resultando em respostas e previsões mais precisas.

De forma geral, a aprendizagem de máquina pode ser classificada como: aprendizagem supervisionada e aprendizagem não supervisionada. A aprendizagem supervisionada necessita que haja um treinamento sobre uma base de dados rotulada. Na aprendizagem não supervisionada, os algoritmos não precisam ser treinados com uma base de dados rotulada. Nesse caso, a aprendizagem ocorre a partir de uma análise no conjunto de dados para determinar se alguns deles podem ser agrupados de alguma maneira, formando *clusters* (grupos).

Técnicas de Aprendizagem Profunda vêm aprimorando a capacidade dos computadores de realizarem tarefas como: classificar imagens; processamento de linguagem natural, reconhecimento de fala, sistema de recomendações, criação de carros autônomos. Sistemas como Siri (assistente virtual da Apple), Cortana (assistente virtual da Microsoft) e Google *Translate* (Google Tradutor) são parcialmente alimentados por aprendizagem profunda. O que aprendizagem profunda faz, é realizar um “treinamento” de um modelo computacional, para

que ele possa decifrar a linguagem natural. O modelo relaciona termos e palavras para inferir significado, uma vez que é alimentado com grandes quantidades de dados. Usando várias camadas de processamento de dados não lineares, é possível obter uma representação complexa e abstrata dos dados de forma hierárquica. Dados sensoriais (pixels de imagens, por exemplo) são apresentados a uma primeira camada, sendo a saída de cada uma delas, torna-se a entrada da camada seguinte. Logo o empilhamento de várias camadas desses "neurônios", é a ideia básica dos algoritmos de aprendizagem profunda.

O Processamento Digital de Imagens (PDI) é a manipulação de imagens de maneira que tanto a entrada quanto a saída do processo seja uma imagem. Em algumas fases no processo de reconhecimento facial é necessário realizar o PDI para melhorar o desempenho do algoritmo de treinamento e reconhecimento de imagens.

A principal motivação para a realização deste trabalho é disponibilizar uma aplicação de reconhecimento facial utilizando os conceitos de aprendizagem de máquina e aprendizagem profunda, que, embora neste trabalho esteja focada no escopo de controle de acesso a condomínios, poderá ser facilmente adaptada para outros cenários de controle de acesso a ambientes restritos, visto que o projeto foi desenvolvido de forma modularizada, onde cada módulo da aplicação é responsável por uma tarefa específica. O módulo de visão computacional pode ser integrado com qualquer outra aplicação que implemente uma interface adequada para consumir a resposta do reconhecimento facial. Algumas áreas em que o sistema de reconhecimento facial pode ser aplicado: aeroportos, para identificação de terroristas; automação bancária; reconhecimento de usuários em caixas eletrônicos, entre outras atividades que requerem o reconhecimento de uma pessoa específica.

Desta forma, este trabalho pretende resolver a seguinte problemática: Qual a melhor estratégia para combinar técnicas de visão computacional para alcançar melhores resultados de eficácia para o problema de reconhecimento facial de imagens dentro do contexto de controle de acesso a condomínios?

1.2 Objetivos

1.2.1 Objetivo geral

O principal objetivo deste trabalho é investigar e aplicar técnicas otimizadas de visão computacional na implementação de um sistema que se utiliza do reconhecimento facial para controlar acessos em condomínios.

1.2.2 Objetivo específico

- Fazer o levantamento das principais técnicas de reconhecimento facial e de processamento digital de imagens aplicáveis ao contexto deste trabalho;
- Implementar módulo de reconhecimento facial e avaliar a eficácia dos resultados obtidos.
- Propor uma solução eficiente para integração entre os módulos de visão computacional e aplicação do usuário.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece informações fundamentais para auxiliar no entendimento de alguns conceitos envolvidos na implementação deste trabalho.

2.1 Processamento digital de imagens (PDI)

Processamento digital de imagens é a manipulação de imagens de maneira que tanto a entrada quanto a saída do processo seja uma imagem. Em algumas fases no processo de reconhecimento facial é necessário realizar o PDI para redimensionar a resolução das imagens, converter em escala de cinza, entre outras funções, a ideia em processar as imagens é que ela seja representada com mais efetividade no contexto computacional, uma vez que ela é convertida em dados, melhorando a performance dos algoritmos de aprendizagem de máquina.

Os dados capturados após o processamento são submetidos a técnicas a fim de obter variações favoráveis da imagem para que essas variações sejam utilizadas em outros processamentos CÂMARA(1996).

De acordo com Gonzalez e Woods (2012) o espectro que vai do PDI até a visão computacional, passa por 3 níveis, sendo eles baixo nível, nível médio e alto nível. O processo de baixo nível consiste em operações primitivas como redução de ruídos ou melhorias no contraste da imagem, nível médio que são processos como segmentação ou classificação, já os de alto nível são relacionados a tarefas de cognição associadas a visão humana. São tarefas relativas ao baixo nível SANTOS (2011).

2.2 Biometria

A biometria é a ciência que consiste em mensurar as características dos indivíduos, tal ciência demonstra que cada indivíduo é singular, JAIN (2007) explica que a biometria é a ciência que estabelece a identidade do indivíduo baseado nos seus atributos físicos, químicos e comportamentais, que é exclusivo de cada pessoa.

Seguindo essa linha de raciocínio temos que SRIVASTAVA (2013) complementa que o corpo de uma pessoa ou os perfis comportamentais da mesma podem ser usados para identificá-la de maneira única e exclusiva através da biometria, tais dados biométricos podem ser fisiológicos ou comportamentais.

SRIVASTAVA (2013) destaca que a biometria se divide em duas categorias, a citar, características fisiológicas e comportamentais onde cada categoria de biometria pode ser medida usando diferentes métricas.

Biometria Fisiológica: Está atrelada a cada corpo humano, sendo difícil ser falsificada e com poucas chances de perder suas características ao longo do tempo. As mais comuns são os DNAs, impressões digitais, características faciais, geometria da mão, palma da mão, retina e íris.

Biometria Comportamental: Dependente da condição psicológica ou mesmo do ambiente que a pessoa está inserida, podendo sofrer variações dependendo do estado sentimental do indivíduo ou doenças, que afetam seu jeito de falar, andar ou se expressar de uma maneira geral, como exemplo temos a voz, assinaturas e ritmo de digitação.

2.3 Aprendizagem de Máquina

A aprendizagem de máquina é utilizada na fase de mineração de dados do processo de descoberta do conhecimento em base de dados e surgiu da percepção de criar programas computacionais que aprendam um determinado comportamento ou padrão automaticamente, a partir de exemplos e observações. A ideia por trás da aprendizagem é que percepções devem ser usadas não apenas para agir, mas também para melhorar a habilidade do agente para agir no futuro, RUSSEL e NORVIG (2009).

Muitas de nossas atividades diárias já funcionam a partir da utilização de algoritmos de Aprendizagem de Máquina. Dentre as atividades rotineiras mais comuns proporcionadas pelo uso desta tecnologia estão a categorização de e-mails, a filtragem de *SPAMs*, o reconhecimento ótico de caracteres, a organização de resultados de pesquisas na web, classificação de imagens, a pontuação de crédito e a indicação de melhores ofertas em sites, a detecção de invasão na rede, o uso de anúncios em tempo real em páginas da internet e dispositivos móveis, a análise de sentimentos baseada em textos, e ainda a detecção de fraudes e previsão de falhas em equipamentos.

A tecnologia de aprendizagem de máquinas pode ser categorizada como: supervisionada ou não supervisionada.

Aprendizagem supervisionada: nos é dado um conjunto de dados rotulados que já sabemos qual é a nossa saída correta e que deve ser semelhante ao conjunto, tendo a ideia de que existe uma relação entre a entrada e a saída. Problemas de aprendizagem supervisionados

são classificados em problemas de “regressão” e “classificação”. Em um problema de regressão, estamos tentando prever os resultados e uma saída contínua, o que significa que estamos tentando mapear variáveis de entrada para alguma função contínua. Em um problema de classificação, estamos tentando prever os resultados em uma saída discreta. Em outras palavras, estamos tentando mapear variáveis de entrada em categorias distintas.

Aprendizagem não supervisionada: nos permite abordar problemas com pouca ou nenhuma ideia do que nossos resultados deve ser aparentar. Podemos derivar estrutura de dados onde nós não necessariamente saberíamos o efeito das variáveis. Podemos derivar essa estrutura, agrupando os dados com base em relações entre as variáveis nos dados. Também pode ser usada para reduzir o número de dimensões e num conjunto de dados para concentrar somente nos atributos mais úteis, ou para detectar tendências. Com aprendizagem não supervisionada não há *feedback* com base nos resultados da previsão, ou seja, não há professor para corrigi-la.

2.4 Aprendizagem Profundo

Deep learning, ou aprendizagem profunda, é a parte do aprendizado de máquina que, por meio de algoritmos de alto nível, imita a rede neural do cérebro humano.

As redes neurais artificiais não são necessariamente novas, existem pelo menos desde a década de 1950. Mas durante várias décadas, embora a arquitetura desses modelos tivesse evoluído, ainda faltavam ingredientes que fizessem os modelos realmente funcionar. E esses ingredientes surgiram quase ao mesmo tempo. *Big Data*, volume de dados, gerado em variedade e velocidade cada vez maiores, permite criar modelos e atingir altos níveis de precisão. Mas ainda falta um ingrediente. Como processar grandes modelos de *Machine Learning* com grandes quantidades de dados? As CPUs não conseguiam dar conta do recado. Programação Paralela em GPUs. As unidades de processamento gráfico, permitem realizar operações matemáticas de forma paralela, principalmente operações com matrizes e vetores, elementos presentes em modelos de redes neurais artificias, formaram a tempestade perfeita, que permitiu a evolução na qual nos encontramos hoje: *Big Data* + Processamento Paralelo + Modelos de Aprendizagem de Máquina = Inteligência Artificial.

De forma simplificada, podemos dizer que *Deep Learning* são esses algoritmos complexos construídos a partir de um empilhamento de diversas camadas de “neurônios”, alimentados por quantidades imensas de dados, que são capazes de reconhecer imagens e fala,

processar a linguagem natural e aprender a realizar tarefas extremamente avançadas sem interferência humana.

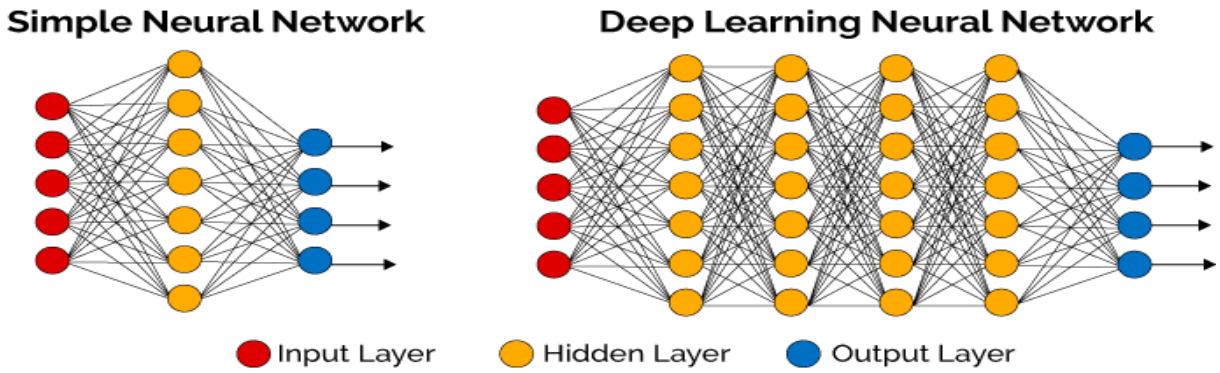


Figura 2- Representando Deep Learning

Deep Learning usa camadas de neurônios matemáticos para processar dados, compreender a fala humana e reconhecer objetos visualmente. A informação é passada através de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada. A primeira camada em uma rede é chamada de camada de entrada, enquanto a última é chamada de camada de saída. Todas as camadas entre as duas são referidas como camadas ocultas. Cada camada é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação. A principal aplicação dos algoritmos de *Deep Learning* são as tarefas de classificação, em especial, reconhecimento de imagens.

2.5 OpenFace

OpenFace é um modelo de reconhecimento facial de aprendizagem profunda desenvolvida por Brandon Amos, Bartosz Ludwiczuk e Mahadev Satyanarayanan. Ele é baseado no artigo: FaceNet: A Unified Embedding for Face Recognition and Clustering por Florian Schroff, Dmitry Kalenichenko, and James Philbin no Google. Foi implementado utilizando Python e Torch para que possa ser implementado em CPUs ou GPUs. Embora seja uma implementação recente, vem sendo bastante utilizado, pois oferece níveis de precisão semelhante aos modelos de reconhecimento facial encontrados em sistemas privados como FaceNet do Google, ou DeepFace do Facebook.

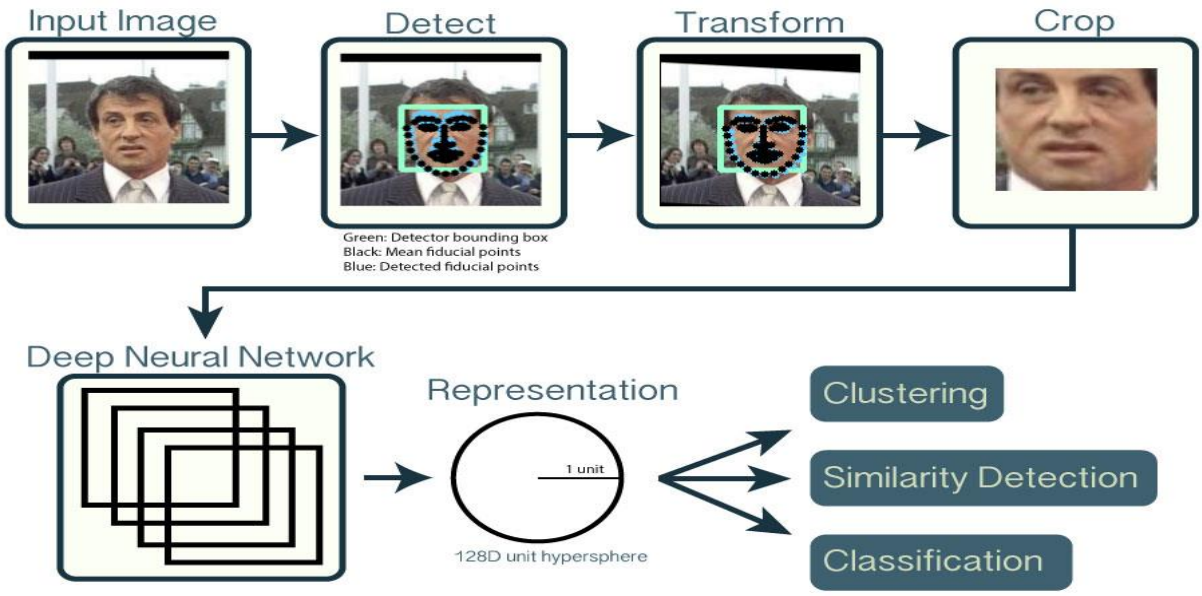


Figura 3- Representação OpenFace

De uma perspectiva de alto nível, o OpenFace usa o Torch, uma estrutura de computação científica para fazer treinamento *off-line*, o que significa que é feito apenas uma vez pelo OpenFace e o usuário não precisa ficar treinando centenas de milhares de imagens. Como é mostrado na Figura 3, após detectar e normalizar as faces, elas passadas para uma rede neural profunda para extração de recursos usando o modelo FaceNet do Google. Isso resulta na representação da face (incorporação) em uma esfera unitária de 128 dimensões faciais. Ao contrário de outras representações faciais, essa incorporação tem a boa propriedade de que uma distância maior entre duas incorporações de faces significa que as faces provavelmente não são da mesma pessoa. Essa propriedade torna as tarefas de *clustering*, detecção de similaridade e classificação mais fáceis do que outras técnicas de reconhecimento de face, nas quais a distância euclidiana entre recursos não é significativa. O FaceNet conta com uma função de perda de tripla para calcular a precisão da rede neural classificando uma face e é capaz de agrupar as faces por causa das medidas resultantes em uma hiperesfera.

O OpenFace é o primeiro kit de ferramentas capaz de detecção de ponto de referência facial, estimativa de posicionamento de cabeça, reconhecimento de unidade de ação facial e estimativa de olhar fixo com código-fonte disponível para execução e treinamento dos modelos. Os algoritmos de visão computacional que representam o núcleo do OpenFace demonstram resultados de última geração em todas as tarefas mencionadas acima. Além disso, nossa ferramenta é capaz de desempenho em tempo real e é capaz de rodar a partir de uma simples webcam sem qualquer hardware especializado.

2.6 Classificador K-Neareste Neighbors

O K-NN é um dos algoritmos de classificação mais utilizados na área de aprendizagem de máquina DINIZ (2013). É baseado na procura dos k vizinhos mais próximos do padrão de teste. A busca pela vizinhança é feita utilizando uma medida de distância nessa procura.

A implementação do algoritmo K-NN foi realizada com peso pela distância, e não pela frequência. Dessa maneira, o padrão é classificado de acordo com a soma dos pesos dos k vizinhos – o peso é o inverso das distâncias. O k escolhido será aquele que proporcionar menor quantidade de erros na classificação das imagens DINIZ (2013).

3 TRABALHOS RELACIONADOS

Este capítulo tem o intuito de discorrer em trabalhos que abordam o reconhecimento facial utilizando algoritmos de aprendizagem de máquina ou outros métodos a fim de chegar a esse objetivo.

No trabalho apresentado por Braga(2013), foi proposto um sistema de reconhecimento facial que recebe como entrada uma imagem e como saída retorna uma resposta positiva ou negativa no caso de haver ou não, na imagem faces previamente cadastrada em um banco de dados. O projeto consiste em três parte: detecção de faces, extração das características e reconhecimento. A detecção facial foi implementado usando algoritmo de Viola Jones, que tem como base o treinamento de classificadores fortes agregando-se diversos classificadores fracos. A extração de característica foi feita utilizando análise discriminante, que reduz a dimensionalidade do espaço e o reconhecimento foi feito utilizando métricas baseadas em distâncias entre projeções.

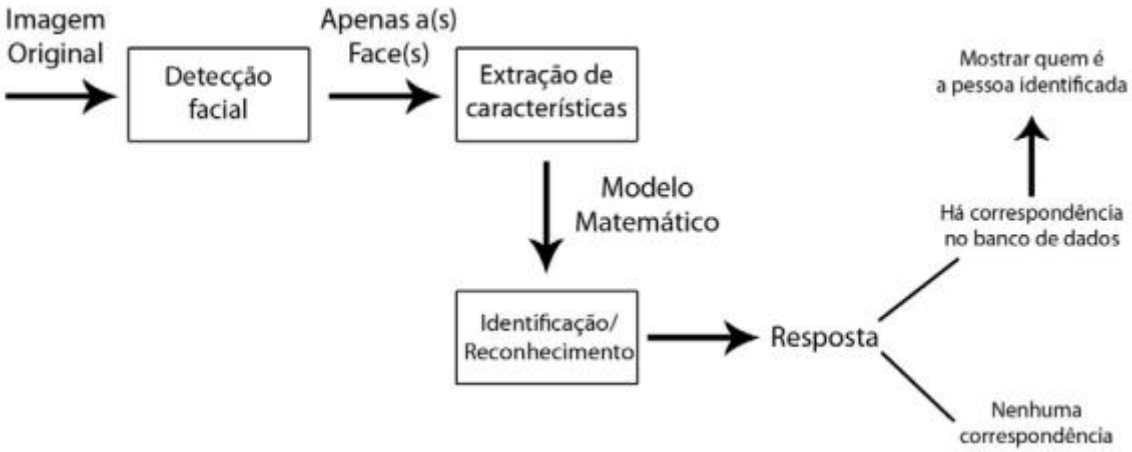


Figura 4- Arquitetura, trabalho relacionado

Na extração de características é utilizado o PCA (Análise de Componentes Principais), uma técnica matemática de redução de dimensionalidade. Essa técnica é utilizada para reconhecimento de padrões para eliminar redundâncias de informação. Diferente do PCA, ele utiliza o LDA (Análise discriminante linear) responsável por discriminar as classes de faces, essa técnica procura direções na qual a classes são melhores separadas. Para implementação do sistema de reconhecimento facial foi utilizado a plataforma Matlab, pois oferece uma vasta biblioteca de funções matemáticas e algoritmos numéricos e oferece boa portabilidade, podendo rodar tanto em Linux como Windows.

Resultado dos testes do sistema de reconhecimento facial:

Tabela 1- Resultados de trabalhos relacionados

Quantidade de pessoas cadastradas			**40**		
Quantidade de amostras			5		
Tempo processamento 200 faces			1,6s		
Tipos de Teste	Faces para teste	Threshold	Classificadas Corretamente	Falso Positivo	Falso Negativo
Teste1	200	0,9	130	1	69
Teste2	200	0,7	156	5	39

No artigo de GUSTAVO(2015) buscou uma alternativa mais avançada relacionado ao reconhecimento facial. O reconhecimento restritamente frontal como o caso de sistemas de segurança onde existe um ambiente controlado atinge acurácia a cima de 95%, com tudo, fatores como a iluminação, diferentes expressões e principalmente posições podem trazer dificuldades e diminuir a acurácia de algoritmos. Este artigo utilizando Redes Neurais buscou mostrar uma solução eficiente para este tipo de desafio. Para o desenvolvimento deste trabalho foi utilizado um código aberto chamado *OpenFace* desenvolvido pelo grupo da Google, no qual implementa o reconhecimento de face apresentado no artigo SCHROFF (2015). O trabalho foi elaborado em 4 *scripts*:

- ci-split: Dado como entrada um caminho para um pasta, onde contém diversas pastas que representam pessoas. Para cada pessoa, ou seja, para cada pasta contém um conjunto de imagens desta pessoa. Ao final do script obtém-se uma nova pasta com a divisão de conjunto de treino e o conjunto de testes;
- ci-create: Dado um caminho para uma pasta com um conjunto de treino e um conjunto de teste, converte-se as imagens de cada conjunto para a representação desejada (CNN com o modelo NN-2 da GoogLeNet ou com o descritor SURF ou SIFT). Por final a conversão de todas as imagens e guardada em dois arquivos, train.pkl e test.pkl;
- ci-train: Dado um caminho para uma pasta contendo o arquivo train.pkl, treina-se um SVM com parâmetros variados de C e Gamma, e por final escolhe-se o classificador com maior pontuação e o salva em classifier.pkl;
- ci-test: Dado um caminho para uma pasta contendo o arquivo classifier.pkl e test.pkl, carrega-se o SVM e prediz todas as representações contidas no arquivo test.pkl e ao final do script devolve a acurácia obtida;

4 FERRAMENTAS E TECNOLOGIAS

4.1 OPENCV

OpenCV é a principal biblioteca de código aberto para se trabalhar com visão computacional, processamento de imagem e aprendizagem de máquina. Os algoritmos apresentados neste trabalho, utilizam métodos desta biblioteca para análise de imagens e aprendizagem de máquina.

OpenCV (*Open Source Computer Vision Library*) é lançado sob uma licença BSD e, portanto, é gratuito para uso acadêmico e comercial. Possui interfaces C ++, Python e Java e suporta Windows, Linux, Mac OS, iOS e Android. O OpenCV foi projetado para eficiência computacional e com um forte foco em aplicativos em tempo real. Escrito em C / C ++ otimizado, a biblioteca pode aproveitar o processamento de vários núcleos. Habilitado com o OpenCL, ele pode aproveitar a aceleração de hardware da plataforma de computação heterogênea subjacente.

Adotado em todo o mundo, o OpenCV tem mais de 47 mil pessoas da comunidade de usuários e um número estimado de downloads que ultrapassam 14 milhões. O uso varia de arte interativa a inspeção de minas, costura de mapas na web ou através de robótica avançada.

4.2 Python

Python é uma linguagem de programação poderosa e fácil de aprender. Ele possui estruturas de dados eficientes de alto nível e uma abordagem simples, mas eficaz, para programação orientada a objetos. A sintaxe elegante e a tipagem dinâmica do Python, junto com sua natureza interpretada, fazem dele uma linguagem ideal para scripts e desenvolvimento rápido de aplicativos em muitas áreas na maioria das plataformas. Alguns exemplos de usos diversos da linguagem são: *back-end* de sistemas web, ciência de dados, *machine learning*, simulações. A criação dos serviços de persistência de dados e processamento de imagens será feita através dessa ferramenta.

4.3 Angular 2x

Angular é uma plataforma e *framework* utilizado para construção da interface de aplicações web dinâmicas, usando HTML, CSS e, principalmente, JavaScript, criada pelos desenvolvedores da Google. O módulo Web desenvolvido neste trabalho será utilizando esta tecnologia, possibilitando realizar uma interação entre o usuário e toda lógica de negócio de forma mais agradável.

O Angular é todo baseado em componentes, sendo partes/estruturas de códigos modulares, que possuem e geram em si suas próprias regras de negócio.

A arquitetura de um projeto Angular é similar ao conhecido MVC (*Model View Controller*), chamado MVVM (*Model View View-Model*):

- *View*: O que o usuário vê;
- *View-Model*: componente, HTML e templates, fazendo ponte entre o que vem do banco e aquilo que deve ser exposto ao cliente;
- *Model:* Se encarrega de toda lógica da aplicação.

Utilizando o conceito de SPA (*Single Page Aplication*), o Angular apresenta a mudança de componentes sem o *loading* da página, trazendo uma experiência muito fluida aos sistemas que o usam. O projeto foi originalmente desenvolvido por Miško Hevery, em 2009, através da empresa Brat Tech LLC que, após um breve lançamento, optou por difundi-lo enquanto software livre (2010).

Sua principal característica que o diferencia dos outros frameworks é que ele pode funcionar como uma extensão do HTML e não é necessário manipular o DOM, ainda que o Angular possibilite isso. O que permite a criação de um *front-end* mais elaborado, sem precisar manusear diretamente o HTML e os dados. Para o desenvolvimento da aplicação web deste projeto, foi utilizado a última versão (4.0).

4.4 Representational State Transfer (REST)

Representational State Transfer, abreviado como REST, não é uma tecnologia, uma biblioteca, e nem tampouco uma arquitetura, mas sim um modelo a ser utilizado para se projetar arquiteturas de software distribuído, baseadas em comunicação via rede. REST é um dos modelos de arquitetura que foi descrito por Roy Fielding, um dos principais criadores do protocolo HTTP, em sua tese de doutorado e que foi adotado como o modelo a ser utilizado na evolução da arquitetura do protocolo HTTP.

Figura 5- Arquitetura REST

Conforme a Figura 5, este modelo será utilizado para a implementação dos Serviços Web deste trabalho, como: mostrar e cadastrar moradores e enviar uma requisição ao servidor uma imagem de face a fim de que ele retorne se essa imagem é de um morador ou não, gerenciando os recurso do módulo web.

4.5 UML

A UML é uma linguagem usada para modelar sistemas orientados a objeto. É voltada para a visualização, especificação, construção e documentação de artefatos (BOOCH, et al., 2006). A importância de modelar um sistema antes do seu desenvolvimento pode trazer benefícios como uma melhor comunicação entre os envolvidos na implementação do projeto, uma melhor compreensão e análise do sistema, torna a programação e a manutenção do código mais fácil, e diminui a porcentagem de erros.

Foram utilizados neste trabalho os seguintes diagramas da UML: Diagrama de Caso de Uso e Diagrama de Classes.

4.6 Bootstrap

Desenvolvido por Jacob Thorton e Mark Otto, engenheiros do Twitter, como uma tentativa de resolver incompatibilidades dentro da própria equipe. O intuito era otimizar o desenvolvimento de sua plataforma através da adoção de uma estrutura única. Isto reduziria inconsistências entre as diversas formas de se codificar, que variam de profissional para profissional. E a tentativa deu tão certo que eles perceberam o grande potencial da ferramenta, lançando-a no GitHub como um software livre.

Bootstrap é um framework *front-end* que facilita a vida de desenvolvedores na criação de aplicações web responsiva, *renderizando* em várias telas de tamanhos diferentes, sem quebrar seus componentes. Além disso, o Bootstrap possui uma diversidade de componentes

(plugins) em JavaScript (jQuery) que auxiliam o designer a implementar: tootlip, menu-dropdown, modal, carousel, slideshow, entre outros, otimizando o tempo de desenvolvimento.

4.7 MySQL

MySQL é um sistema de gerenciamento de banco de dados (SGBD). É um dos SGBD's mais populares do mundo, sendo utilizado em grandes empresas como, NASA, HP, Nokia, Sony, U.S. Federal Reserve Bank, Associated Press, Alcatel, Cisco Systems e Google.

O MySQL possui uma lista de características que o fizeram se tornar um dos SGBD's mais utilizados em todo mundo. Algumas dessas características são: portabilidade, compatibilidade, excelente desempenho e estabilidade, exige poucos recursos de hardware, é um *software* livre, suporta *triggers, cursors, stored procedures*, *functions* e possui interface gráfica (MYSQL, 2017).

Optou-se pela utilização do MySQL neste trabalho pelo fato de ser um SGBD gratuito, além de possuir todas as outras características mencionadas no parágrafo acima.

4.8 Java – Spring Boot

Neste trabalho será utilizado a linguagem de programação Java com o Framework Spring Boot. O Spring cuida da infraestrutura do projeto, deixando ao desenvolvedor somente a preocupação com as regras de negócio da aplicação. O modelo Spring Boot, busca solucionar a complexidade da inicialização e gerenciamento de dependências de um projeto com Spring.

Essencialmente, o Spring Boot pode ser considerado um *plugin* para a ferramenta de *building*, seja ela o Maven ou o Gradle. Seus principais objetivos são gerenciar dependências de maneira opinativa e automática, e simplificar a execução do projeto em tempo de desenvolvimento e depuração.

O Spring Boot também possui a funcionalidade de empacotamento da sua aplicação em um JAR executável contendo todas as dependências necessárias, inclusive o *Servlet Container*, seja ele o Tomcat, Jetty ou mesmo Undertow, apesar de ainda ser possível empacotar um WAR da forma tradicional.

O principal benefício do Boot, entretanto, é a configuração de recursos baseada no que se encontra no *classpath*. Se o POM de seu Maven inclui a dependência do JPA e o driver do MySQL, ele irá criar uma unidade de persistência baseada no MySQL.

Se adicionarmos alguma dependência web, iremos perceber que o Spring MVC assumirá configurações *default* e dependências com relação a diversos aspectos, como a

tecnologia de apresentação (o *default* é o Thymeleaf), o mapeamento de recursos e *marshalling* de JSON (o *default* é o Jackson) e/ou XML (o *default* é o JAXB 2) para o tratamento de dados de requisição e resposta, que necessitamos em uma aplicação REST, por exemplo.

4.9 Apache Kafka

O Apache Kafka foi originalmente desenvolvido pelo LinkedIn e posteriormente liberado como um projeto *open-source*, em 2011. O Apache Kafka é um sistema para gerenciamento de fluxos de dados em tempo real, gerados a partir de web sites, aplicações e sensores. Algumas das empresas que usam Kafka: LinkedIn, Netflix, PayPal, Spotify, Uber, AirBnB, Cisco, Goldman Sachs e SalesForce. Recentemente a IBM anunciou a criação de dois projetos envolvendo o Apache Kafka. O LinkedIn possui o maior ambiente Kafka do qual se tem notícia, com 1.1 trilhões de mensagens por dia.

Essencialmente, o Kafka age como uma espécie de "sistema nervoso central", que coleta dados de alto volume como por exemplo a atividade de usuários (clicks em um web site), logs, cotações de ações, dentre outras, e torna estes dados disponíveis como um fluxo em tempo real para o consumo por outras aplicações. O Kafka vem ganhando cada vez mais popularidade em Big Data pois além de ser um projeto *open-source* de alta qualidade, possui a capacidade de lidar com fluxos de alta velocidade de dados, característica cada vez mais procurada para uso em Internet das Coisas, por exemplo.

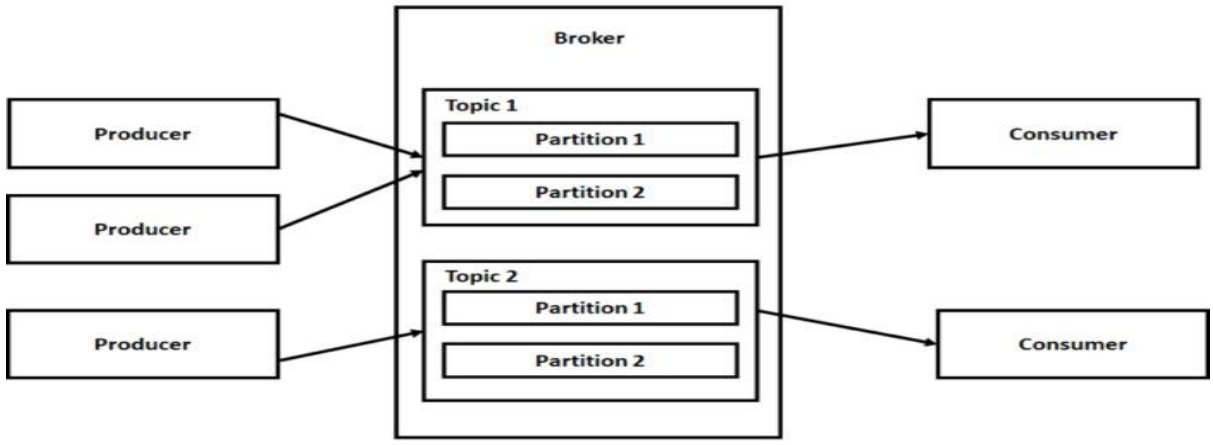


Figura 6- Arquitetura básica Apache Kafka

No Apache Kafka, toda a interação entre produtores e consumidores é realizada através de tópicos. Tópicos representam fluxos de eventos e desacoplam a comunicação entre sistemas, uma vez que o produtor e o consumidor das mensagens não se conhecem.

Quando um produtor gera uma mensagem no tópico, os consumidores daquele tópico são notificados com uma cópia da mensagem publicada. Não há limites para a quantidade de consumidores que um tópico pode possuir.

Internamente, o Kafka quebra os tópicos em partições. O número de partições é indicado quando o tópico é criado e não há limite de partições. Para facilitar a alta disponibilidade, as partições de um tópico são espalhadas entre os *brokers* do cluster. Abaixo, um diagrama explicando essa separação em partições de um tópico:

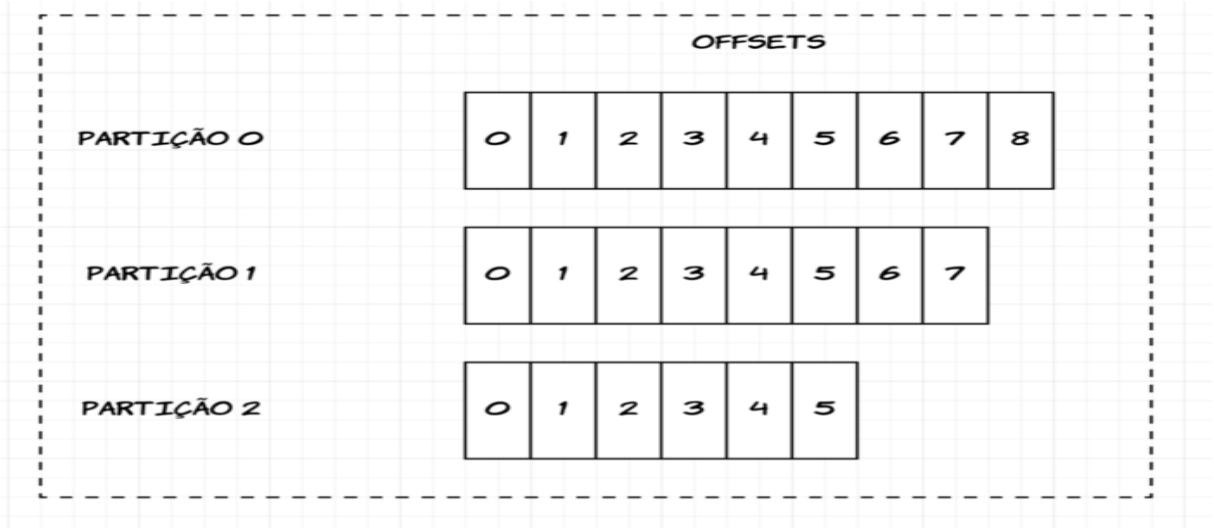


Figura 7- Patitions Apache Kafka

Como pode ser visto na imagem anterior, o tópico possui 23 mensagens separadas em 3 partições. Dentro de cada partição, as mensagens são ordenadas por um metadado chamado *offset*. O offset serve para ordenar apenas no contexto da partição. *Offsets* iguais em partições diferentes resultam em mensagens diferentes.

4.10 Dlib

O Dlib é um kit de ferramentas C ++ moderno que contém algoritmos e ferramentas de aprendizado de máquina para criar softwares complexos em C ++ para resolver problemas do mundo real. Ele é usado tanto na indústria quanto na academia em uma ampla gama de domínios, incluindo robótica, dispositivos incorporados, telefones celulares e grandes ambientes de computação de alto desempenho. O licenciamento de código aberto da Dlib permite que você o use em qualquer aplicativo, gratuitamente.

5 PROPOSTA DE SOLUÇÃO

5.1 Introdução

Neste capítulo, serão apresentadas as etapas propostas para o desenvolvimento do módulo de reconhecimento facial aplicado a um sistema de controle de acesso a condomínios.

5.2 Estrutura da Proposta

Neste trabalho foi proposto uma aplicação web onde o usuário possa cadastrar novos moradores capturando fotos e coletando outras informações como: nome, apartamento, email, telefone e data de nascimento, a fim de que esses dados fiquem armazenados em um banco de dados e disponível para consultas. Outra funcionalidade proposta para este trabalho é a de reconhecimento facial, responsável por auxiliar o profissional da portaria se a pessoa que está na entrada do prédio é um morador, ou seja, tem um cadastro realizado na base de dados, ou se é um desconhecido.

5.3 Modelagem do Sistema

Esta seção irá mostrar os diagramas de casos de uso e o diagrama de classes que foram confeccionados durante a fase de modelagem do sistema para orientar o desenvolvimento da aplicação proposta.

5.3.1 Diagrama de caso de uso

O diagrama de casos de uso é a visão do sistema, a qual evidencia suas funcionalidades e seus principais atores. O sistema aqui proposto apresenta um único ator, Usuário, que será responsável por cadastrar moradores do condomínio e identificá-los, a fim de permitir ou não o acesso ao condomínio. As descrições dos casos de usos encontram-se no Apêndice A desse documento.

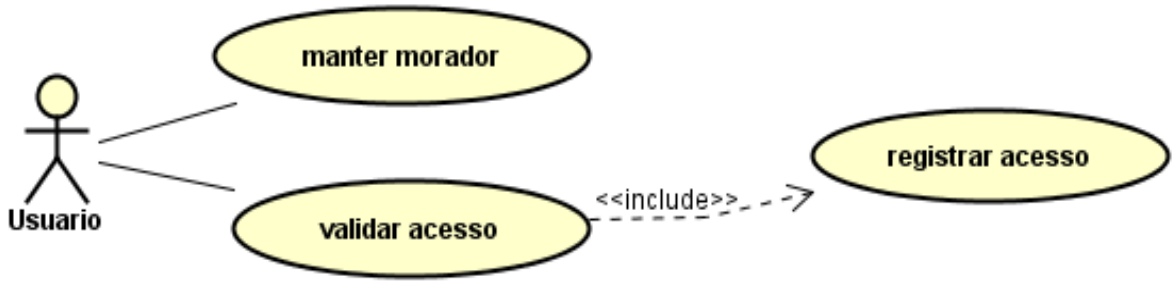


Figura 8- Diagrama de Caso de Uso

5.3.2 Modelagem do banco de dados

A Figura 9 apresenta os tipos de dados armazenados e forma de como estão relacionados.

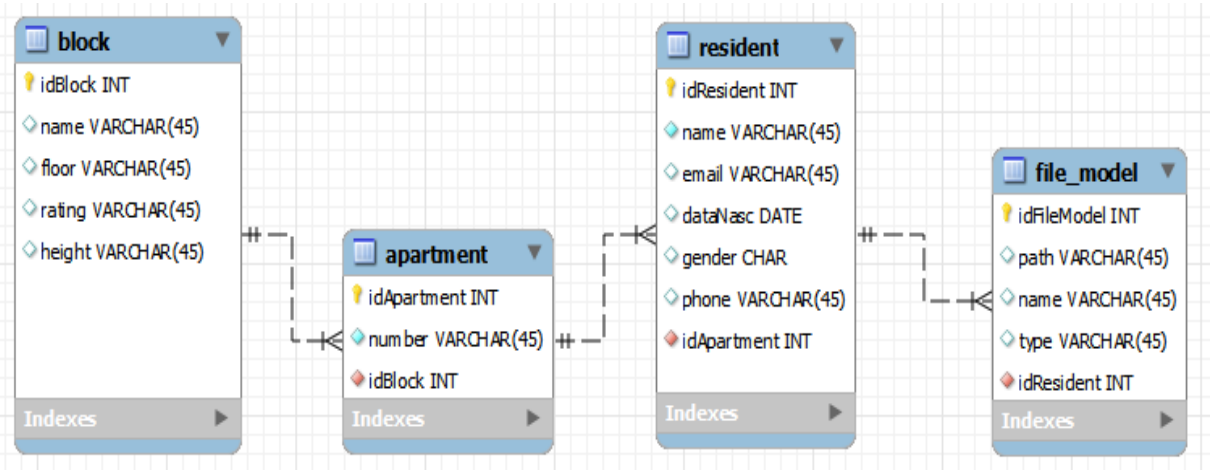


Figura 9- Modelagem do banco de dados

5.3.3 Diagrama de classe

Uma classe é a descrição de um conceito decorrente do domínio da aplicação ou da solução da aplicação. Com base nisso foi modelado um Diagrama de Classes conforme é representado no Figura 10. Onde existe a classe *block* que possui uma lista de *apartments* que por sua vez possui uma lista de residentes. Cada *resident* possui uma galeria contendo pelo menos uma foto para o cadastro. Para trabalhos futuros podem ser criado novas classes responsáveis por registrar a entrada e saída do morador no condomínio.

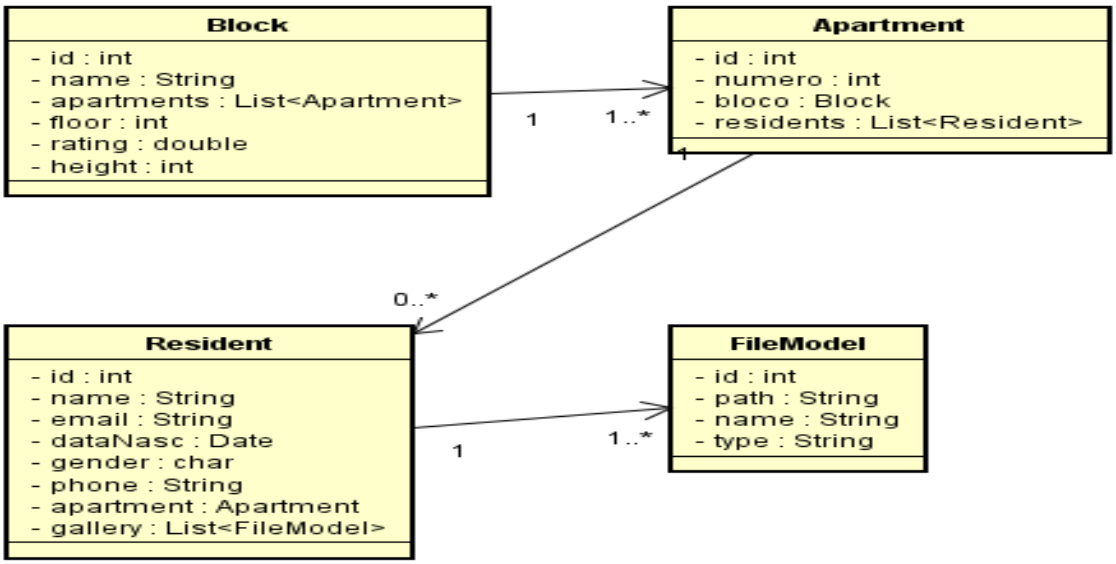


Figura 10- Diagrama de Classe

6 IMPLEMENTAÇÃO DA PROPOSTA

Neste capítulo será apresentado o detalhamento sobre a implementação do sistema proposto neste trabalho.

6.1 Telas da Aplicação

A Figura 11 apresenta a tela inicial do sistema desenvolvido. Nesta tela, vamos dar destaque para as funcionalidades associadas às abas do Menu: “Blocos”, “Cadastro” e “Monitoramento”.

Figura 11- Tela inicial

Na opção “Blocos” do menu é apresentada a tela, representada pela Figura 12, na qual é exibida uma lista dos blocos do condomínio cadastrados no sistema juntamente com suas respectivas informações.

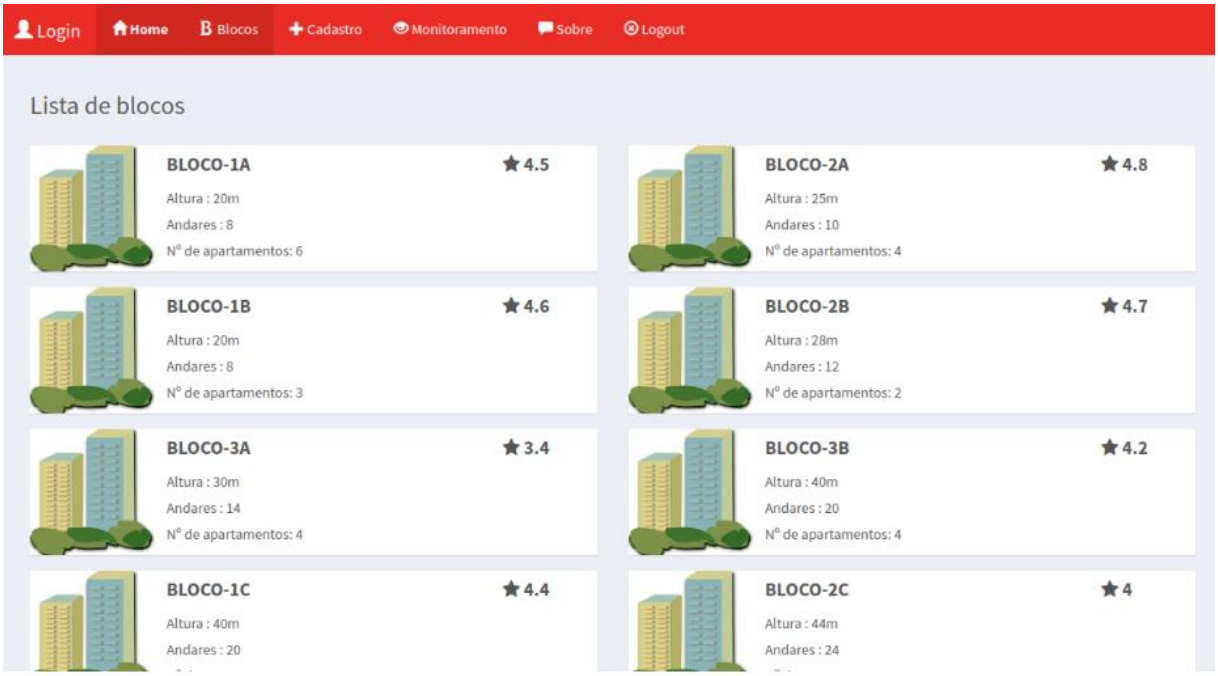


Figura 12- Tela de listagem de blocos

Quando algum dos blocos é selecionado, o usuário receberá uma tela de listagem de apartamentos, contendo informações sobre a quantidade de moradores daquele apartamento, conforme apresentado na Figura 13.

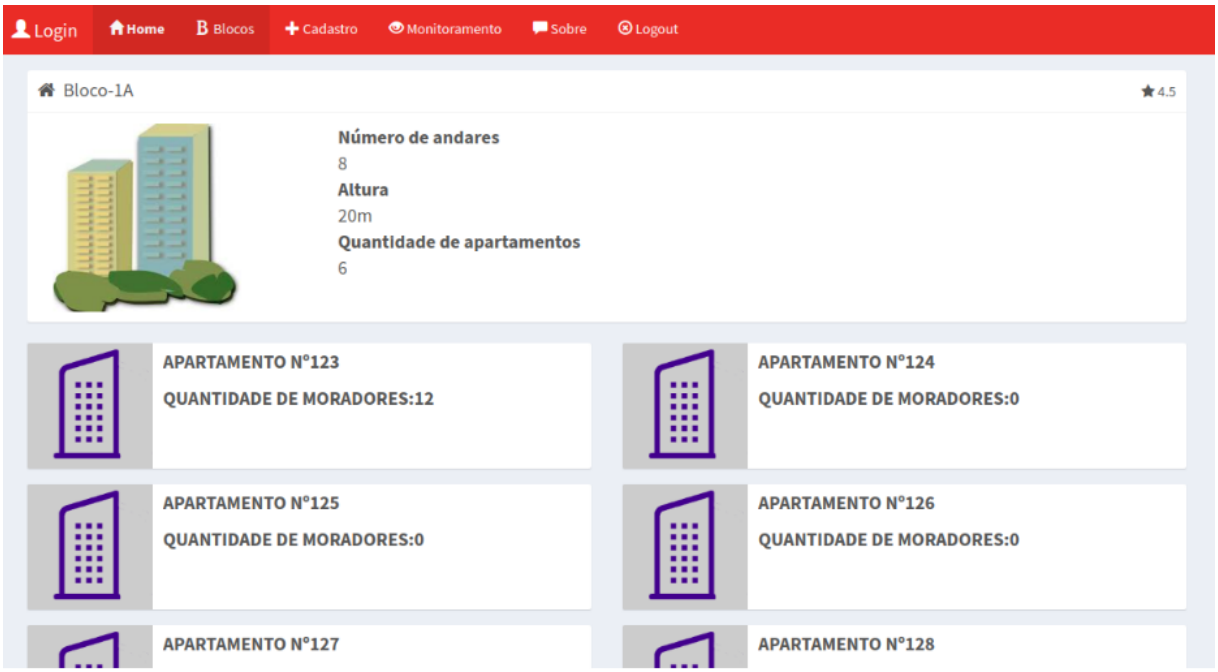


Figura 13- Tela de listagem de apartamentos

Ao selecionar um dos apartamentos da listagem, é possível ver as informações dos moradores desse apartamento, conforme apresentado na Figura 14.

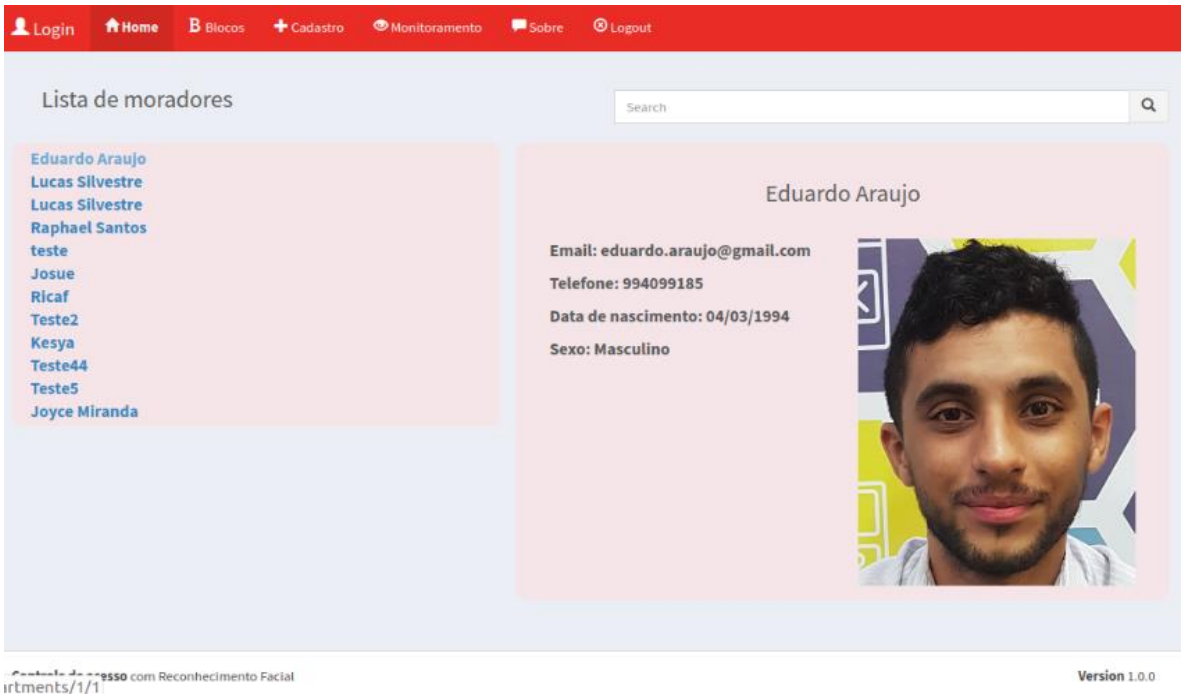


Figura 14- Informações do morador I

Acessando a aba “Cadastro” no menu superior, é possível visualizar a tela de cadastro para novos moradores, conforme apresentado na Figura 15.

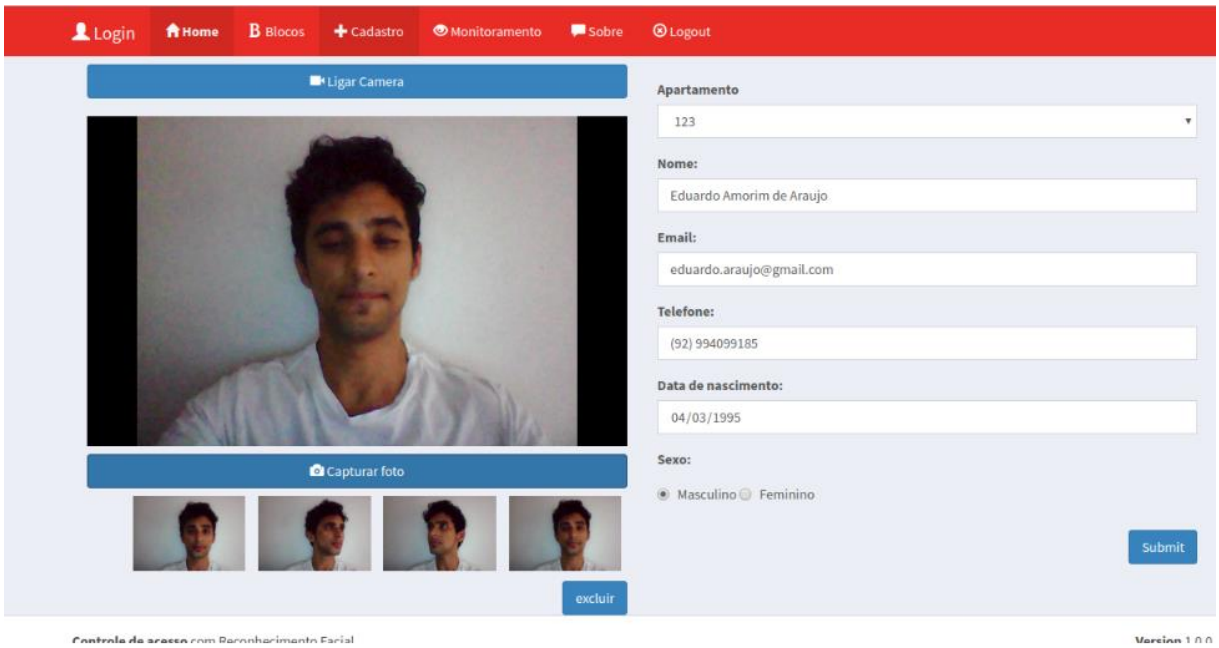


Figura 15- Tela de Cadastro

Na aba de “Monitoramento” é possível visualizar a detecção da face e o retorno do resultado do processamento de reconhecimento facial rotulado na tela, conforme apresentado na Figura 16. Esta tela externa a integração da webcam com a aplicação de visão computacional.

Figura 16- Tela de interação do módulo de visão com webcam

As Figuras 17 e 18 mostram o resultado do processamento do módulo de reconhecimento, demonstrando uma tela onde houve o reconhecimento da pessoa e outra onde não houve o reconhecimento.

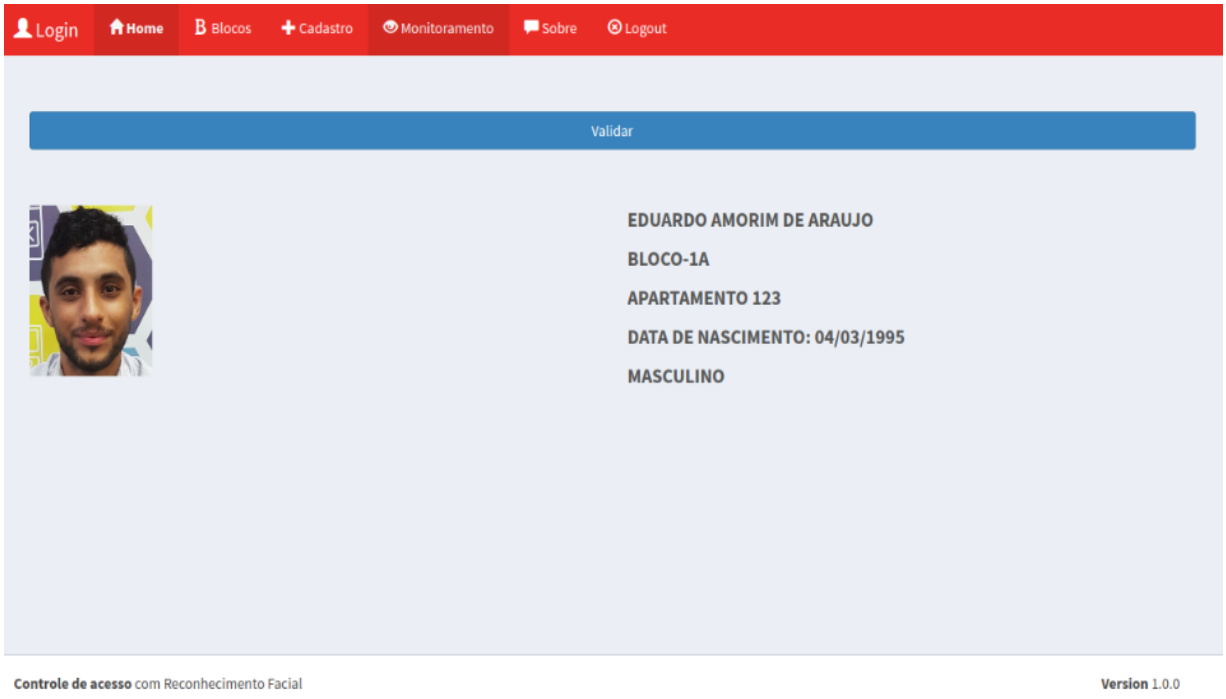


Figura 17- Tela Com Validação

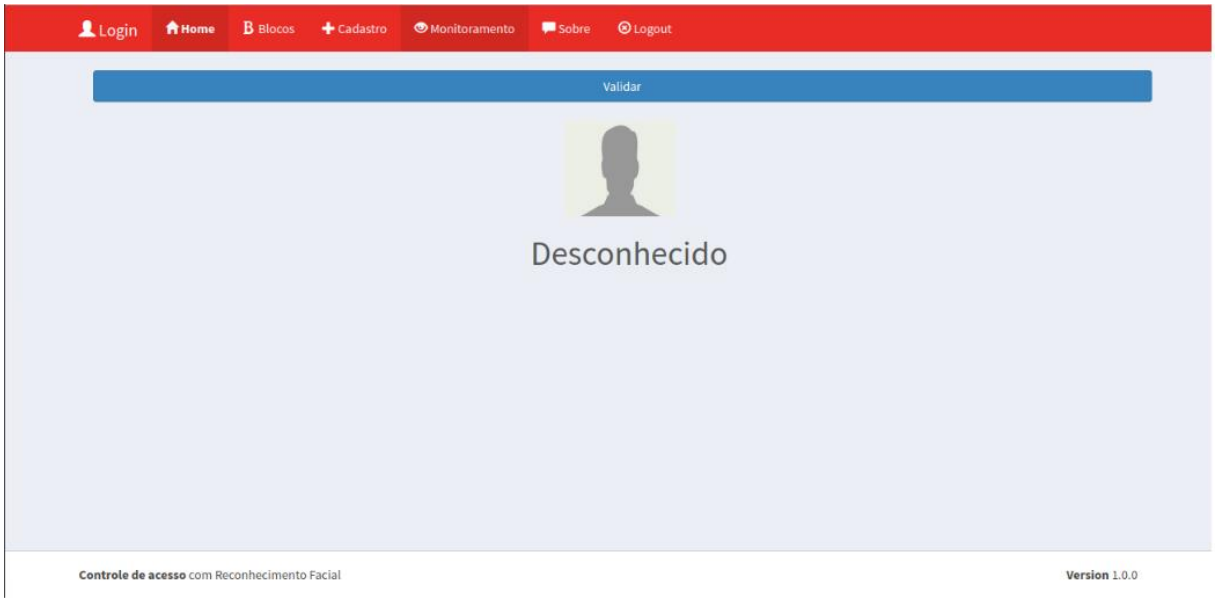


Figura 18- Tela sem validação

6.2 Arquitetura da Aplicação

A arquitetura do sistema, foi dividida em 4 módulos principais:

- Aplicação Front-End;
- Servidor de mensageria.
- Aplicação Back-end;
- Visão computacional;

6.2.1 Aplicação Front-End

O módulo web da aplicação é responsável pela interação com o usuário. Esta parte da aplicação foi implementada com o Angular 4.0, um framework JavaScript, baseado em componentes que utiliza o modelo de SPA (*Single Page Application*), fazendo com que todo o conteúdo da página seja criado com um único carregamento, tornando a interação do usuário com a aplicação mais fluida. Também foi utilizado o Framework Bootstrap, para deixar a aplicação visualmente padronizada e responsiva.

6.2.2 Servidor de mensageria

Este módulo utiliza o Servidor Apache Kafka para prover a comunicação entre os módulos da visão computacional e a aplicação *back-end*, por meio da implementação de tópicos. Para que o compartilhamento dos resultados do módulo de reconhecimento facial esteja disponível e acessível para aplicação de *back-end* consumir, o módulo de mensageria do Kafka

cria um tópico somente para retornar essa informação e quem estiver observando esse tópico, poderá receber a informação aguardada de forma instantânea. A arquitetura do Kafka utilizada na aplicação, pode ser visualizada na Figura 19.

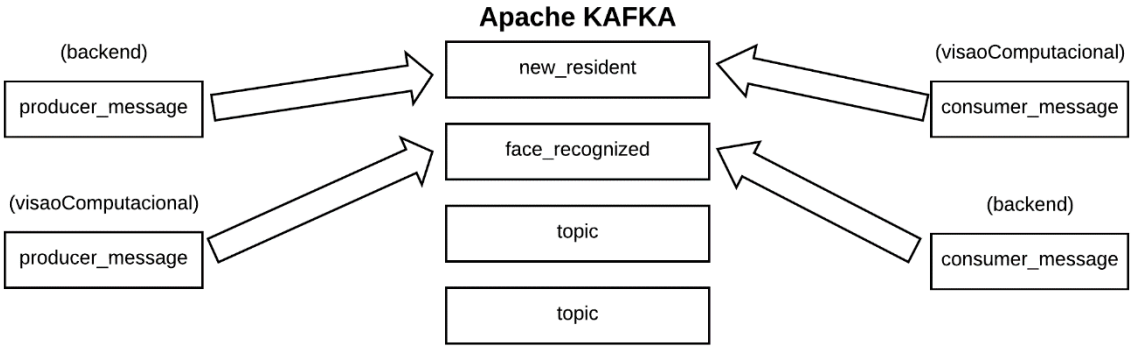


Figura 19- Arquitetura Kafka

O sistema desenvolvido neste trabalho utiliza dois tópicos implementados no Servidor Apache Kafka: "*new_resident*" e "*face_recognized"*. O módulo *back-end*, ao salvar um novo morador, envia uma mensagem para o tópico "*new_resident*". O módulo de visão computacional responsável por gerar os *encondings* de cada morador está observando o tópico de "*new_resident*", quando chega uma nova mensagem, esse serviço é executado.

O módulo de visão computacional, enquanto está realizando o processamento de reconhecimento facial, estará enviando uma mensagem para o tópico "*face_recognized*" a todo momento. Esta mensagem irá retornar o *id* da pessoa, caso a pessoa seja conhecida e valor *null* caso a pessoa não seja reconhecida.

6.2.1 Aplicação Back-end

Esse módulo é responsável pela implementação das regras de negócio, comunicação com o banco de dados e a persistência das informações.

A Figura 20 apresenta o módulo de cadastro de novos moradores. O cadastro é realizado através de uma aplicação web que se conecta a uma webcam, onde devem ser capturadas uma sequência de imagens de cada morador, gerando assim um diretório de fotos (*Dataset*) pra cada morador em um servidor de arquivos. Em seguida, os dados preenchidos no formulário, junto com as imagens, são enviados da aplicação *front-end* para a aplicação *back-end* desenvolvida em Java, via API REST.

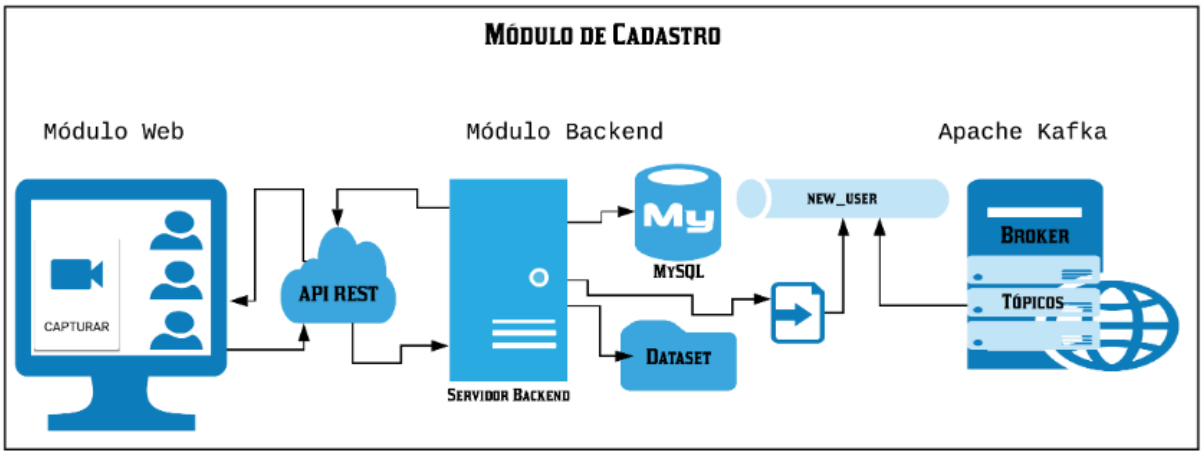


Figura 20- Arquitetura geral do módulo de cadastro

A aplicação *back-end* recebe as informações enviadas pela aplicação *front-end* e persiste as informações no banco de dados MySQL e as imagens são enviadas para um servidor de arquivos para compor o *Dataset*, conforme representado na Figura 20.

Quando o módulo de serviços *back-end* realiza um novo cadastro, é emitida uma mensagem para o Tópico do *Broker* chamado "*new_resident*", informando ao servidor de mensageria Kafka que houve um novo cadastro (Figura 21). No módulo de visão computacional, o serviço que está observando o tópico "*new_resident*" é acionado. O módulo de visão computacional, utilizando técnicas de aprendizagem profunda, é responsável por construir a incorporação das 128 dimensões da face e adicionar esse vetor de característica a um arquivo Python chamado *encodings* com extensão *pickle* (*encodings.pickle*).

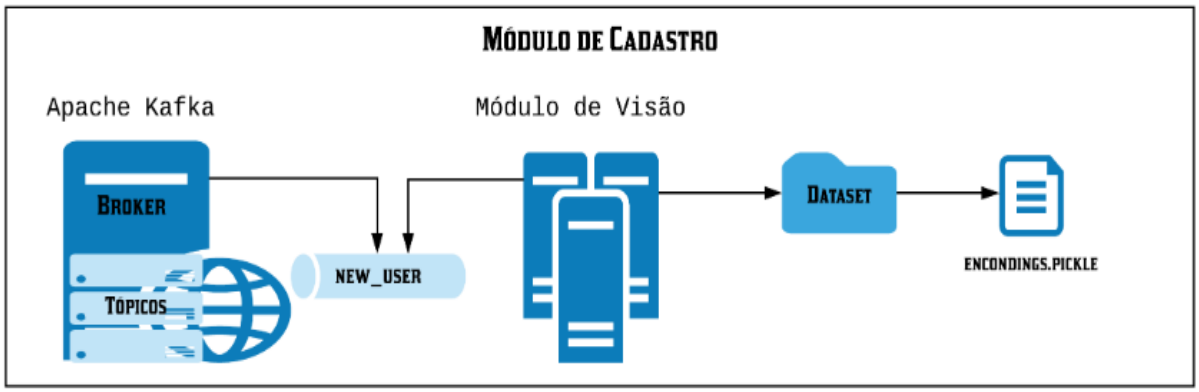


Figura 21- Arquitetura geral do módulo de cadasto II

Uma das principais funcionalidades deste módulo é a criação do *Dataset* (diretório com os arquivos de imagens dos usuários cadastrados) a ser utilizado pela aplicação de visão computacional e a implementação dos serviços/*endpoints*, para que a aplicação web consiga

consumir seus dados, utilizando API REST. A Tabela 2 apresenta os principais *endpoints* utilizados na aplicação back-end.

Tabela 2- Tabela de endpoints

Repositório	ApartmentREST	ResidentREST	BlockREST	FileModelREST
URL **REQUEST** **METHOD**	**/rest/apartments** **GET** **findAll()**	/rest/resident/ap/{id} GET findByApartment()	/rest/block GET findAll()	/rest/file/{name} POST upload()
URL **REQUEST** **METHOD**	**/rest/apartments** **POST** **save()**	/rest/resident/ POST save()	/rest/block POST save()	/rest/file/uploadMultipleFiles/{name} POST uploadMultipleFiles()
URL **REQUEST** **METHOD**	**/rest/apartments/block/{id}** **GET** **listApartmentByBlock()**	/rest/resident/{id} GET findOne()	/rest/block/{id} GET findOne()	/rest/file/{id} GET downloadFile()

Na Figura 22 é apresentada a classe *ResidentREST* contendo os seguintes serviços: método *findAll* (retorna uma lista dos moradores cadastrados no banco de dados), método *save* (persisti as informações do morador no banco de dados), *findOne* (recebe um id do módulo web, faz uma consulta no banco de dados e retornar os dados do morador daquele id, caso esteja cadastrado, se não estiver, é retornado uma mensagem informando que não há registro com aquele id), *delete* (deleta o morador com aquele id).

```java
@RestController
@RequestMapping(value = "/rest/residents")
public class ResidentREST {

    @Autowired
    private ResidentService residentService;

    @GetMapping
    public List<Resident> findAll(){
        return residentService.findAll();
    }

    @PostMapping
    public ResponseEntity<Resident> save(@RequestBody Resident resident){
        Apartment ap = apartmentService.findOne(resident.getApartment().getId());
        resident.setApartment(ap);
        return  new ResponseEntity<>(residentService.save(resident), HttpStatus.CREATED);
    }

    @RequestMapping(method = RequestMethod.GET, path = "/{id}")
    public ResponseEntity<Resident> findOne(@PathVariable Long id){
        Resident b = residentService.findOne(id);
        return b !=  null ? ResponseEntity.ok(b) : ResponseEntity.notFound().build();
    }

    @RequestMapping(method = RequestMethod.DELETE, path = "/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void delete(@PathVariable Long id){
        residentService.delete(id);
    }

```

Figura 22- Exemplo endpoints

É necessário que a aplicação *back-end* receba a resposta do módulo de reconhecimento facial e envie uma mensagem para o módulo de visão computacional, sempre que existir um novo cadastro. O módulo intermediário que irá estabelecer essa comunicação é o Apache Kafka, mencionado na seção 6.2.2.

A Figura 23 mostra as classes necessária para que a aplicação *back-end* consiga se comunicar com o servidor de mensageria apache Kafka. A classe OrderConsumer é responsável é utilizada para consumir as informações geradas pelo tópico *face-recognized*, o resultado do módulo de reconhecimento facial. A classe OrderProducer é encarregada de enviar uma mensagem para o tópico *new-register*, esse método é chamado sempre que é adicionado um novo morador. A classe OrderProducer realiza o envio das mensagens para o tópico do Apache Kafka.

```java
@Component
@Slf4j
public class OrderConsumer {
    @KafkaListener(topics = "${order.face-recognized}")
    public void consumer(String order) {
        log.info("Order: " + order);
    }
}
@Component
public class OrderProducer {

    @Value("${order.new-register}")
    private String orderTopic;
    private final KafkaTemplate kafkaTemplate;
    public OrderProducer(final KafkaTemplate kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }
    public void send(final @RequestBody String order) {
        kafkaTemplate.send(orderTopic, order);
    }
}
@RestController
@RequestMapping(value = "/orders")
@Slf4j
public class OrderController {
    private final OrderProducer orderProducer;
    public OrderController(OrderProducer orderProducer) {
        this.orderProducer = orderProducer;
    }
    @RequestMapping(method = RequestMethod.POST)
    public void send(@RequestBody String order) {
        orderProducer.send(order);
    }
}
```

Figura 23- Código *back-end* acessando Apache Kafka

A aplicação *back-end* também é responsável por criar um diretório no servidor de arquivos para cada novo morador. O nome do diretório é o id identificador deste morador para associar aquela pasta a sua pessoa. A Figura 24 representa a estrutura do *Dataset*.

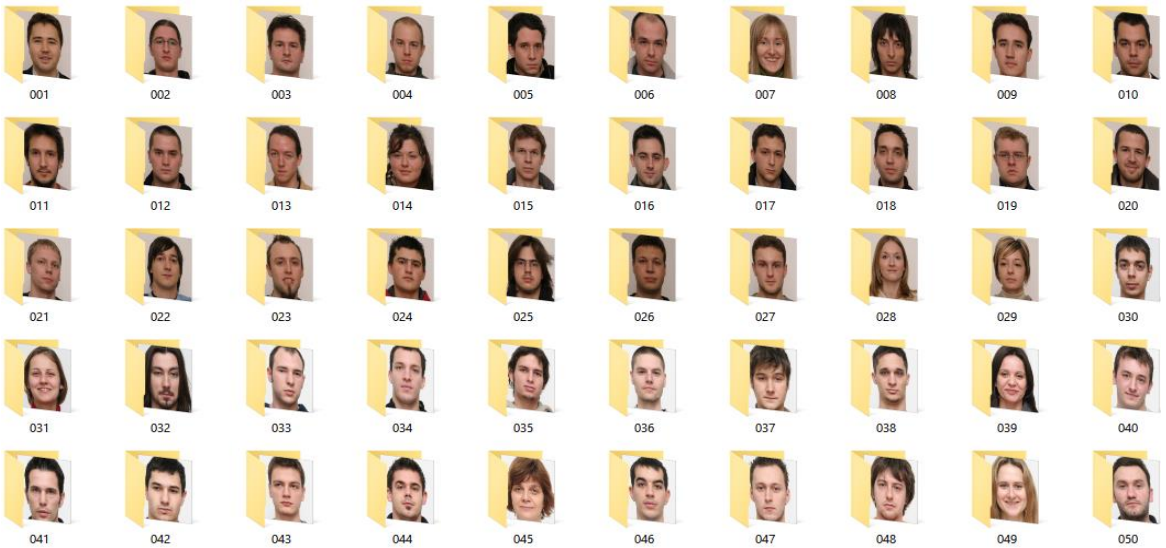


Figura 24- Exemplo Dataset

6.2.4 Módulo de Visão computacional

O módulo de visão computacional é composto por dois serviços: Geração de *Encodings* e Reconhecimento Facial. A Figura 25 representa o processo de treinamento utilizado na rede neural profunda para gerar a extração a incorporação de 128 bytes para cada face, implementado pela rede neural FaceNet implementada pela biblioteca OpenFace, conforme explicado na fundamentação teórica.

O treinamento é feito com três imagens diferentes. A primeira é uma imagem conhecida, chamada como âncora, em seguida, outra imagem da mesma pessoa é utilizada (inserção positiva), enquanto a última é uma imagem de uma pessoa diferente (inserção negativa). Em seguida, o algoritmo analisa as medições que está gerando atualmente para cada uma dessas três imagens. Ele ajusta os pesos na rede neural, para garantir que as medições sejam geradas entre a âncora e a segunda imagem da pessoa conhecida, para que estejam mais próximas, garantindo que as medidas da segunda imagem conhecida e a terceira imagem do peso desconhecido, fique mais distante.

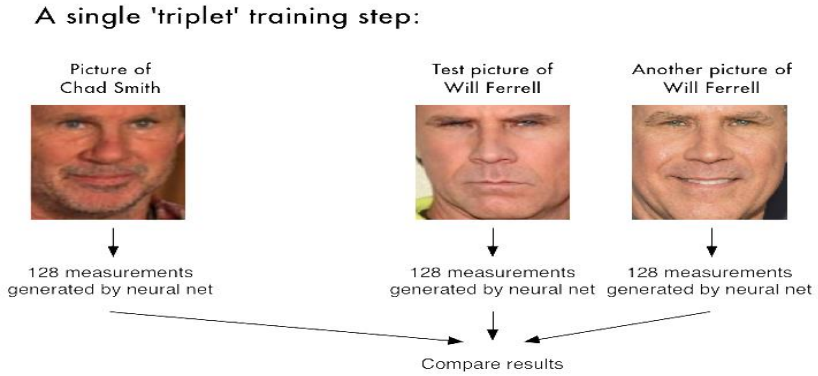


Figura 25- Técnica utilizada no treinamento da rede

Depois de repetir essa etapa milhões de vezes para milhões de imagens de milhares de pessoas diferentes, a rede neural aprende a gerar, com segurança, 128 medições para cada pessoa. Quaisquer dez imagens diferentes da mesma pessoa devem dar aproximadamente as mesmas medidas.

Para diferenciar uma pessoa da outra, é necessário extrair algumas medidas básicas de cada face, por exemplo, podemos medir a distância entre os olhos, medir o tamanho de cada orelha, e altura do nariz. Para essa finalidade de extração de características, a abordagem mais precisa é permitir que o computador calcule as medidas a serem coletadas. Aprendizagem profunda faz um trabalho melhor do que os seres humanos em descobrir quais partes de um rosto são importantes para medir.

O serviço de Geração de *Encodings*, necessita acessar o servidor de arquivos contendo o a base de faces pessoas cadastradas. Para gerar os *encodings*, foi utilizado uma biblioteca *python*, *face_recognition*, acessando o método face_encodings desenvolvida por Adam Geitgey. Esse método recebe dois parâmetros de entrada, a localização da face e a imagem na escala de cinza. Em seguida é criado um dicionário de dados associando o nome do morador cadastrado que é o mesmo nome dado ao diretório e os *encodings* extraídos.

O resultado do código da Figura 26, é a geração de um arquivo python chamado *encodings* com extensão *pickle*. Esse arquivo irá conter as 128 medições de cada face.

O código da Figura 20 possui um loop no diretório das nossas imagens de cadastro, recuperamos os nomes das pessoas a partir do nome do diretório e associamos ao *encoding* extraído. É identificado a localização da face na imagem e convertida para escala de cinza, em seguida são extraídos os *encodings* daquela face e adicionado a um vetor, o mesmo se faz com o nome obtido. No final do código, é criado um dicionário de dados a partir desses dois vetores e adicionado ao arquivo *encoding.pickle* .

```
for (i, imagePath) in enumerate(imagePaths):

    print("[INFO] processing image {}/{}".format(i + 1,len(imagePaths)))
    name = imagePath.split(os.path.sep)[-2]
    if name not in knownNames:
        image = cv2.imread(imagePath)
        rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        boxes = face_recognition.face_locations(rgb,
            model=args["detection_method"])

        encodings = face_recognition.face_encodings(rgb, boxes)

        for encoding in encodings:
            knownEncodings.append(encoding)
            knownNames.append(name)
```

Figura 26- Gerando encodings

O serviço de Reconhecimento Facial utiliza o modelo de aprendizagem profundo proposto pelo OpenFace, citado na seção 2.5 da fundamentação teórica. Para o serviço de reconhecimento, é necessário carregar o arquivo *encodings.pickle* em memória, extrair os *encodings* da novo imagem e utilizar a técnica de vizinhança mais próxima do Knn para fazer

a classificação da face. A Figura 27 apresenta o código referente ao processo de reconhecimento facial.

No código da Figura 27, é ligada a webcam do notebook utilizado função do OpenCV e a conversão da entrada dos frames em escala de cinza. O método *face_locations* implementado utilizando *libs* do OpenFace e otimizado pelo módulo *face_recognition*, é responsável por localizar as faces em uma imagem. Em seguida, é executado o método *face_encodings* que recebe dois parâmetros, imagem em escala de cinza e a matriz de localização da face e retorna o vetor de *embeddings* 128d. Após a extração de características, é chamado o método *compare_faces*, responsável por medir as distâncias entre os *encodings* extraídos e da pessoa de entrada com os já cadastrados contidos no arquivo *encoding.pickle*. Caso a distância seja menor que o *threshold* definido, onde o default é 0.6, é acessado o nome da pessoa cadastrada e atribuído a variável que será mostrada na tela abaixo da face detectada.

```
data = pickle.loads(open(args["encodings"], "rb").read())
face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

while True:
                ret, frame = video_capture.read()

                small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
                #small_frame = cv2.resize(frame, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_LINEAR)

                rgb_small_frame = small_frame[:, :, ::-1]

                if process_this_frame:

                        face_locations = face_recognition.face_locations(rgb_small_frame)
                        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

                        face_names = []

                        for encoding in face_encodings:

                            matches = face_recognition.compare_faces(data["encodings"],
                            encoding)
                            name = "Desconhecido"

                            if True in matches:
                                    first_match_index = matches.index(True)
                                    name = data["names"][first_match_index]

                            face_names.append(name)
```

Figura 27- Method face_recognition

No capítulo Experimentos, serão apresentados novos parâmetros e métodos que melhoraram a performance do algoritmo de reconhecimento e apresentado as métricas obtidas a partir das variações de imagens de entrada.

O módulo de visão computacional recebe uma imagem da webcam, detecta a face e realiza o processamento de extrair os *encodings* dessa face. Após gerar os *encodings* da face de entrada, é feito um processamento para medir a menor distância comparado com os salvos no arquivo *encondings.pickle* esse processo é realizado utilizando KNN e envia a resposta para o tópico *face_recognized*, como mostra a Figura 28.

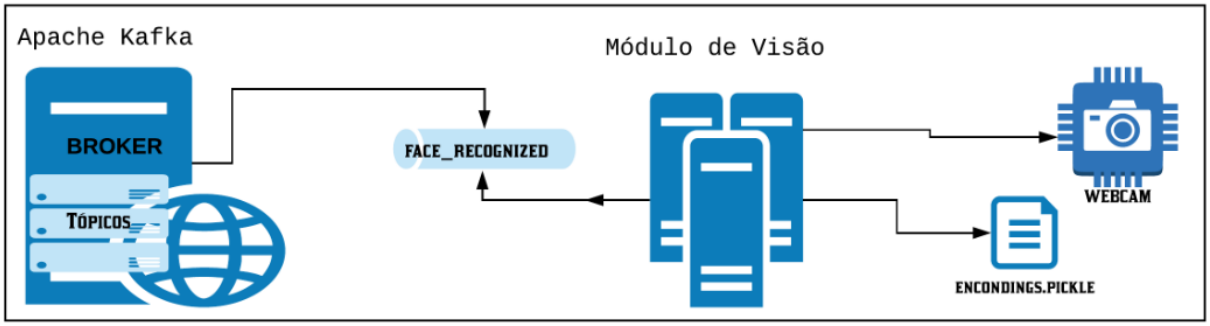


Figura 28- Arquitetura de alto nível do módulo de reconhecimento facial

Enquanto o processo de reconhecimento estiver sendo executado, o retorno do processamento é enviado para aplicação *back-end* via mensageria no tópico do *Broker*. Caso o retorno seja de alguém reconhecido, é realizada uma consulta no banco para trazer os outros dados dessa pessoa que são enviados para a aplicação web mostrar para o usuário como resposta do reconhecimento, representado no Figura 29.

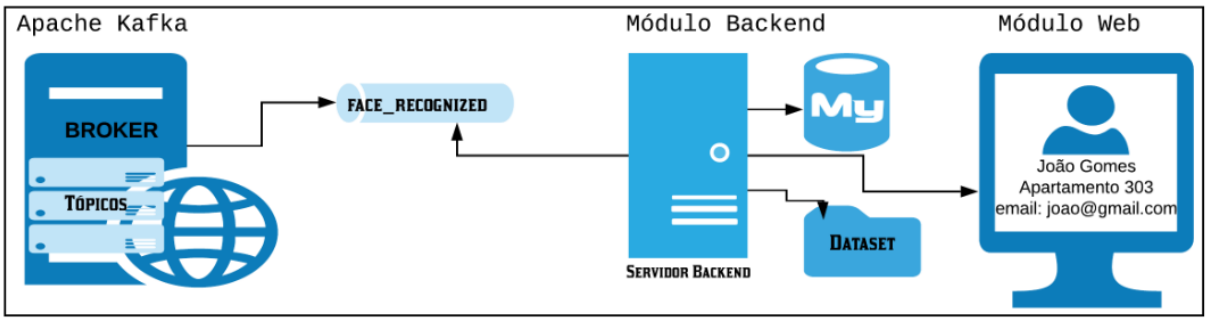


Figura 29- Arquitetura de alto nível do módulo de reconhecimento II

7 EXPERIMENTOS

Neste capítulo são apresentados os experimentos realizados com o módulo de visão computacional. O código desenvolvido para realizar os experimentos pode ser encontrado no Apêndice B e está disponibilizado no repositório Github[1].

Para aplicação dos experimentos, foi utilizado o *Dataset* Scface [2]contendo imagens de faces. As imagens foram capturadas em ambientes internos descontrolados usando cinco câmeras de vigilância de várias qualidades. O *dataset* contém 4.160 imagens (no espectro visível de infravermelho) de 130 pessoas. Imagens de diferentes câmeras de qualidade imitam as condições do mundo real e permitem experimentos robustos de algoritmos de reconhecimento.

A eficiência dos algoritmos implementados neste trabalho está relacionada à arquitetura do seu *hardware*. Os experimentos que serão apresentados a seguir, foram feitos por um notebook com as seguintes especificações:

- Processador Intel core i7 - 7ºGeração - 2.90GHz;
- Memória RAM 8GB;
- Placa de vídeo NVIDIA GeForce MX110 2GB;
- Disco Rígido 1TB;
- Sistema Operacional Ubuntu 18.04 LTS;

Algumas das imagens utilizadas nos experimentos podem ser visualizadas na Figura 30.

Figura 30 – Amostra de imagens do *Dataset* utilizado nos experimentos

Na realização de cadastro, as imagens devem ser capturadas com uma iluminação controlada e com uma boa qualidade. As imagens de cadastro utilizadas para os experimentos

[1] https://github.com/duaraujo/tcc-openface
[2] *www.scface.org*

foram capturadas por uma câmera digital Canon EOS 10D, no formato JPEG de 24 bits com o tamanho original 3.072 x 2.048 pixels apresentado na Figura 31, cortados para 1.600 x 1.200 pixels, apresentado no Figura 32. O corte foi feito após a Recomendação do padrão ANSI 385-2004 [2] para que o rosto ocupe aproximadamente 80% da imagem. Existem no total 130 imagens faciais capturadas de forma frontal, uma por pessoa.

Figura 31- Dataset de cadastro original

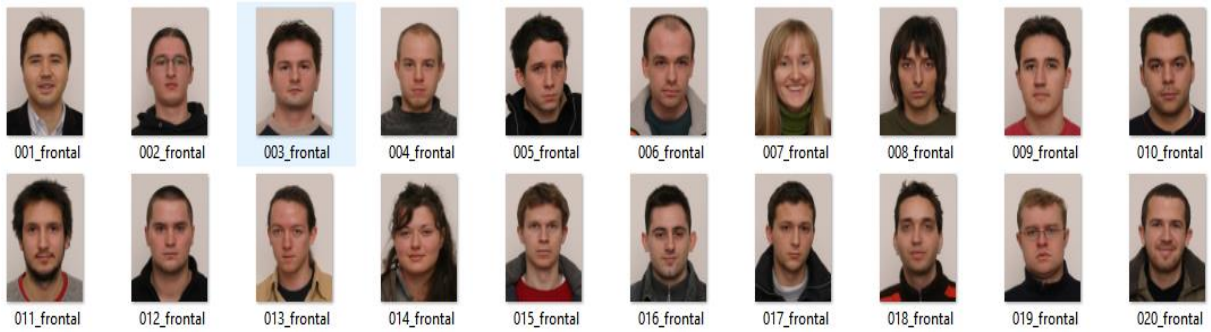


Figura 32- Dataset de cadastro recortado

Foram utilizadas as seguintes métricas necessárias para avaliar a eficácia da solução de reconhecimento facial proposta por este trabalho:

- Quantidade de acertos;
- Quantidade de pessoas desconhecidas;
- Quantidade de falsos positivos (confundir uma pessoa com outra)
- Tempo de processamento;
- Acurácia;

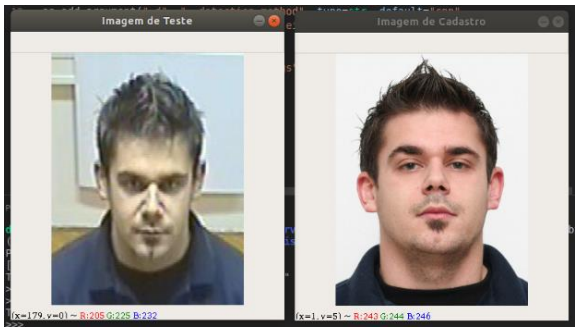
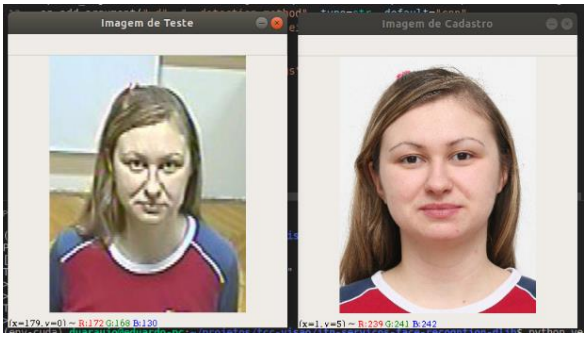
Os algoritmos testados fazem uso de alguns parâmetros, variáveis importantes para os testes e os valores variam pra cada parâmetro. Um parâmetro que influencia no resultado do reconhecimento é o *Threshold* estabelecido no momento do teste. Esse valor varia entre 0-1, essa variável está relacionada a distância do vetor de características de uma nova pessoa, para a base de pessoas cadastradas. Quanto mais baixo, mais rigoroso o processo de classificação.

Outro parâmetro é a quantidade de *Jitters*, que deforma o rosto em várias posições, reavaliando a face ao calcular a codificação. Quanto maior, mais preciso, porém mais demorado. Os experimentos foram realizados seguindo os seguintes procedimentos:
Arquivo run-teste.py, os algoritmos podem ser encontrados no apêndice B.

Tabela 3- Procedimentos de teste

1.	Definir qual *Dataset* utilizar, a partir da distância das pessoas da câmera e o tipo da câmera;
2.	Carregar as 130 imagens do *Dataset*;
3.	As informações serão enviadas para o tópico do *broker*, chamado *'face-detected'*
4.	As informações serão enviadas no formato de objetos;
5.	O objeto a ser enviado, contém 3 atributos: [*'faceDetected'*] contendo os bytes da imagem; [*'faceIndex'*] o ID verdadeiro daquela imagem; [*'fileName'*] nome do arquivo;
6.	O serviço de reconhecimento facial está observando o tópico *'face-detected'*
7.	Recebe as informações do *broker* e adiciona mais dois atributos no objeto: [*'idUser'*] que receberá o rótulo previsto pelo reconhecimento; [*'time'*] contendo o tempo utilizado para classificar a pessoa;
8.	Finaliza retornando o mesmo objeto para o tópico do *broker* chamado *'face-recognized'*
9.	O serviço *run-test* está observando o tópico *'face-recognized'*
10.	Recebe os dados enviados pelo módulo de reconhecimento
11.	Adiciona essas informações em três vetores diferentes, um contendo os ID verdadeiro associado aquela imagem, outro adicionado os valores previsto pelo módulo de reconhecimento e o terceiro contendo o tempo de processamento;
12.	A partir desses vetores é possível obter as métricas.
13.	É utilizado a uma biblioteca python chamado sklearn.metrics importando o método accuracy_score, para retornar os valores da acurácia, total de acertos, falsos positivos, desconhecidos, e o tempo de processamento. O tempo foi calculado utilizando a função *mean*() da biblioteca *numpy*.
14.	As informações ficam visíveis no console.
15.	No final, são abertas duas janelas contendo a imagem prevista ao lado da imagem de cadastra, para fins comparativos, mostrada na Tabela 4.

Tabela 4- Comparação entre imagens de teste e cadastro

 Reconhecido	 Reconhecido
 Reconhecido	 Falso Positivo

Na Tabela 4, estão sendo apresentados os resultados obtidos através dos experimentos utilizando CUDA (Arquitetura de Dispositivos de Computação Unificada), destinada a computação paralela, GPUs. Os computadores que possuem uma placa de vídeo podem fazer uso dessa arquitetura aumentando a velocidade de processamento do módulo de visão computacional. Fica evidente a demora do processamento sendo executado sem a utilização da GPU.

Tabela 5- Testes sem/com CUDA

	SEM CUDA	COM CUDA	SEM CUDA	COM CUDA
Threshold	**0.9**	**0.9**	**0.8**	**0.8**
Nº Jitter	**4**	**4**	**10**	**10**
Acurácia	90%	90%	90,76%	90,76%
Tempo m/s	**853,30**	**65,71**	**2055,72**	**100,46**

Para se obter o menor percentual de falsos positivos, ou seja, evitar classificar alguém como sendo “outra pessoa”, um dos fatores mais perigosos nas situações em que o reconhecimento facial se faz necessário, é a variável de *Threshold* ou variável de confiança. Quanto menor seu valor, a classificação se torna mais rigorosa, diminuindo a probabilidade de falsos positivos. Porém com o seu valor muito baixo, deixa de acertar em determinados casos.

Outra técnica que é possível utilizar e que pode melhorar o desempenho do algoritmo, é a quantidade de *Jitters* daquela imagem, onde o método irá deformar a imagem em vários outros ângulos e obter mais dados daquela imagem, porém, o tempo de processamento para classificar aquela imagem é mais demorado, em casos como se deve ter um retorno de imediato, mas não tão preciso, essa técnica não vale muito.

As *Tabela 6* e *Tabela 7* mostram experimentos realizados com diferentes valores para os parâmetros *Threshold* e *Jitters* utilizando GPU. Na Tabela 6 as imagens de testes foram capturadas a uma distância de 1metro da pessoa em relação à câmera e por sua vez, obteve melhores resultados em diversos cenários, em comparação com os dados da Tabela 7, onde as imagens de testes foram capturadas a 3 metros de distância.

Tabela 6- Resultado de testes com distância de 1m

Distância	03	03	03	03	03	03	03	03	03	03
Câmera	01	01	01	01	01	01	01	01	01	01
Acertou	116	63	119	118	67	117	117	117	118	116
Desconhecido	4	67	3	3	63	4	4	3	3	3
Falso Positivo	10	0	8	9	0	9	9	10	9	11
Threshold	0.6	0.5	0.9	0.8	0.5	0.6	0.6	0.7	0.9	0.9
Nº Jitters	4	4	4	10	40	20	10	15	10	1
Tempo m/s	70,91	70,83	65,71	100,46	211,75	135,32	98,33	122,06	97,64	61,17
Acurâcia	89,23%	48,46%	91,53%	90,76%	51,53%	90%	90%	90%	90,76%	89,21%

Tabela 7- Resultado de testes com distância de 3 m

Distância	04	04	04	04	04	04	04	04	04	04
Câmera	01	01	01	01	01	01	01	01	01	01
Acertou	111	58	114	113	62	112	112	111	113	110
Desconhecido	5	70	5	7	64	16	4	2	13	8
Falso Positivo	14	2	11	10	4	2	14	16	4	12
Threshold	0.8	0.5	0.7	0.7	0.5	0.5	0.9	0.7	0.7	0.6
Nº Jitters	3	5	6	6	6	4	4	3	8	1
Tempo m/s	83,21	78,73	76,25	90,36	86,83	115,52	68,43	108,32	87,44	90,46
Acurâcia	85,38 %	44,61 %	87,69 %	86,92 %	47,69 %	86,15 %	86,15 %	85,38 %	86,92 %	84,61 %

8 CONCLUSÃO

Este trabalho apresentou uma solução para o controle de acesso a condomínios, através de reconhecimento facial. Foi verificado que a implementação de reconhecimento facial exige algoritmos robustos, capazes de lidar com variações de poses dos indivíduos, qualidade das imagens de entrada, e diferentes configurações de iluminação do ambiente.

A aplicação proposta foi desenvolvida de forma modular e com a implementação de técnicas de Aprendizagem Profunda para a tarefa de reconhecimento facial. O sistema desenvolvido, foi composto por um módulo web e um de visão computacional.

O módulo web é responsável por garantir a interação com o usuário final de forma que ele consiga realizar as funcionalidades básicas para o controle de acesso, gerenciamento de blocos, apartamentos, moradores e validação da entrada a partir do auxílio da aplicação de reconhecimento facial. O módulo de visão computacional, utilizando técnicas de aprendizagem profunda, é responsável por extrair as *features* de cada indivíduo e classificar aquela pessoa como identificada ou não.

Os experimentos realizados apontam que as técnicas de visão computacional abordadas neste trabalho, atendem às expectativas do controle de acesso a condomínios, visto que a ferramenta alcançou 91,53% de acurácia nos experimentos realizados.

8.1 TRABALHOS FUTUROS

Como trabalhos futuros, para melhorar o controle de acesso, pode-se utilizar o retorno do módulo de reconhecimento e implementar novas regras de negócio na aplicação *back-end*, como registro de entrada e saída das pessoas. No módulo de visão computacional é possível realizar experimentos ampliando a quantidade de câmeras, buscar soluções aumentando a quantidade de servidores Apache Kafka e criar containers do módulo de reconhecimento facial e fazer com que esse processamento fique distribuído, possibilitando a escalabilidade da aplicação.

REFERÊNCIAS

- AMOS, Brandon; LUDWICZUK, Bartosz; SATYANARAYANAN, Mahadev; OpenFace: A general-purpose face recognition library with mobile applications, 2016.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. UML: guia do usuário. Elsevier Brasil, 2006.
- BOMBARDELLI, Felipe [Estudo sobre reconhecimento facial, Paraná. 2015]
- BRAGA, Luis Felipe [Sistemas de reconhecimento facial, São Carlos. 2013]
- CHRISTIAN, Szegedy; LIU, Wei; JIA, Yangqing; SERMANET, Pierre; REED, Scott; ANGUELOV, Dragomir; ERHAM Dumitru; VANHOUCKE, Vincent; AND RABINOVICH, Andrew. Going deeper with convolutions. CoRR, abs/1409.4842, 2014.
- DINIZ Fábio, NETO Francisco, JÚNIOR Francisco, FONTES Laysa: Revista Brasileira de Computação Aplicada, Passo Fundo, v. 5, n. 1, p. 42-54, 2013.
- GEITGEY, Adam. The world's simplest facial recognition api for Python and the command line, 2016. Disponível em: https://adamgeitgey.com/. Acesso em 22 de junho de 2019
- JAFRI, Rabia AND ARABNIA, Hamid [W. Zhao; R. Chellappa. Face Processing: Advanced Modeling and Methods. Academic Press, Inc., Orlando, FL, USA, 2005].
- JONES, Viola. Robust Real-Time Face Detection**,** International Journal of Computer Vision, p.1-18, 11 jul. 2003.
- LIN, SHANG-HUNG, 2000 [An Introduction to Face Recognition Technology volume 3 n1, 2000].
- MATTHEW, Turk AND PENTLAND, Alex [Eigenfaces for Recognition. Massachusetts, Estados Unidos: Journal of Cognitive Neuroscience, vol. 3, [6 p.]. 1991].
- MISLAV, Grgic; KRESIMIR, Delac; SONJA, Grgic. SCface – surveillance cameras face database, 30 de October 2009
- MYSQL. Documentação MySQL – Manuais de Referência, 2017. Disponivel em: <http://dev.mysql.com/doc/>. Acesso em: 26/11/2017.
- SCHROFF, Florian; KALENICHENKO, Dmitry; AND FACENET, J. Philbin: A unified embedding for face recognition and clustering. In CVPR, pages 815–823. IEEE, 2015.
- SILVA, A. A. da; SEGUNDO, M. P. Reconhecimento facial 2d para autenticação contınua. 2015.
- SILVA, A. L. Redução de características para classificação de imagens de faces. 2016
- TAIGMAN, Yaniv; YANG, Ming, RANZATO, Marc'Aurelio AND WOLF, Lior. Deepface: Closing the gap to humanlevel performance in face verification. In Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

APÊNDICE A – DESCRIÇÃO DE CASO DE USO

Tabela 8- Descrição Caso de Uso

Nome do Caso de uso	Manter morador
Sumário	Consultar moradores na base de dados; Excluir e editar dados de moradores; Cadastrar novos moradores;
Atores	Usuário
Pré-Condição	1-Usuário autenticado no sistema; 2 – Para cadastro e edição de informações dos moradores, será necessário estar com o documento de identificação;
Fluxo Principal	1-Esse caso de uso inicia quando existe a necessidade de se cadastrar um novo morador do condomínio. 2-O usuário do sistema seleciona a opção para realizar um novo cadastro. 3 – O usuário solicita um documento oficial com foto do novo morador. 4 – O “novo morador” fornece seu documento de identificação. 5 – O usuário do sistema confirma o documento com o “novo morador” e registra os dados obrigatórios no sistema.
Fluxos Alternativos	Caso o morador já esteja registrado na base de dados, o usuário do sistema terá as seguintes opções: 1-Consultar por um morador específico digitando seu nome no campo de pesquisa; 2 - Editar dados, consultando pelo nome e clicando no ícone de edição abrindo um formulário com as informações cadastradas daquele morador e habilitando edição nos campos de entrada desse formulário; 3 – Excluir morador da base dados, consultado pelo nome e clicando no ícone de lixeira;

APÊNDICE B – MÉTRICAS DE TESTES

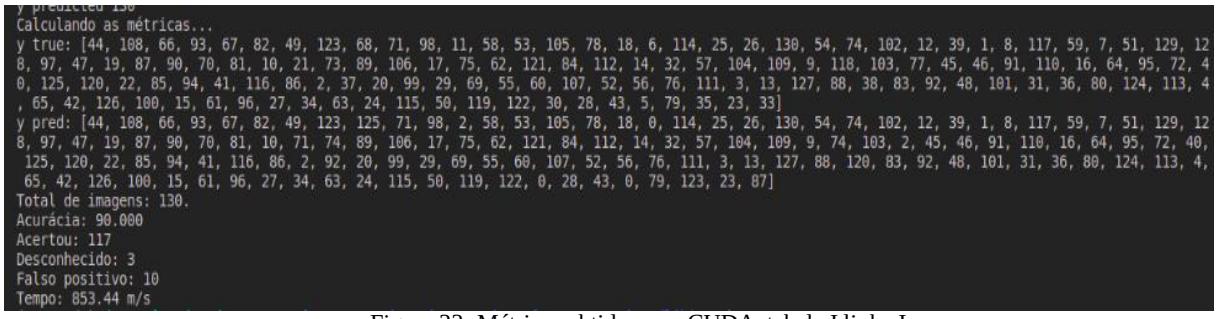


Figura 33- Métricas obtidas sem CUDA, tabela I linha I

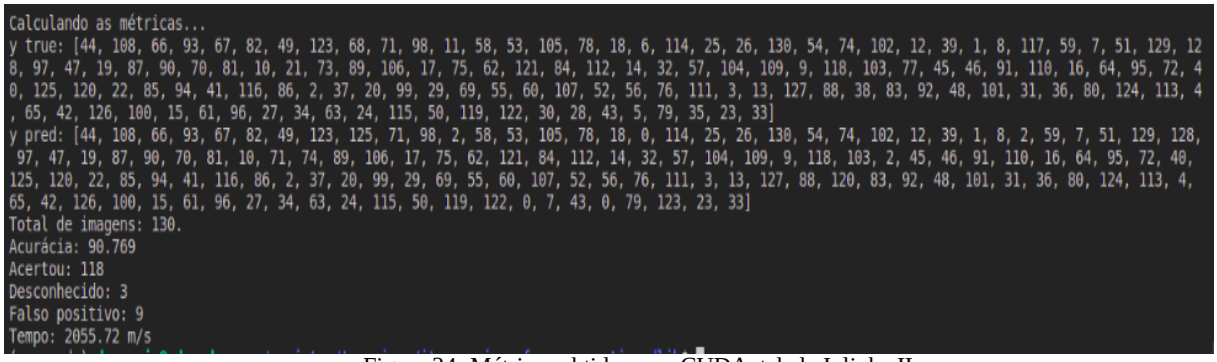


Figura 34- Métricas obtidas sem CUDA, tabela I, linha II

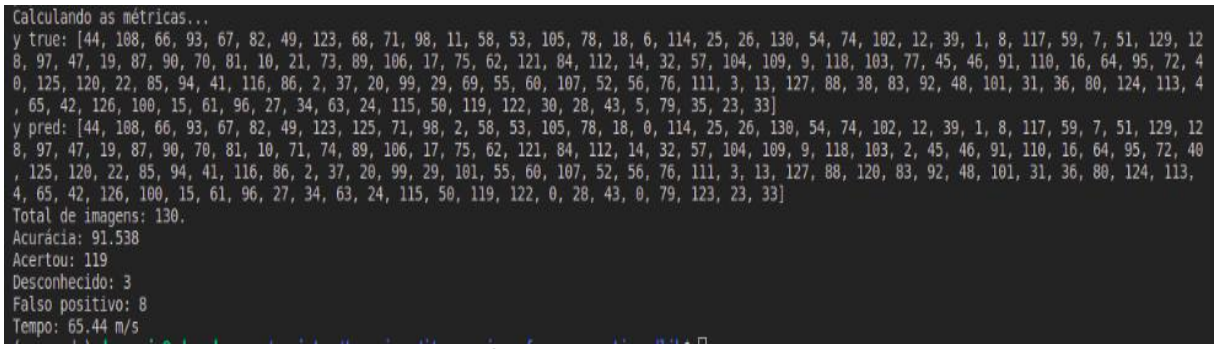


Figura 35- Métricas obtidas com CUDA, tabela I, linha IV

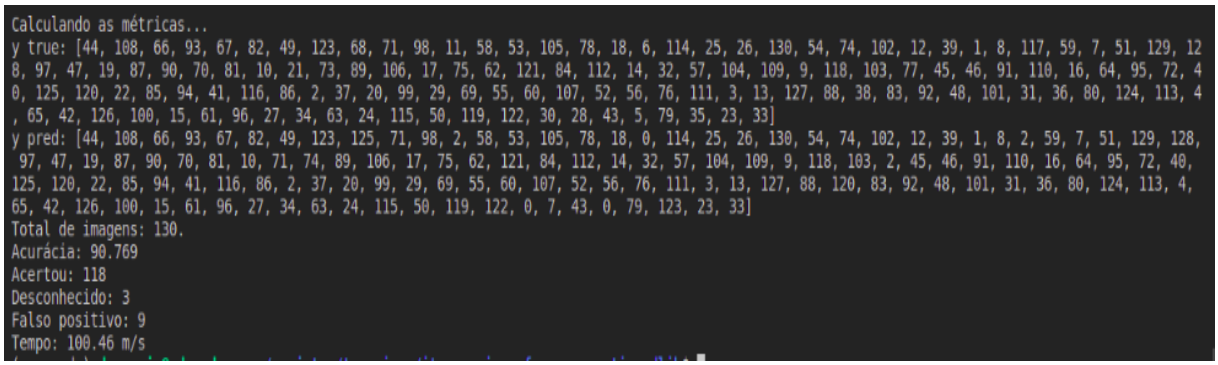


Figura 36- Mátricas obtidas com CUDA, tabela I linha V

```
77
78  def send_batch_images(entries):
79      producer = KafkaProducer(
80          bootstrap_servers=BROKER_URI, acks='all',
81          value_serializer=lambda m: msgpack.packb(m, use_bin_type=True))
82
83
84      face_index = 0
85
86      for file_name in entries[:]:
87          path_filename = dir + '/' + file_name
88
89          img = cv2.imread(path_filename)
90          ret, img_encoded = cv2.imencode('*.png', img, [cv2.IMWRITE_PNG_COMPRESSION, 0])
91
92          msg_object = dict()
93          msg_object['faceDetected'] = img_encoded.tobytes()
94          msg_object['faceIndex'] = int(file_name[0:3])
95          msg_object['fileName'] = file_name
96
97          producer.send(topic_out, msg_object).add_errback(on_send_error)
98          # print("Enviado: ", face_index)  # For debug
99          face_index += 1
100
101     producer.flush()
102     producer.close()
103     # For Debug
104     print("For debug:")
105     print(face_index)
106
```

Figura 37 - Enviar objeto para o broker

```
107
108 def get_predicts(data):
109     predicted_value = int(data['idUser'])
110     true_value = int(data['faceIndex'])
111
112     y_predicted.append(predicted_value)
113     y_true.append(true_value)
114     y_images_receveid.append(data['faceDetected'])
115
116     time_f = float (data['time'])
117     times.append(time_f)
118
119     #For debug
120     for key, val in data.items():
121         if key != 'faceDetected':
122             print(key, "=>", val)
123
```

Figura 38- Criando vetores com os IDs, previsão e os tempos

```
25
26  topic = args["topic1"]
27  topic2 = args["topic2"]
28  BROKER_URI = args["broker"]
29
30  consumer = KafkaConsumer(
31      topic,
32      value_deserializer=lambda m: msgpack.unpackb(m, raw=False),
33      bootstrap_servers=BROKER_URI)
34
35  producer = KafkaProducer(
36          bootstrap_servers=BROKER_URI,
37          value_serializer=lambda m: msgpack.packb(m, use_bin_type=True))
38
```

Figura 39- Enviar e consumir dados do broker

```
41      data = msg.value
42      img = data['faceDetected']
43      data['idUser'] = '0'
44      start = timer()
45
46      i = PIL.Image.open(io.BytesIO(img)).convert("RGB")
47      iConvert = np.array(i)
48
49      unknown_face_encodings = face_recognition.face_encodings(iConvert, num_jitters=10)
50
51      if len(unknown_face_encodings) > 0:
52          unknown_face_encoding = unknown_face_encodings[0]
53
54          matchIndex = my_compare_faces(rede["encodings"], unknown_face_encoding, 0.8)
55          if matchIndex is not None:
56              data['idUser'] = rede["names"][matchIndex["index"]]
57          else:
58              data['idUser'] = '0'
59
60      end = timer()
61      data['time'] = str((end - start)*1000);
62      print ("end...")
63      producer.send(topic2, data)
```

Figura 40- Adicionar atributos de tempo e ID previsto e enviar pro broker