RESIGeT On-Line
Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica
Uni-FACEF Centro Universitário Municipal de Franca
Vol. 13 Nro. 1 2023

# DESENVOLVIMENTO DE UMA API PARA PROCESSAMENTO DE IMAGENS EM PROL AO RECONHECIMENTO FACIAL

Matheus Henrique Gotardo Pintor
Graduado em Engenharia de Software no Uni-FACEF
matheushgotardo007@gmail.com

Geraldo Henrique Neto
Docente do Uni-FACEF
geraldo.henriqueneto@gmail.com

## Resumo

Devido ao avanço da tecnologia, metodologias para armazenamento e captura de imagens sugiram e foram aprimoradas. Como consequência disso, diversas necessidades surgiram ao longo desse avanço, como por exemplo, empresas necessitando de ferramentas e metodologias para buscar pessoas em imagens. Com isso, surgiu a ideia de criar um projeto que utiliza técnicas de inteligência artificial, visão computacional e reconhecimento facial que possibilite realizar essa busca. Inicialmente, foi projetado a criação de uma interface *WEB*, porém, como cada empresa tem seu padrão de interface/sistema, o objetivo foi alterado para a criação de uma *API*, possibilitando essas pessoas e empresas utilizá-la como serviço e conecta-la às suas interfaces. O resultado atendeu a todas as expectativas, visto que a *API* foi precisa e performática na busca e identificação das pessoas.

**Palavras-chave**: Imagens. Inteligência Artificial. Visão Computacional. Reconhecimento Facial. *API*.

## Abstract

Due to the advancement of technology, methods for storing and capturing images have been suggested and improved. As a consequence of this, several needs have emerged throughout this advance, such as people and companies needing tools and methodologies to search for people in images. With that, the idea of creating a project that uses artificial intelligence and computer vision techniques that makes it possible to carry out this search arose. Initially, the creation of a WEB interface was designed, however, as each company has its interface/system standard, the objective was changed to the creation of an API, allowing these people and companies to use it as a service and connect it as their interfaces. The result met all expectations, as the API was very accurate in the search and identification of people, in addition to being very performant.

**Keywords**: Images. Artificial Intelligence. Computer Vision. Facial Recognition. API*.*

## 1. Introdução

A primeira câmera fotográfica foi inventada em meados de 1839. Naquela época, a tecnologia utilizada era limitada, de tal forma que o tempo de exposição necessário para que a fotografia fosse feita era de até 30 minutos, por isso é muito comum que nas imagens realizadas nesse tempo só apareçam as fotos das paisagens das cidades (EDUCAMAISBRASIL, 2020).

Com o passar do tempo e com o aprimoramento das tecnologias, dispositivos fotográficos mais modernos surgiram, aumentando a capacidade de armazenamento e de captura de imagens. Dessa forma, diversas necessidades surgem em paralelo, como a de encontrar fotografias com alguém diante de milhares de imagens capturadas na galeria do celular. Ou então, caso uma empresa de fotografia deseje separar as fotos retiradas em uma noite de festa para fazer os álbuns dos formandos, muitas horas de análise e trabalho serão necessárias, visto que mais de uma pessoa pode estar presente em uma mesma foto. Se abranger mais, órgãos ou empresas de segurança necessitam encontrar determinadas pessoas diante de diversas imagens ou vídeos.

Diante desse cenário, surge a ideia de desenvolver uma *API* (*Application Programming Interface*) que utilize técnicas de inteligência artificial para fazer uma busca nas imagens, a fim de encontrar pessoas e facilitar o processo, evitando frustrações e facilitando o trabalho de empresas que utilizam fotografias digitais como meio de trabalho.

Durante o desenvolvimento desse projeto, foram realizados estudos dentro da área de Inteligência Artificial, mais especificamente na área de aprendizado de máquina voltado para visão computacional.

## 2. Inteligência Artificial

A Inteligência Artificial (IA) é um ramo de estudo da Ciência da Computação e vem sendo cada vez mais estudada e utilizada nos últimos anos. Sua adoção global cresceu de forma constante em todo o mundo, com 41% das empresas no Brasil indicando que implementaram ativamente a tecnologia. Isso ressalta ainda mais que o crescimento da IA está prestes a acelerar à medida que continua a amadurecer, tornando-se mais acessível e fácil de realizar, segundo uma pesquisa de mercado encomendada pela IBM (INFORCHANNEL, 2022).

Segundo Ludger (2013), ela pode ser definida como o ramo da ciência da computação que se ocupa da automação do comportamento inteligente e deve ser baseada em princípios teóricos e sólidos aplicados nesse campo. Esses princípios incluem as estruturas de dados usadas na representação do conhecimento, os

algoritmos necessários para aplicar esse conhecimento e as linguagens e técnicas de programação usadas em sua implementação.

Apesar do avanço da tecnologia, ainda não se conseguiu criar um modelo de IA genérica, capaz de realizar qualquer tipo de tarefa, mas sim, modelos específicos treinados para tarefas específicas.

## 3. Imagens Digitais

Para entender como o algoritmo utilizado para o reconhecimento facial funciona, primeiramente precisamos entender como uma imagem digital é formada e representada no computador.

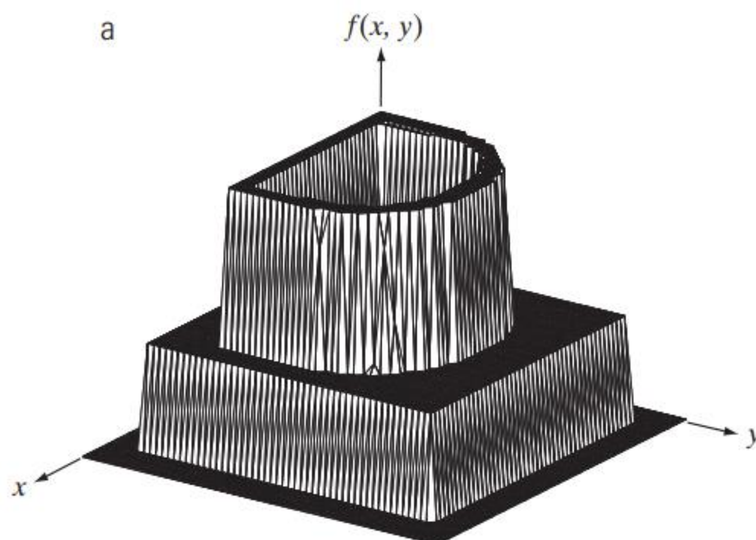
De acordo com Scuri (1999), a imagem contínua é modelada matematicamente pela função:

$$f(x, y) = Cor \text{ no ponto } (x, y) \\ x \quad [0, X]\ e\ y \quad [0, Y] \qquad (1)$$

, onde x e y são números reais, limitados ao intervalo de 0 a X, e de 0 a Y.

De acordo com Gonzales e Woods (2010), existem basicamente três formas de se representar a função f(x,y), conforme podemos visualizar na Figura 1. A primeira delas, é de forma gráfica, com dois eixos determinando a localização espacial e o terceiro eixo representando os valores de f (intensidades) como uma função das duas variáveis espaciais x e y.

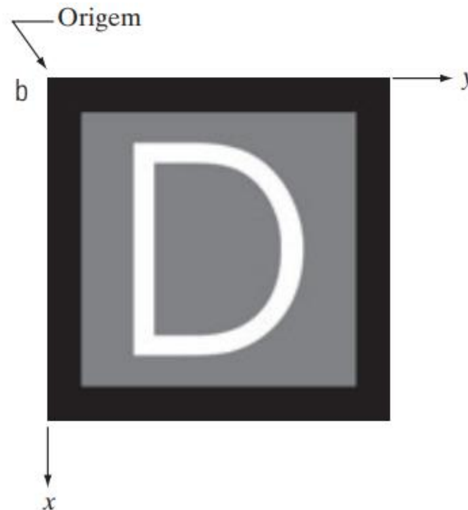
**Figura 1 -** Representação gráfica da imagem digital




**Fonte:** GONZALES; WOODS, 2010

Já a segunda forma, Figura 2, é muito mais comum. Ela mostra como uma imagem representada pela função f(x,y) seria visualizada em um monitor ou em uma fotografia. Aqui, o nível de cinza de cada ponto é proporcional ao valor da intensidade f desse ponto.

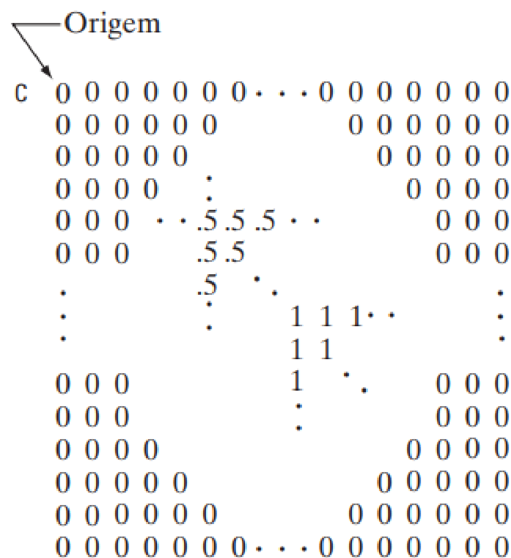
**Figura 2 -** Representação da imagem digital visualizada no monitor




**Fonte:** GONZALES; WOODS, 2010

A terceira representação, Figura 3, é somente para mostrar os valores numéricos f(x,y) na forma de uma matriz. Essa é utilizada para processamento e desenvolvimento de algoritmos.

**Figura 3 -** Representação da imagem digital em forma de matriz




**Fonte:** GONZALES; WOODS, 2010

Transformando essa matriz para a forma de equação, obtemos o seguinte resultado, ora visualizado na Figura 4:

**Figura 4 -** Representação matriz na forma de equação

$$f(x,y)=\begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \quad (2)$$

**Fonte:** GONZALES; WOODS, 2010

Cada elemento dessa matriz é chamado de *pixel* (abreviação de *Picture Element*), cujos valores variam entre 0 e 255 na representação RGB (*Red*, *Green* e *Blue*), onde 0 simboliza a cor preta e 255 a cor branca.

Na Figura 5, é factível apresentar os conceitos explicados anteriormente em uma imagem digital convencional, onde cada divisão (quadrado) corresponde a um valor entre 0 e 255.

**Figura 5 -** Representação da imagem digital

**Fonte:** Compilação do próprio autor

## 4. Reconhecimento Facial

O reconhecimento facial tem sido um dos temas mais pesquisados nas áreas de inteligência artificial e visão computacional nos últimos anos.

Destina-se a usar computadores para extrair informações de reconhecimento eficazes de imagens faciais e identificar a identidade dos objetos faciais semelhante a outras características biológicas (como íris, impressões digitais etc.) do corpo humano (WANG, 2023).

Sua aplicação vem se tornando cada vez mais difundida e utilizada nos últimos anos, principalmente em aplicações de vigilância, a biometria, controle de acesso e aplicações das leis.

Por esse motivo, diversas soluções e algoritmos para solucionar problemas dessas áreas de aplicações surgiram ao longo desses anos. Nesse trabalho iremos falar de um algoritmo em específico, chamado LBPH (*Local Binary Patterns*).

### 4.1 LBPH

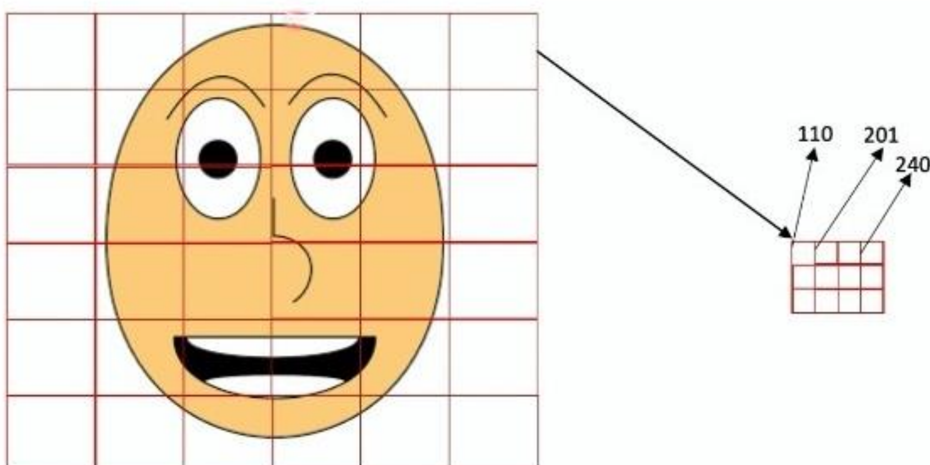
LBPH é um algoritmo criado por Ojala e é um poderoso meio de descrição de textura de imagens. Ele define o valor do *pixel* de uma imagem com base no valor de seus *pixels* vizinhos (AHONEN; HADID; PIETIKÄINEN, 2004).

Como ele é um descritor de textura, pode também ser usado para tarefas de reconhecimento facial (Prado, 2017).

- Inicialmente, a imagem precisa ser convertida para escala de cinza. Esse passo está presente em diversos algoritmos, porém é especialmente necessário para o LBPH.

- Posteriormente, é realizado uma divisão da imagem em sub-areas, conforme ilustrado na Figura 6, onde cada área será uma matriz de *pixels* 3x3, cujos valores variam de 0 até 255.

**Figura 6 -** Representação da forma como o algoritmo LBPH divide a imagem em sub-areas




**Fonte:** SINGH, 2021

RESIGeT On-Line
Revista Eletrônica de Sistemas de Informação e Gestão Tecnológica
Vol. 13 Nro. 1 2023
Uni-FACEF Centro Universitário Municipal de Franca

- O próximo passo é analisar os *pixels* de cada sub-area e criar matrizes com base no resultado dessa análise. O *pixel* central será o valor de referência e os *pixels* vizinhos serão comparados com ele. Caso seja maior ou igual, seu novo valor será 1, caso seja menor, será 0. Esse processo pode ser observado nas Figuras 7 e 8.

**Figuras 7 e 8 -** Representação do processo de limiarização dos pixels

| 12 | 15 | 18 |
|---|---|---|
| 5 | 8 | 3 |
| 8 | 1 | 2 |

| 1 | 1 | 1 |
|---|---|---|
| 0 | 8 | 0 |
| 1 | 0 | 0 |

**Fonte:** SINGH, 2021

- Na sequência, torna-se necessário gerar um valor binário dos elementos dessa nova matriz. Primeiramente, devemos pegar o valor do elemento no canto superior esquerdo, e posteriormente, os valores dos *pixels* no sentido horário ao *pixel* central. O resultado será 11100010. Esse processo pode ser observado na Figura 9.

**Figura 9 -** Representação do processo de geração do valor binário dos *pixels vizinhos*

| 1 | 1 | 1 |
|---|---|---|
| 0 | 8 | 0 |
| 1 | 0 | 0 |

**Fonte:** SINGH, 2021

- Agora, deve-se converter o valor binário para decimal, obtendo o seguinte resultado: 226. Esse será o valor que melhor representa os *pixels* vizinhos.

- Um histograma das ocorrências dos valores de cada sub-area é gerado e concatenados no final, gerando um histograma maior com todas as ocorrências dos *pixels*.

- A última etapa é o reconhecimento. Iremos gerar os histogramas da imagem a ser reconhecida e eles serão comparados com os histogramas das imagens de treinamento. O resultado da comparação será um número, representando a distância entre os dois histogramas.

## 5. Materiais e Métodos

O desenvolvimento desse projeto utiliza diversas ferramentas e métodos dentro da área de tecnologia e serão detalhados a seguir.

### 5.1. Python

Para a implementação do projeto, foi utilizado o *Python*, que é uma linguagem de programação de alto nível, dinâmica, interpretada, modular, multiplataforma e orientada a objetos, o que possibilita maior controle e estabilidade de códigos para projetos de grandes proporções (Kriger, 2022).

O motivo de sua escolha, se dá pelo fato de ser muito poderosa e possui diversas ferramentas e bibliotecas na área de análise de dados e visão computacional, o que facilita o desenvolvimento do projeto.

- numpy: ferramenta feita em *Python*, que possui diversas funcionalidades e métodos para processar e trabalhar com grandes *arrays* e matrizes, que representarão as imagens.

- aiofiles: é uma biblioteca *Apache2* licenciada, escrita em *Python*, para lidar com arquivos de disco locais em aplicativos assíncronos.

- autopep8: ferramenta para auxiliar na formatação e organização do código fonte, durante o desenvolvimento.

- dlib: biblioteca com várias soluções de aprendizado de máquina. Sua utilização foi necessária para realizar a detecção facial.

- OpenCV: é uma poderosa ferramenta de visão computacional, que permite o processamento de imagens e disponibiliza diversos algoritmos de reconhecimento facial, inclusive o LBPH.

- FastAPI: é um *framework* construído em *Python*, que possibilita a criação de um servidor Web, ao qual iremos criar uma API (*Application Programming Interface*) que disponibilize funcionalidades para *upload* de imagens, treinamento e reconhecimento de faces.

- pika: biblioteca que possui as ferramentas necessárias para fazer a comunicação com o *broker* de mensageria.

- pydantic-settings: biblioteca utilizada para configuração e parametrização do projeto. Ela tem a funcionalidade de ler dados de variáveis de ambiente, mantendo credenciais, como senhas e *tokens* seguros, além de deixar a configuração do *software* mais dinâmica, uma vez que necessário alterar algum valor, basta alterar a variável de ambiente, sem necessidade de alterações no código fonte.

- shortuuid: biblioteca utilizada para geração de uuid (*Universally Unique Identifier*)

- virtualenv: ferramenta para criar ambientes virtuais *python*. Foi utilizada a fim de isolar as instalações de bibliotecas, ou seja, tudo que é instalado para o funcionamento da aplicação, ficará em uma pasta, sem afetar bibliotecas e configurações existentes no computador.

### 5.2. RabbitMQ

*RabbitMQ* é um *message broker* altamente consolidado e utilizado por quem trabalha com comunicação entre sistemas. Operando de forma assíncrona, ele age como um intermediário que processa as mensagens entre produtores e consumidores, além de contar com filas que possuem diversas opções de encaminhamento (WILLIANS, 2022).

Sua utilização é necessária para que o projeto consiga processar diversas imagens simultaneamente, sem perder a performance, garantindo escalabilidade.

### 5.3 Colaboratory

*Colaboratory* ou *Colab*, é um serviço de nuvem gratuito hospedado pelo Google para incentivar a pesquisa de Aprendizado de Máquina e Inteligência Artificial. Ele permite escrever e executar *Python* no navegador, sem a necessidade de realizar nenhuma instalação ou configuração.

Sua utilização é de extrema importância, pois facilita a criação e os testes dos algoritmos utilizados no projeto.

### 5.4 Docker

O *Docker* é um *software* usado para implantar aplicativos dentro de *containers* virtuais. A conteinerização permite que vários aplicativos funcionem em diferentes ambientes complexos.

Sua utilização é opcional, pois o projeto não depende dele para funcionar. Ele foi apenas um facilitador, permitindo criar um *container* para o servidor *RabbitMQ*, dispensando qualquer configuração adicional no computador.

## 6. Desenvolvimento

Essa seção tem o objetivo de apresentar os passos seguidos durante a implementação do projeto, desde a criação dos algoritmos para realizar o reconhecimento facial até a criação das rotas da API para o processamento das imagens.

### 6.1. Documentações

Documentar um *software* é um requisito fundamental para seu ciclo de vida (manutenção, consulta, melhorias). Por esse motivo, foram elaborados os documentos apresentados a seguir.

#### 6.1.1 Documento de Requisitos

O documento de requisitos contém todos os requisitos solicitados pelo cliente (dono ou usuário final da aplicação) que o projeto deve ter, além de ser uma das principais etapas dentro de um projeto de desenvolvimento de *software*, visto que os erros cometidos nessa fase serão perpetuados e amplificados ao longo da cadeia de desenvolvimento, acarretando em prejuízos consideráveis ao projeto, seja econômico ou de qualidade do produto desenvolvido. (ROBERTO, 2018).

De acordo com as Tabelas 1, 2 e 3, os requisitos do projeto foram detalhados em *id*, descrição, categoria, prioridade, campos e regra de negócio

**Tabela 1 -** Tabela do requisito funcional para upload das imagens

| TABELA DE REQUISITOS | |
|---|---|
| **ID:** RF001 | **NOME DO REQUISITO**: Upload das imagens |
| **DESCRIÇÃO** | A API deverá possuir uma rota “/train” para que seja possível informar o nome e as fotos que correspondem ao rosto de determinada pessoa e treinar o algoritmo |
| **CATEGORIA** | Evidente |

| | |
|---|---|
| **PRIORIDADE** | Essencial |
| **CAMPOS** | name **Varchar**;<br>images **List[Blob]**; |
| **REGRA DE NEGÓCIO** | • O usuário deverá informar os campos necessários e realizar a chamada na API<br>• Após realizar a chamada, a API deverá salvar as informações recebidas<br>• Após salvar as informações, a API deverá enviar uma mensagem ao RabbitMQ contendo o id do usuário salvo, para que o *worker* possa treinar o algoritmo<br>• Uma resposta de sucesso deverá ser retornada ao usuário |

**Fonte:** Compilação do próprio autor

**Tabela 2 -** Tabela do requisito funcional para treinamento do algoritmo

| **ID:** RF002 | **NOME DO REQUISITO**: Treinamento do algoritmo |
|---|---|
| **DESCRIÇÃO** | A API deverá enviar uma mensagem ao *worker* para que ele processe as imagens de cada pessoa e treine o algoritmo |
| **CATEGORIA** | Não Evidente |
| **PRIORIDADE** | Essencial |
| **CAMPOS** | |
| **REGRA DE NEGÓCIO** | • O *Worker* irá processar a mensagem enviada pela API, recuperar as imagens salvas e enviá-las ao algoritmo LBPH. |

**Fonte:** Compilação do próprio autor

**Tabela 3 -** Tabela do requisito funcional para busca de pessoas

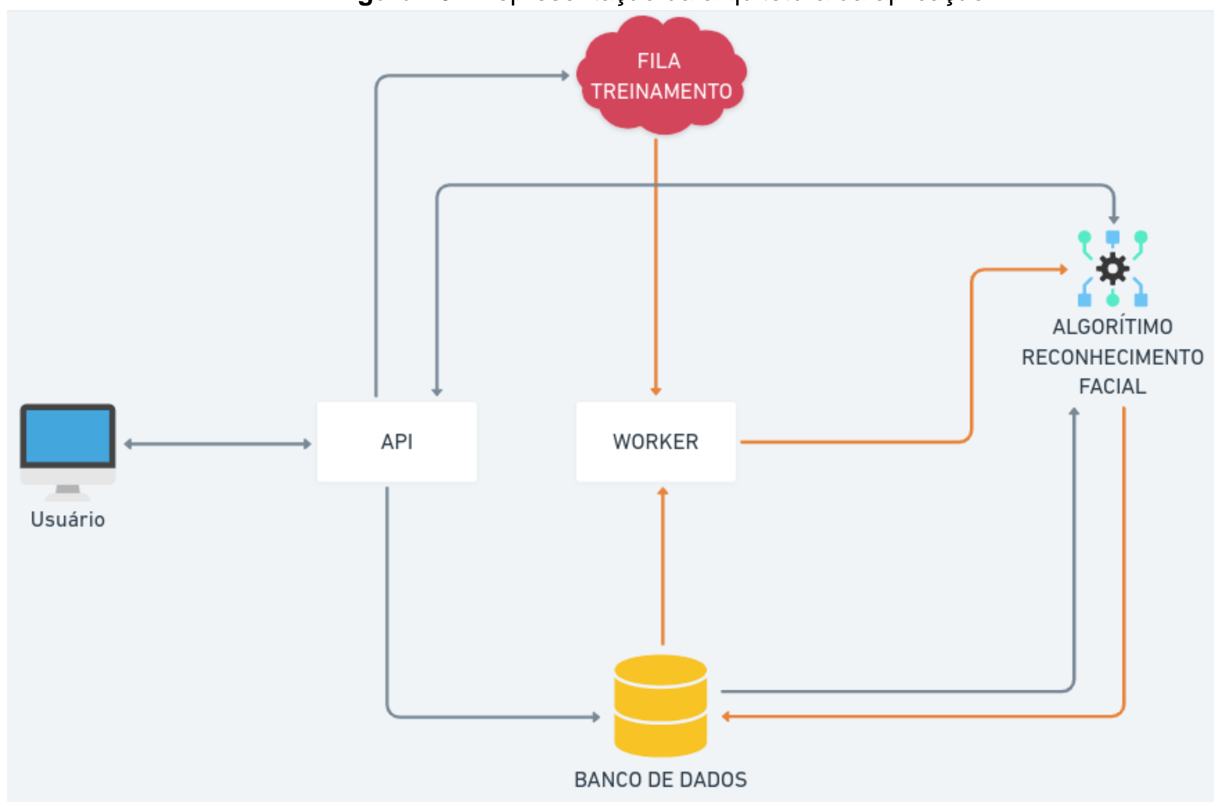
| **ID:** RF003 | **NOME DO REQUISITO**: Busca de pessoas |
|---|---|
| **DESCRIÇÃO** | A API deverá possuir uma rota “/recognize” para que seja possível informar a foto em que deseja buscar as pessoas previamente treinadas |
| **CATEGORIA** | Evidente |
| **PRIORIDADE** | Essencial |
| **CAMPOS** | image **Blob**; |
| **REGRA DE NEGÓCIO** | • O usuário deverá informar o campo necessário e realizar a chamada na API<br>• Após realizar a chamada, a API deverá salvar a informação recebida<br>• Após salvar a informação, a imagem deverá ser enviada ao algoritmo para encontrar as pessoas na imagem<br>• Os nomes das pessoas encontradas deverão ser retornados ao usuário. |

**Fonte:** Compilação do próprio autor

### 6.1.2 Arquitetura

A arquitetura utilizada para o projeto também foi documentada e ilustrada. Essa documentação tem o objetivo de identificar a comunicação entre as dependências técnicas da aplicação, bem como explicar os recursos de infraestrutura utilizados.

**Figura 10 -** Representação da arquitetura da aplicação




**Fonte:** Compilação do próprio autor

Conforme visto na Figura 10, existem seis elementos importantes na arquitetura geral da aplicação, que serão explicados a seguir:

- Usuário: é o consumidor da API, podendo ser desde um terminal *Linux* até uma aplicação *WEB Frontend*. Ele que será o usuário final do projeto, ou seja, realizará o *upload* das imagens para treinamento e busca.

- API: é o módulo principal construído em *Python*, juntamente com o *framework* FastAPI, que será utilizado pelo usuário como *interface* para poder enviar as imagens e realizar a busca.

- Fila Treinamento: fila de mensagens do *RabbitMQ*, onde as mensagens (*ids* das pessoas) para treinamento do algoritmo serão enviadas. Será por meio dela que a API e o WORKER irão se comunicar.

- Worker: é o consumidor da fila de mensagens. Ele será responsável por receber as mensagens (*ids* das pessoas) e realizar o treinamento do algoritmo.

- Algoritmo Reconhecimento Facial: é o algoritmo LBPH. Ele será responsável por gerar os histogramas e armazená-los para serem utilizados no reconhecimento facial.

- Banco de Dados: local onde as informações das faces das pessoas, juntamente com seus respectivos nomes e histogramas, serão armazenadas. A fim de simplificar o desenvolvimento do projeto, foi utilizado o próprio sistema de arquivos do sistema operacional para armazenar os dados. Porém, caso seja necessário utilizar algum armazenamento externo, como banco de dados dedicado ao armazenamento de arquivos, a aplicação funcionará normalmente, sem grandes alterações.

Para facilitar o entendimento dos fluxos identificados pelas ligações (flechas) na Figura 10, a arquitetura foi desmembrada em: arquitetura do fluxo de treinamento e arquitetura do fluxo de busca.

### 6.1.3 Arquitetura do Fluxo de Treinamento

Nesse tópico será explicado como o fluxo de treinamento do algoritmo, apresentado na Figura 11, funcionará.

- Fluxo 1: nessa primeira etapa, o usuário realizará uma chamada na API, informando o nome e as fotos da pessoa que deseja informar ao algoritmo de treinamento.

- Fluxo 2: a API recebe os dados informados no fluxo 1, realiza as validações necessárias e salva os dados no banco de dados.

- Fluxo 3: após salvar os dados, uma mensagem (*id* da pessoa salva no banco) é enviada a fila de treinamento.

- Fluxo 4: uma resposta é enviada ao usuário, informando que sua solicitação ocorreu com sucesso.

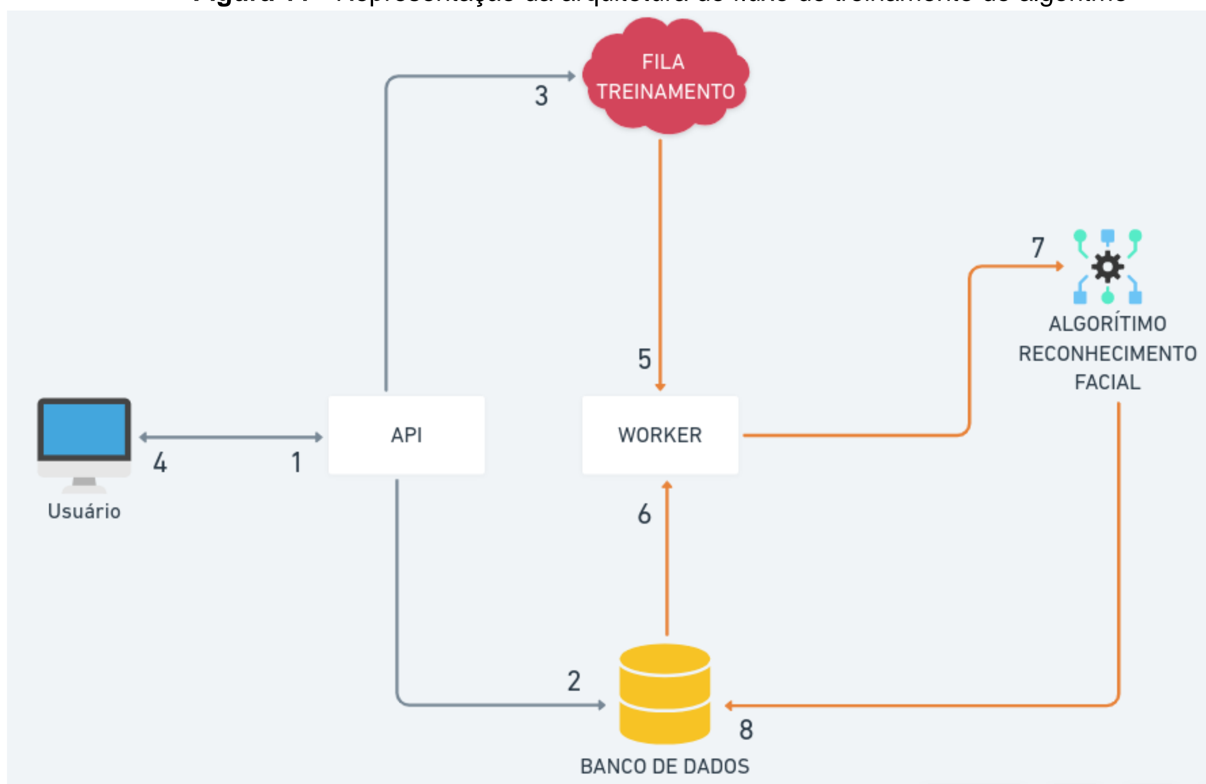
- Fluxo 5: em segundo plano, o processamento das imagens e treinamento do algoritmo será feito. O *worker* receberá a mensagem enviada no fluxo 3.

- Fluxo 6: após receber a mensagem contendo o *id* da pessoa, o *worker* realizará uma busca das imagens salvas no banco de dados (ação realizada no fluxo 2).

- Fluxo 7: após ter as imagens, o *worker* irá enviá-las ao algoritmo de treinamento (LBPH), juntamente com o *id* recebido da mensagem.

- Fluxo 8: o algoritmo irá gerar os histogramas que representam as faces e salvá-los em um arquivo **.yml** no banco de dados.

**Figura 11 -** Representação da arquitetura do fluxo de treinamento do algoritmo




**Fonte:** Compilação do próprio autor

### 6.1.4 Arquitetura do Fluxo de Busca

Nesse tópico será explicado como o fluxo de busca, que está representado na Figura 12, funcionará. Ele é um pouco mais simples, visto que toda comunicação entre as partes é realizada de forma síncrona.

- Fluxo 1: nessa primeira etapa, o usuário realizará uma chamada na API, informando a imagem em que está(ão) presente(s) a(s) pessoa(s) que deseja encontrar.
- Fluxo 2: a API recebe a imagem informada no fluxo 1, realiza as validações necessárias e a salva no banco de dados, para futuras consultas.

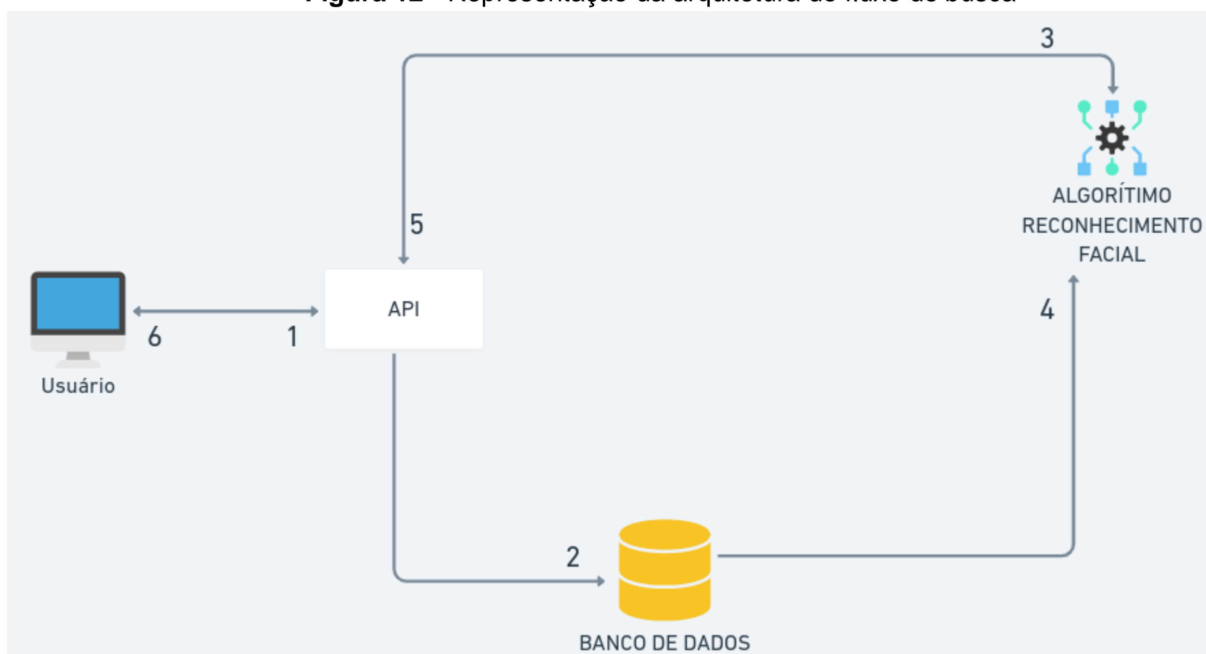
- Fluxo 3: após salvar a imagem, sua localização (caminho do arquivo salvo) é enviada ao algoritmo de reconhecimento facial (LBPH).

- Fluxo 4: o algoritmo recupera a imagem salva no banco e identifica qual(is) pessoa(s) está(ão) presente(s) nela.

- Fluxo 5: o algoritmo devolve a informação (nomes) para os módulos da API.

- Fluxo 6: a API retorna quem está presente na imagem para o *client* (usuário que realizou a chamada na API).

**Figura 12 -** Representação da arquitetura do fluxo de busca




**Fonte:** Compilação do próprio autor

## 6.2 Código Fonte

O código fonte contém todas as instruções de alto nível que o computador irá interpretar. Nessa seção, será apresentado as principais partes do código fonte do projeto e as demais estarão disponíveis no Github.

### 6.2.3 *Interface* para Treinamento e Reconhecimento Facial

Para realizar operações de treinamento e reconhecimento facial, foi necessário criar uma classe, a qual irá abstrair a comunicação com as bibliotecas *OpenCV* e *dlib* e disponibilizar essas funcionalidades para as outras partes do código fonte utilizar.

De acordo com as Figuras 13, 14, 15, 16 e 17, foi criado a classe *FaceRecognize*, contendo 6 (seis) métodos:

**Figura 13 -** Representação da *interface* para realizar operações de treinamento e reconhecimento facial - Parte 1

```
class FaceRecognize:

    def __init__(self, train_folder, recognize_folder, train_filename):
        self.train_folder = os.path.join(config.DATABASE_FOLDER, train_folder)

        self.recognize_folder = os.path.join(
            config.DATABASE_FOLDER, recognize_folder)

        self.train_filepath = os.path.join(
            config.DATABASE_FOLDER, train_filename)

        self.hog_classifier = dlib.get_frontal_face_detector()
        self.lbph_classifier = cv2.face.LBPHFaceRecognizer_create()

    def train_or_update_model(self, images_path):
        ids, faces = self.get_images_data(
            images_path
        )

        if os.path.isfile(self.train_filepath):
            self.update_model(ids, faces)
        else:
            self.train_model(ids, faces)
```

**Fonte:** Compilação do próprio autor

**Figura 14 -** Representação da *interface* para realizar operações de treinamento e reconhecimento facial - Parte 2

```
def update_model(self, ids, faces):
    self.lbph_classifier.read(self.train_filepath)
    self.lbph_classifier.update(faces, ids)
    self.lbph_classifier.write(self.train_filepath)

def train_model(self, ids, faces):
    self.lbph_classifier.train(faces, ids)
    self.lbph_classifier.write(self.train_filepath)
```

**Fonte:** Compilação do próprio autor

**Figura 15 -** Representação da *interface* para realizar operações de treinamento e reconhecimento facial - Parte 3

```python
def recognize(self, image_path):
    faces = self.get_faces(os.path.join(self.recognize_folder, image_path))
    predictions = []

    for face in faces:
        self.lbph_classifier.read(self.train_filepath)
        label, dist = self.lbph_classifier.predict(face)
        predictions.append(label)

    return predictions
```

**Fonte:** Compilação do próprio autor

**Figura 16 -** Representação da *interface* para realizar operações de treinamento e reconhecimento facial - Parte 4

```python
def get_faces(self, image_path, resize=(200, 200), zoom=20):
    image = cv2.imread(image_path)

    detections = self.hog_classifier(image)

    cropped_images = []

    for detection in detections:
        l, t = detection.left(), detection.top()
        r, b = detection.right(), detection.bottom()

        cropped_image = image[t + zoom:b - zoom, l + zoom:r-zoom]
        cropped_image = cv2.resize(cropped_image, resize)

        cropped_images.append(cv2.cvtColor(
            cropped_image, cv2.COLOR_BGR2GRAY))

    return cropped_images
```

**Fonte:** Compilação do próprio autor

**Figura 17 -** Representação da *interface* para realizar operações de treinamento e reconhecimento facial - Parte 5

```
def get_images_data(self, images_path):
    new_path = os.path.join(self.train_folder, images_path)
    faces_files = [file for file in os.listdir(new_path)]

    ids = [int(images_path.split('-')[0])] * len(faces_files)
    faces = []

    for file in faces_files:
        path = new_path + f'/{file}'
        image = self.get_faces(path)[0]

        faces.append(image)

    return np.array(ids), faces
```

**Fonte:** Compilação do próprio autor

- train_or_update_model: método responsável por treinar ou atualizar as informações da base de treinamento, que contém os dados dos histogramas das faces.

- update_model: método responsável por atualizar os dados que contém os histogramas das faces.

- train_model: método responsável por criar uma base de dados que contém os histogramas das faces.

- recognize: método responsável por realizar o reconhecimento facial, retornando o nome das pessoas correspondentes as faces identificadas e reconhecidas.

- get_faces: método responsável por identificar as faces em uma imagem, recortá-las, redimensioná-las, aplicar um determinado *zoom* e convertê-las para a escala de cinza. A aplicação do *zoom* é necessária para que as bordas e o fundo da imagem sejam desconsiderados, focando apenas nos traços do rosto da pessoa, evitando influências negativas nos resultados do reconhecimento facial.

- get_images_data: método responsável por atribuir um *id* único para cada imagem, retornando os dados necessários aceitos pelo *OpenCV* para realizar o treinamento do algoritmo.

#### 6.2.4 Rota para Treinamento do Algoritmo de Reconhecimento Facial

É a camada do projeto que contém a configuração da rota de treinamento. De acordo com o documento de requisitos, ela deverá ser o prefixo */train*.

**Figura 18 -** Representação da rota */train*

```
from typing import List, Annotated
from fastapi import APIRouter, UploadFile, Form

from app.core.services.train import TrainService

router = APIRouter(
    prefix="/train",
    tags=["train"],
)


@router.post("")
async def upload_train_images(
    name: Annotated[str, Form()],
    images: List[UploadFile]
):
    await TrainService().upload_images(name, images)

    return {"message": "upload realizado com sucesso"}
```

**Fonte:** Compilação do próprio autor

Conforme ilustrado na Figura 18, ela receberá os dados enviados pelo usuário e passará para a camada *TrainService*.

#### 6.2.5 Rota para Reconhecimento Facial

É a camada do projeto que contém a configuração da rota de reconhecimento. De acordo com o documento de requisitos, ela deverá ser o prefixo */recognize*.

Conforme ilustrado na Figura 19, ela receberá os dados enviados pelo usuário e passará para a camada *RecognizeService*.

**Figura 19 -** Representação da rota */recognize*

```
from fastapi import APIRouter, UploadFile

from app.core.services.recognize import RecognizeService

router = APIRouter(
    prefix="/recognize",
    tags=["recognize"],
)


@router.post("")
async def recognize(image: UploadFile):
    names = await RecognizeService().recognize(image)

    return {"names": names}
```

**Fonte:** Compilação do próprio autor

#### 6.2.6 *Worker* para Treinamento do Algoritmo de Reconhecimento Facial

É o *software* responsável por ler as mensagens enviadas pela API (*Application Programming Interface*) ao *broker* e enviá-las a camada de serviço para o treinamento do algoritmo de reconhecimento facial.

**Figura 20 -** Representação do worker

```
from app.core.interfaces.message_broker import MessageBroker
from app.core.services.train import TrainService
from app.core.config import config


def train_callback(ch, method, properties, body):
    train_service = TrainService()
    message = body.decode()
    train_service.train(message)


def main():
    message_broker = MessageBroker(
        queue_name=config.RABBIT_TRAIN_QUEUE
    )

    message_broker._declare_consumer(train_callback)

    message_broker.start_consuming()


if __name__ == '__main__':
    main()
```

**Fonte:** Compilação do próprio autor

De acordo com a Figura 20, o *worker* é uma aplicação que irá se comunicar com o *broker* de mensageria e possui duas funções:

- main: é a função que será chamada quando o *worker* iniciar. Ela irá realizar a conexão com o *broker* e informar a ele qual função deverá ser chamada para processar as mensagens da fila.

- train_callback: é a função responsável por receber a próxima mensagem da fila e enviar ao serviço de treinamento.

## 7. Resultados

Para demonstrar o funcionamento e o resultado do projeto, foi utilizado 130 imagens de 13 pessoas distintas, as quais 65 foram utilizadas para treinamento do algoritmo e 65 para teste. As imagens de teste contêm diversos cenários propícios a erro, como imagens de pessoas realizando expressões faciais e imagens com baixa luminosidade

### 7.1 Porcentagem de Acerto do Algoritmo

A porcentagem de acerto foi calculada por meio de um *script* criado e executado na ferramenta *Colab*. Após fazer o treinamento do algoritmo, as imagens de teste são carregadas e enviadas uma a uma para reconhecimento facial. Caso o resultado esperado for igual ao retornado pelo algoritmo, ele é contabilizado e a porcentagem de acerto é calculada ao final do processo.

Utilizando um total de 26 imagens de pessoas com boa luminosidade e sem expressões faciais, o algoritmo apresentou 100% de acerto, conforme demonstrado na Figura 21.

Já utilizando um total de 39 imagens, sendo 26 do cenário anterior e 13 de pessoas com expressões faciais, o algoritmo apresentou 87,18% de acerto, conforme demonstrado na Figura 22.

Por fim, foi utilizado 65 imagens, sendo as mesmas 39 do cenário anterior e 26 de pessoas em um cenário com baixa luminosidade. O algoritmo apresentou 75,38% de certo, conforme demonstrado na Figura 23.

**Figura 21 -** Representação da porcentagem de acerto utilizando imagens com boa luminosidade e sem expressões faciais

```
# porcentagem de acerto do algorítimo
test_files = os.listdir(yale_faces_test_path)

total = len(test_files)
correct_results = 0

for test_file in test_files:
  expect_result = int(test_file.split('.')[0].replace('subject', ''))

  face = get_faces(f'{yale_faces_test_path}/{test_file}')[0]
  result, dist = lbph_classifier.predict(face)

  if result == expect_result:
    correct_results += 1

print(f'Total de imagens para treinamento: {len(os.listdir(yale_faces_train_path))}')
print(f'Total de imagens para teste: {total}')
print(f'Total de imagens classificadas corretamente: {correct_results}')
print(f'Porcentagem de imagens classificadas corretamente: {round((correct_results/total) * 100, 2)}%')
```

```
Total de imagens para treinamento: 65
Total de imagens para teste: 26
Total de imagens classificadas corretamente: 26
Porcentagem de imagens classificadas corretamente: 100.0%
```

**Fonte:** Compilação do próprio autor

**Figura 22 -** Representação da porcentagem de acerto utilizando imagens com boa luminosidade, com e sem expressões faciais

```
# porcentagem de acerto do algorítimo
test_files = os.listdir(yale_faces_test_path)

total = len(test_files)
correct_results = 0

for test_file in test_files:
  expect_result = int(test_file.split('.')[0].replace('subject', ''))

  face = get_faces(f'{yale_faces_test_path}/{test_file}')[0]
  result, dist = lbph_classifier.predict(face)

  if result == expect_result:
    correct_results += 1

print(f'Total de imagens para treinamento: {len(os.listdir(yale_faces_train_path))}')
print(f'Total de imagens para teste: {total}')
print(f'Total de imagens classificadas corretamente: {correct_results}')
print(f'Porcentagem de imagens classificadas corretamente: {round((correct_results/total) * 100, 2)}%')
```

```
Total de imagens para treinamento: 65
Total de imagens para teste: 39
Total de imagens classificadas corretamente: 34
Porcentagem de imagens classificadas corretamente: 87.18%
```

**Fonte:** Compilação do próprio autor

**Figura 23 -** Representação da porcentagem de acerto utilizando imagens com boa e baixa luminosidade, com e sem expressões faciais

```
# porcentagem de acerto do algorítimo
test_files = os.listdir(yale_faces_test_path)

total = len(test_files)
correct_results = 0

for test_file in test_files:
  expect_result = int(test_file.split('.')[0].replace('subject', ''))

  face = get_faces(f'{yale_faces_test_path}/{test_file}')[0]
  result, dist = lbph_classifier.predict(face)

  if result == expect_result:
    correct_results += 1

print(f'Total de imagens para treinamento: {len(os.listdir(yale_faces_train_path))}')
print(f'Total de imagens para teste: {total}')
print(f'Total de imagens classificadas corretamente: {correct_results}')
print(f'Porcentagem de imagens classificadas corretamente: {round((correct_results/total) * 100, 2)}%')
```

```
Total de imagens para treinamento: 65
Total de imagens para teste: 65
Total de imagens classificadas corretamente: 49
Porcentagem de imagens classificadas corretamente: 75.38%
```

**Fonte:** Compilação do próprio autor

## 7.2 Comunicação com a API

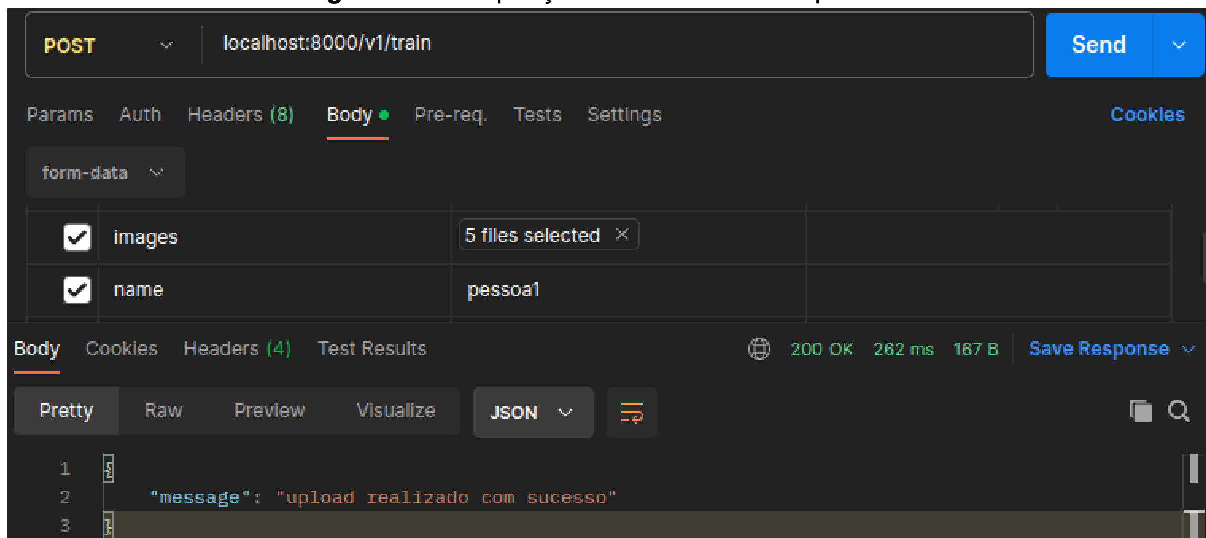
Para demonstrar o funcionamento da *API*, foi utilizado a ferramenta *Postman* para realizar as requisições e enviar os arquivos para treinamento do algoritmo e para busca das pessoas presentes na imagem.

### 7.3.1 Upload das Imagens para Treinamento

Para enviar as imagens para a *API*, duas pessoas e cinco imagens de cada foram selecionadas.

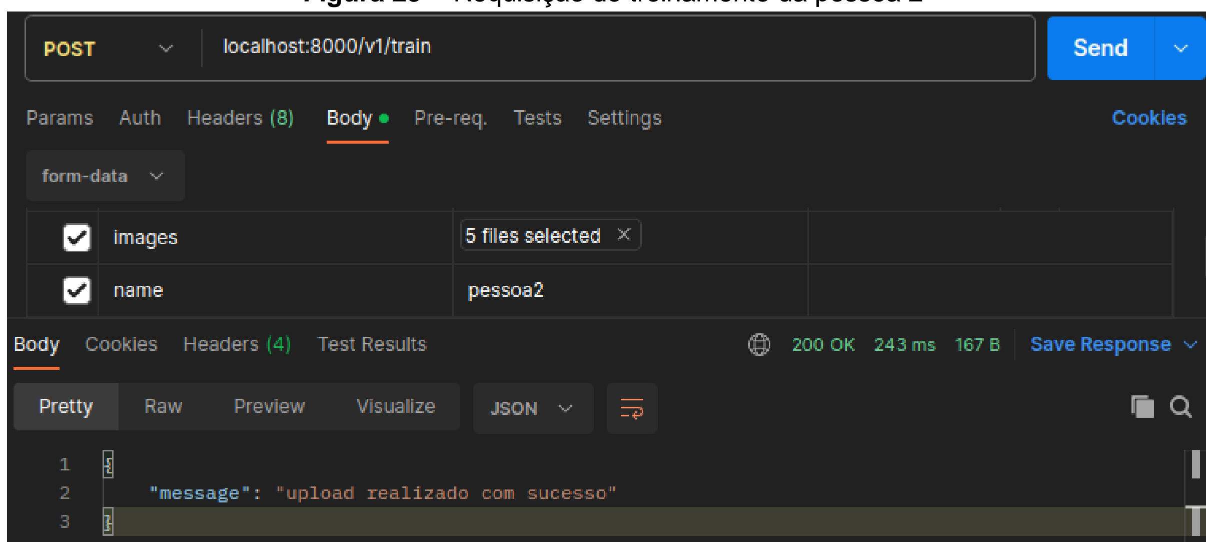
Conforme demonstrado nas Figuras 24 e 25, a primeira pessoa foi denominada de “pessoa1” e a segunda de “pessoa2”. Após enviar a requisição, a mensagem “*upload* realizado com sucesso” é retornada, indicando que as imagens foram salvas e enviadas ao *worker* para treinar o algoritmo.

**Figura 24 -** Requisição de treinamento da pessoa 1




**Fonte:** Compilação do próprio autor

**Figura 25 -** Requisição de treinamento da pessoa 2



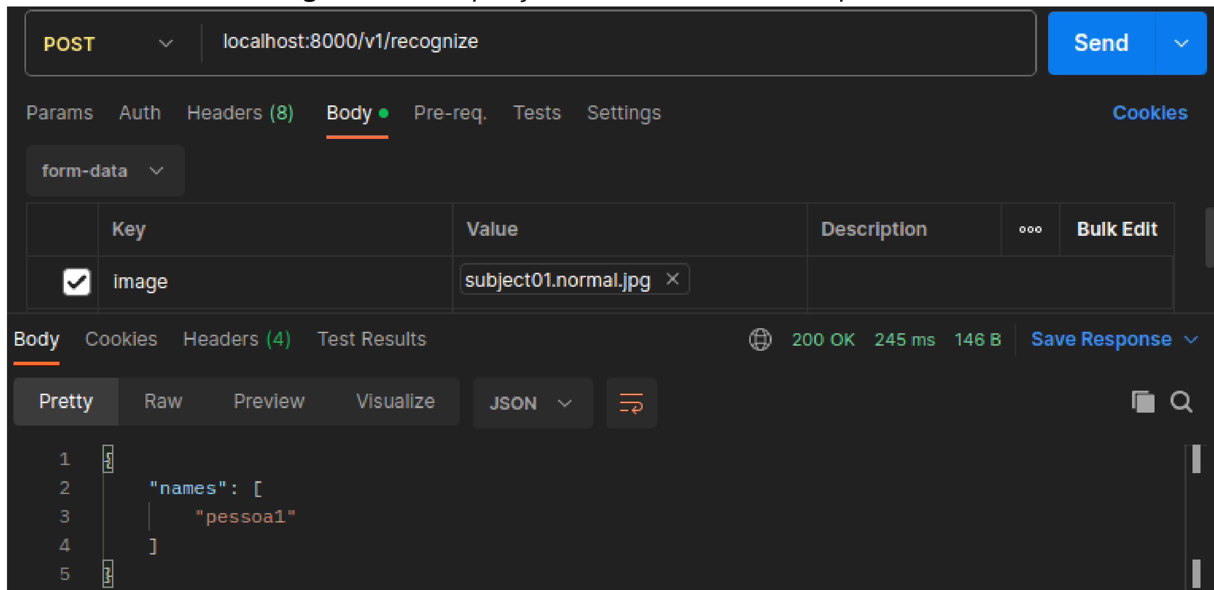

**Fonte:** Compilação do próprio autor

### 7.3.2 Upload das Imagens para Busca

Para realizar a busca das pessoas, três cenários foram preparados. O primeiro contém uma imagem representando a “pessoa1”, o segundo a “pessoa2” e o terceiro uma imagem que contém ambas as pessoas, a fim de demonstrar a capacidade do projeto de encontrar mais de uma pessoa na mesma imagem.

Conforme demonstrado na Figura 26, uma imagem contendo a “pessoa1” foi enviada por *upload* e a *API* retornou o nome dela.

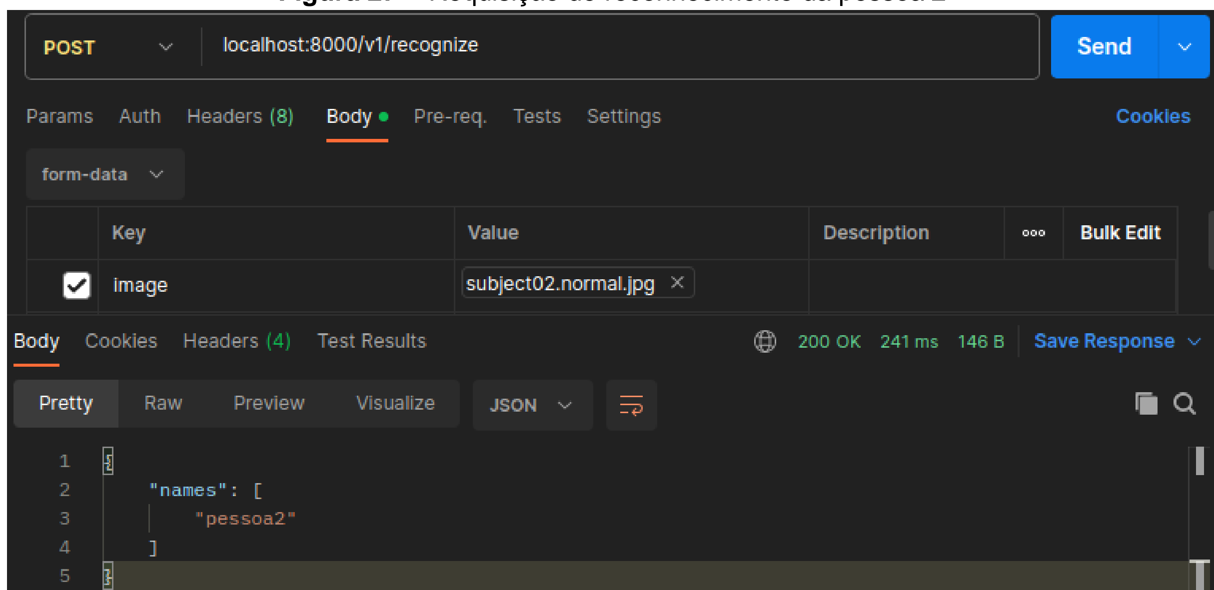
**Figura 26 -** Requisição de reconhecimento da pessoa 1




**Fonte:** Compilação do próprio autor

Conforme demonstrado na Figura 27, uma imagem contendo a “pessoa2” foi enviada por *upload* e a *API* retornou o nome dela.

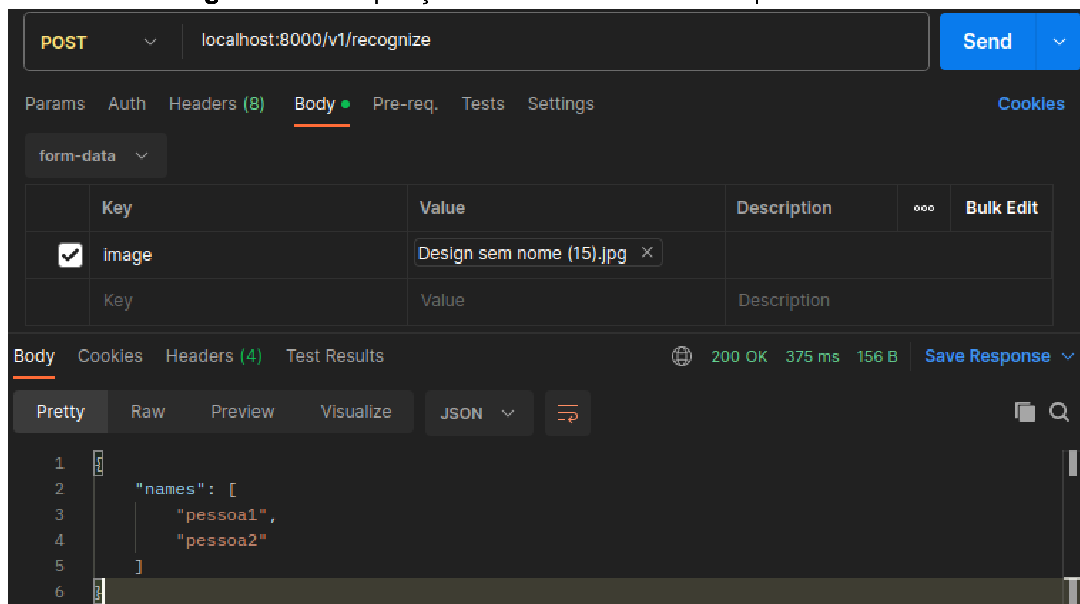
**Figura 27 -** Requisição de reconhecimento da pessoa 2




**Fonte:** Compilação do próprio autor

Conforme demonstrado na Figura 28, uma imagem contendo ambas as pessoas, foi enviada por *upload* e a *API* retornou o nome das duas pessoas corretamente.

**Figura 28 -** Requisição de reconhecimento das pessoas 1 e 2




**Fonte:** Compilação do próprio autor

## 8. Conclusão

Tento em vista a necessidade de encontrar pessoas em imagens, o projeto visa utilizar técnicas de inteligência artificial e visão computacional e disponibilizá-lo para que as pessoas sejam capazes de realizar tal fato. O objetivo inicial era de criar uma interface *WEB* que possibilite as pessoas de realizarem *upload* das imagens e mostre para o usuário o nome de quem esta presente. Porém, como diversas pessoas e empresas poderiam usufruir desse projeto e cada uma tem seu padrão de interface e sistema, o objetivo foi alterado para a criação de uma *API*, a qual irá possibilitá-las usufruir e conectar seus sistemas a ela.

O desenvolvimento do projeto se deu de forma bem organizada e estruturada. Ferramentas auxiliadoras como o *Colab* e o *Docker* foram essenciais para criar e testar os algoritmos de forma prática e rápida, além de permitir praticidade na configuração do ambiente de desenvolvimento. As principais documentações do processo de Engenharia de Software foram criadas, a fim de permitir futuras melhorias e funcionalidades.

Visando garantir o máximo de otimização e assertividade possível, diversos testes foram realizados no algoritmo de reconhecimento facial, possibilitando chegar em 100% de acerto em condições favoráveis, como faces de pessoas sem expressões e ambientes com boa iluminação. O objetivo final foi atingido com sucesso. A *API* criada respondeu com sucesso todas as requisições solicitadas e retornou corretamente o nome das pessoas que estavam presentes nas imagens.

Todo o código fonte, bem como as instruções para inicialização do projeto no ambiente local foi disponibilizado em um repositório público no Github

(https://github.com/matheusHGP/face-recognition-backend), permitido que qualquer pessoa com conhecimentos na área de desenvolvimento de *software* consiga executá-lo.

**Referências**

HISTÓRIA DA FOTOGRAFIA. Educamaisbrasil, 29 Set. 2022, Disponível em: https://www.educamaisbrasil.com.br/enem/artes/historia-da-fotografia. Acesso em 09 ago. 2023.

Estudo da IBM destaca crescimento na adoção global de Inteligência Artificial (IA). Inforchannel, 29 Set. 2022, Disponível em: https://inforchannel.com.br/2022/09/29/estudo-ibm-destaca-crescimento-na-adocao-global-de-inteligencia-artificial-ia/. Acesso em 09 ago. 2023.

GONZALES, R. C.; WOODS, R.E. Processamento Digital de Imagens. 3. ed. São Paulo. Pearson. 2010.

SCURI, Antonio. Fundamentos da Imagem Digital. Tecgraf, web.tecgraf.puc-rio.br, 1, 1, p. 8 - 95, Janeiro, 1999.

AHONEN, T.; HADID, A.; PIETIKÄINEN, M. Face Recognition with Local Binary Patterns. ResearchGate, researchgate.net, 1, 1, p. 1 - 13, Maio, 2004. Disponível em: https://www.researchgate.net/publication/221304831_Face_Recognition_with_Local_Binary_Patterns. Acesso em: 09 ago. 2023.

WANG, H. Studies advanced in face recognition. ResearchGate, researchgate.net, 1, 1, p. 1 - 7, Junho, 2023. Disponível em: https://www.researchgate.net/publication/372203656_Studies_advanced_in_face_recognition. Acesso em: 09 ago. 2023.

KRIGER, Daniel. O que é Python, para que serve e por que aprender?. KENZIE, 08 Jun. 2022, Disponível em: https://kenzie.com.br/blog/o-que-e-python/. Aceso em 09 ago. 2023.

WILLIANS, Wesley. Como funciona o RabbitMQ?. FullCycle, 19 Jul. 2022, Disponível em: https://fullcycle.com.br/como-funciona-o-rabbitmq/. Acesso em 09 ago. 2023.

SINGH, Parth. Understanding Face Recognition Using LBPH Algorithm. Analytics Vidhya, 12 Jul. 2021, Disponível em: https://www.analyticsvidhya.com/blog/2021/07/understanding-face-recognition-using-lbph-algorithm/#:~:text=Introduction,front%20face%20and%20side%20face.. Acesso em 09 ago. 2023.

ROBERTO, Francilvio. Documento de requisitos de software. Analise de Requisitos, 15 Fev. 2018, Disponível em: https://analisederequisitos.com.br/documento-de-requisitos-de-software/. Acesso em 04 set. 2023.