Relatório do Projeto de Reconhecimento Facial e de Emoções

I. Apresentação da Equipe

O grupo é composto por Duarte Lourenço Nº30013174, Rodrigo Simões Nº30012765 e José Lourenço Nº30013434, sendo cada membro responsável por diferentes etapas do desenvolvimento do Reconhecimento Facial e de Emoções. De seguida serão detalhadas as funções e contribuições de cada integrante:

- **Rodrigo Simões**: Responsável pela criação do Gerenciamento de Imagens e Efeitos (Parte Inicial de poo.py) (1º Parte)
- **José Lourenço:** Responsável pela criação da Análise Facial e Reconhecimento (2º Parte)
- **Duarte Lourenço:** Responsável pela criação da Interface do programa (3º Parte)

II. Desenvolvimento do Projeto

1ºParte

- **Gerenciamento de Imagens e Efeitos (Parte Inicial de poo.py)**

Responsabilidade de gereciamento de imagens:

O gerenciamento de imagens e efeitos visuais é uma parte fundamental do sistema, pois permite a manipulação e o processamento eficiente de arquivos visuais. O sistema implementado oferece funcionalidades importantes que garantem o carregamento, restauração e modificação de imagens, além de tratar erros de forma robusta.

Funcionalidades Implementadas

1. Listar e implementar imagens:

- **Método list_all_images()**:
    - Lista todos os arquivos de imagem no diretório ROOT_FOLDER com extensões válidas (.jpg, .jpeg, .png).
    - Facilita a identificação e seleção de imagens disponíveis para processamento.
- **Método load_image(img_path)**:
    - Carrega a imagem especificada no sistema.
    - Define a imagem carregada como a atual e armazena uma cópia original para futuras restaurações.

2. Restaurar Imagem:

- **Método `restore_image()`**:
    - Restaura a imagem ao estado original.
    - Garante que alterações feitas, como aplicação de efeitos, possam ser revertidas facilmente.

3. Aplicação de efeitos visuais:

- **Método apply_effect(effect)**:
    - Permite a aplicação de efeitos visuais, como:
        - **Pixelate**: Reduz temporariamente a resolução da imagem e a restaura para uma aparência pixelada.
        - **Noise**: Adiciona ruído aleatório à imagem.
    - Retorna a imagem processada com o efeito aplicado.
4. Validação e tratamento de erros:

- Inclui mecanismos para verificar a integridade das imagens antes de carregá-las.
- Mensagens de erro claras são fornecidas quando imagens inválidas ou corrompidas são detectadas.

Funcionalidades Adicionais

Além das funcionalidades principais, o sistema inclui recursos extras que complementam o gerenciamento de imagens:

1. **Análise de Emoções**:

a. **Método analyze_emotions()**:

i. Analisa expressões faciais em imagens usando a biblioteca DeepFace.

ii. Retorna uma avaliação detalhada das emoções identificadas.

2. **Reconhecimento Facial**:
    a. **Método recognize_face()**:

i. Implementa reconhecimento facial utilizando o algoritmo LBPH (Local Binary Patterns Histograms).

ii. Realiza o treinamento com imagens do diretório ROOT_FOLDER e associa rótulos a nomes de pessoas.

Pontos de destaque

1. **Estrutura Modular**:
    a. O design modular facilita a manutenção e a expansão futura do sistema.
2. **Versatilidade na Manipulação de Imagens**:
    a. A possibilidade de restaurar imagens e aplicar diferentes efeitos visuais amplia o potencial de uso em diversas aplicações.
3. **Processamento de Imagens com Efeitos**:
    a. O sistema permite que imagens sejam manipuladas de maneira criativa e dinâmica, mantendo a integridade dos dados originais.

Recomendações

1. **Melhorias na Validação**:
    a. Expandir os mecanismos de validação para garantir que apenas imagens completamente compatíveis sejam processadas.
2. **Interface para Escolha de Efeitos**:
    a. Criar uma interface visual ou de linha de comando para simplificar a seleção e aplicação de efeitos.
3. **Documentação e Testes Automatizados**:
    a. Adicionar documentações detalhadas e desenvolver testes automatizados para assegurar a robustez de cada método.

Conclusão

O gerenciamento de imagens e efeitos visuais implementado oferece um conjunto robusto de funcionalidades para manipulação de imagens. A capacidade de carregar, restaurar e modificar imagens, aliada ao tratamento adequado de erros, garante uma experiência

confiável e flexível para o usuário final. A integração de recursos adicionais, como reconhecimento facial e análise de emoções, complementa a funcionalidade principal, tornando o sistema completo e eficiente.

Demonstração do código:

```python
import cv2
import os
import numpy as np
from tkinter import *
from tkinter import messagebox
from PIL import Image, ImageTk
from deepface import DeepFace

# Caminho raiz onde estão as imagens
ROOT_FOLDER = "Pessoas"

class ImageManager:
    def __init__(self):
        self.img_path = None
        self.original_image = None
        self.current_image = None

    def list_all_images(self):
        """Lista todas as imagens na pasta ROOT_FOLDER."""
        valid_extensions = (".jpg", ".jpeg", ".png")
        image_files = []
        for root, _, files in os.walk(ROOT_FOLDER):
            for file in files:
                if file.lower().endswith(valid_extensions):
                    image_files.append(os.path.join(root, file))
        return image_files

    def load_image(self, img_path):
        """Carrega a imagem selecionada e a define como imagem atual."""
        img = cv2.imread(img_path)
        if img is None:
            raise ValueError("Erro ao carregar a imagem.")
        self.img_path = img_path
        self.original_image = img.copy()
        self.current_image = img.copy()
        return img

```

```python
    def restore_image(self):
        """Restaura a imagem original."""
        self.current_image = self.original_image.copy()
        return self.current_image

    def apply_effect(self, effect):
        """Aplica um efeito à imagem atual."""
        if effect == "pixelate":
            h, w = self.current_image.shape[:2]
            temp = cv2.resize(self.current_image, (w // 10, h // 10), interpolation=cv2.INTER_LINEAR)
            self.current_image = cv2.resize(temp, (w, h), interpolation=cv2.INTER_NEAREST)
        elif effect == "noise":
            noise = np.random.normal(0, 25, self.current_image.shape).astype(np.uint8)
            self.current_image = cv2.add(self.current_image, noise)
        return self.current_image
```

2ºParte

Funcionalidades Implementadas

Análise de Emoções

Utiliza a biblioteca DeepFace para avaliar as emoções em uma imagem carregada.

Processo:

A imagem é salva temporariamente.

O método DeepFace.analyze analisa as emoções, retornando valores como felicidade, tristeza, raiva, entre outros.

A imagem temporária é excluída após a análise.

Reconhecimento Facial com LBPH

Usa o algoritmo Local Binary Patterns Histograms (LBPH) para reconhecer rostos com base em um banco de dados de imagens.

Etapas Principais:

Preparação dos Dados de Treinamento:

As imagens são organizadas em pastas separadas por nome da pessoa.

Cada imagem é convertida para escala de cinza e associada a um rótulo único.

Treinamento do Modelo:

O modelo LBPH é treinado com as imagens e rótulos.

Reconhecimento:

A imagem atual é convertida para escala de cinza.

O modelo tenta prever o rótulo e calcula a confiança da correspondência.

O rótulo é mapeado para o nome da pessoa.

3. Estrutura do Código

Funções Principais:

analyze_emotions: Avalia as emoções na imagem usando DeepFace.

recognize_face: Reconhece a pessoa na imagem atual utilizando LBPH.

prepare_training_data: Prepara imagens e rótulos para o treinamento do modelo.

get_person_name_from_label: Mapeia o rótulo para o nome da pessoa.

Bibliotecas Utilizadas:

DeepFace: Análise de emoções.

OpenCV: Processamento de imagens e reconhecimento facial.

NumPy e os: Manipulação de arrays e gerenciamento de arquivos.

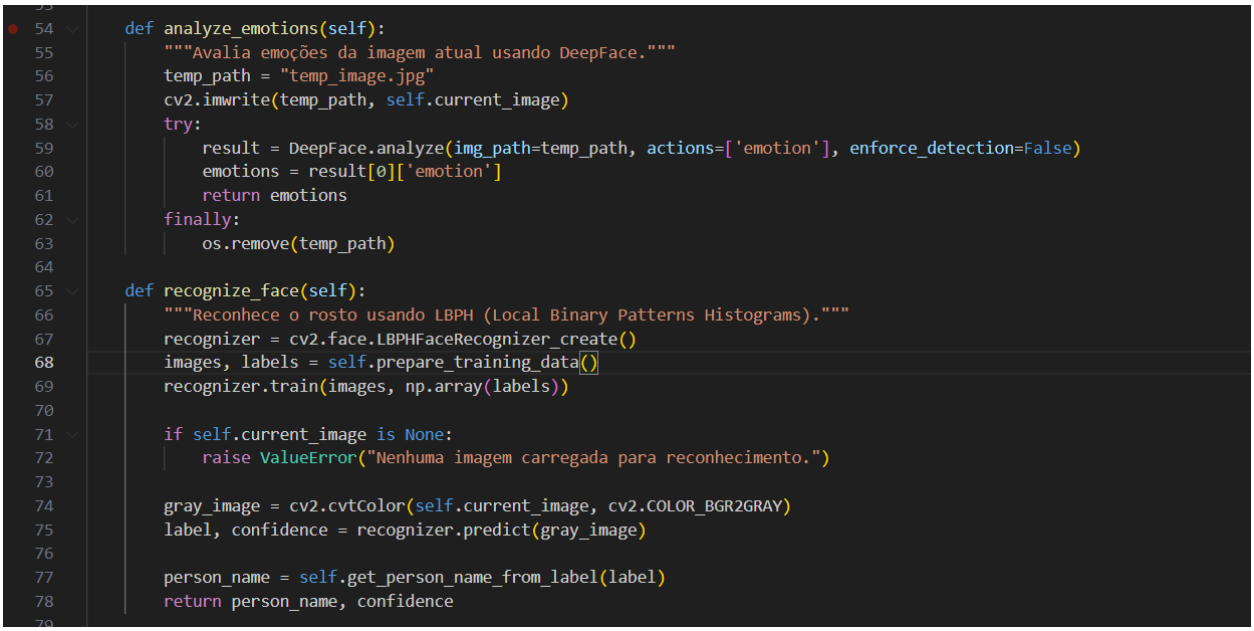
4. Aplicações Práticas

Identificação de indivíduos em sistemas de segurança.

Análise emocional em interações humanas (ex.: suporte ao cliente).

Criação de bases de dados para reconhecimento facial em projetos personalizados.

Demonstração do código:



```python
    def analyze_emotions(self):
        """Avalia emoções da imagem atual usando DeepFace."""
        temp_path = "temp_image.jpg"
        cv2.imwrite(temp_path, self.current_image)
        try:
            result = DeepFace.analyze(img_path=temp_path, actions=['emotion'], enforce_detection=False)
            emotions = result[0]['emotion']
            return emotions
        finally:
            os.remove(temp_path)

    def recognize_face(self):
        """Reconhece o rosto usando LBPH (Local Binary Patterns Histograms)."""
        recognizer = cv2.face.LBPHFaceRecognizer_create()
        images, labels = self.prepare_training_data()
        recognizer.train(images, np.array(labels))

        if self.current_image is None:
            raise ValueError("Nenhuma imagem carregada para reconhecimento.")

        gray_image = cv2.cvtColor(self.current_image, cv2.COLOR_BGR2GRAY)
        label, confidence = recognizer.predict(gray_image)

        person_name = self.get_person_name_from_label(label)
        return person_name, confidence
```

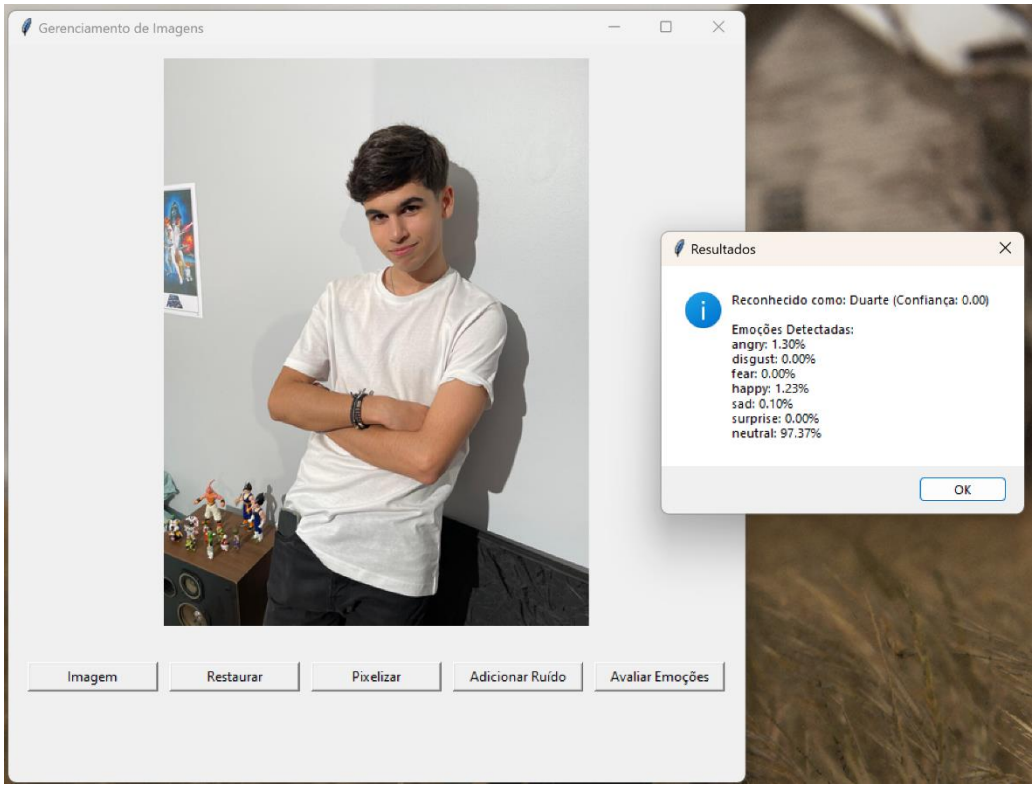
```python
    def prepare_training_data(self):
        """Prepara os dados de treino para o LBPH usando as imagens na pasta 'Pessoas'."""
        images = []
        labels = []
        label_map = {}
        current_label = 0

        for person_folder in os.listdir(ROOT_FOLDER):
            person_path = os.path.join(ROOT_FOLDER, person_folder)
            if os.path.isdir(person_path):
                label_map[current_label] = person_folder
                for img_file in os.listdir(person_path):
                    img_path = os.path.join(person_path, img_file)
                    image = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
                    images.append(image)
                    labels.append(current_label)
                current_label += 1

        self.label_map = label_map
        return images, labels

    def get_person_name_from_label(self, label):
        """Obtém o nome da pessoa a partir do rótulo."""
        return self.label_map.get(label, "Desconhecido")


```

3º Parte

- **Criação da Interface do programa**

Foi desenvolvida uma interface gráfica intuitiva utilizando o Tkinter em Python, que permite ao usuário carregar imagens, aplicar efeitos visuais, restaurar a imagem original e executar a análise de reconhecimento facial e detecção de emoções. O sistema utiliza as bibliotecas OpenCV, DeepFace e PIL para processar as imagens.

Vamos focar especialmente no funcionamento do arquivo da interface gráfica (interface.py) e na interligação deste com o arquivo que contém as funções de manipulação de imagens (poo.py).



- **Objetivo da Interface Gráfica**

A interface gráfica foi desenvolvida com os seguintes objetivos principais:

- Fornecer uma maneira fácil e acessível para o usuário interagir com o sistema de reconhecimento facial.
- Permitir o upload de imagens, a aplicação de efeitos e a exibição dos resultados de reconhecimento facial e detecção de emoções.
- Conectar os botões da interface com as funções do arquivo poo.py.
- Realizar o processamento de imagens e análises diretamente a partir das interações do usuário na interface.

- **Detalhes do Ficheiro interface.py**

O arquivo interface.py contém a classe ImageApp, que é responsável por criar e gerenciar a interface gráfica do usuário. Vamos analisar cada parte do código em detalhe:

Inicialização da Interface

```
class ImageApp:
    Tabnine | Edit | Test | Explain | Document
    def __init__(self, root):
        self.root = root
        self.manager = ImageManager()
        self.panel = None
        self.create_widgets()
```

A interface é criada utilizando a classe ImageApp. Na inicialização, a classe recebe um objeto root, que é a janela principal do Tkinter.

- **self.manager = ImageManager():** Cria uma instância da classe ImageManager do arquivo poo.py, responsável por manipular as imagens carregadas pelo usuário.
- **self.create_widgets():** Chama a função que cria os componentes visuais da interface (botões, labels, etc.).

Função create_widgets

```
def create_widgets(self):
    self.root.title("Gerenciamento de Imagens")
    self.root.geometry("650x650")

    self.panel = Label(self.root)
    self.panel.pack(pady=10)

    button_frame = Frame(self.root)
    button_frame.pack(pady=20)

    Button(button_frame, text="Imagem", command=self.open_image_selection, width=15).grid(row=0, column=0, padx=5)
    Button(button_frame, text="Restaurar", command=self.restore_image, width=15).grid(row=0, column=1, padx=5)
    Button(button_frame, text="Pixelizar", command=lambda: self.apply_effect("pixelate"), width=15).grid(row=0, column=2, padx=5)
    Button(button_frame, text="Adicionar Ruído", command=lambda: self.apply_effect("noise"), width=15).grid(row=0, column=3, padx=5)
    Button(button_frame, text="Avaliar Emoções", command=self.evaluate_emotions, width=15).grid(row=0, column=4, padx=5)
```

Esta função define os componentes da interface, como o título da janela, o tamanho da janela e o painel que exibirá as imagens carregadas. Os botões são criados dentro de um frame, que é um container para organizar os elementos visuais.

Botões Criados:

- **Imagem**: Abre a seleção de imagens.
- **Restaurar:** Restaura a imagem original.
- **Pixelizar:** Aplica um efeito de pixelização na imagem.
- **Adicionar Ruído:** Adiciona ruído aleatório à imagem.
- **Avaliar Emoções:** Executa a análise de emoções e reconhecimento facial.

Cada botão é associado a uma função específica que executa a funcionalidade correspondente.

Função open_image_selection

```
def open_image_selection(self):
    selection_window = Toplevel(self.root)
    selection_window.title("Selecione uma imagem")
    selection_window.geometry("500x400")

    images = self.manager.list_all_images()
    if not images:
        messagebox.showerror("Erro", "Nenhuma imagem encontrada na pasta 'Pessoas'.")
        selection_window.destroy()
        return

    listbox = Listbox(selection_window, selectmode=SINGLE, height=15, width=60)
    for img in images:
        listbox.insert(END, img)
    listbox.pack(pady=10)

    def load_selected_image():
        selected_index = listbox.curselection()
        if not selected_index:
            messagebox.showerror("Erro", "Nenhuma imagem selecionada.")
            return

        img_path = listbox.get(selected_index)
        try:
            img = self.manager.load_image(img_path)
            self.display_image(img)
            selection_window.destroy()
        except ValueError as e:
            messagebox.showerror("Erro", str(e))

    Button(selection_window, text="Carregar", command=load_selected_image).pack(pady=10)
```

Esta função cria uma nova janela para que o usuário possa selecionar uma imagem da pasta **Pessoas**. Ele utiliza a função **list_all_images()** da classe **ImageManager** para listar todas as imagens disponíveis.

- Se nenhuma imagem for encontrada, uma mensagem de erro é exibida.
- Se o usuário selecionar uma imagem e clicar em "Carregar", a imagem é carregada e exibida no painel principal da interface.

Funções de manipulação de Imagem (restore_image e apply_effect)

```python
def restore_image(self):
    img = self.manager.restore_image()
    self.display_image(img)

Tabnine | Edit | Test | Explain | Document
def apply_effect(self, effect):
    img = self.manager.apply_effect(effect)
    self.display_image(img)
```

A interface possui funções para aplicar efeitos visuais na imagem carregada, utilizando as funções disponíveis na classe ImageManager:

- **restore_image():** Restaura a imagem original.
- **apply_effect(effect):** Aplica o efeito selecionado (pixelizar ou adicionar ruído).

Essas funções chamam diretamente as funções implementadas no arquivo poo.py, garantindo que o processamento de imagem ocorra no backend.

Função evaluate_emotions

```python
def evaluate_emotions(self):
    try:
        # DeepFace análise de emoções
        emotions = self.manager.analyze_emotions()
        emotion_text = "\n".join([f"{emotion}: {value:.2f}%" for emotion, value in emotions.items()])

        # LBPH reconhecimento facial
        person_name, confidence = self.manager.recognize_face()
        lbph_text = f"Reconhecido como: {person_name} (Confiança: {confidence:.2f})"

        result_text = f"{lbph_text}\n\nEmoções Detectadas:\n{emotion_text}"
        messagebox.showinfo("Resultados", result_text)
    except Exception as e:
        messagebox.showerror("Erro", f"Erro ao avaliar: {e}")
```

Esta função é a mais importante da interface, pois é responsável por executar as análises de emoções e reconhecimento facial. Ela utiliza duas funções principais do arquivo poo.py:

- **analyze_emotions():** Realiza a análise de emoções usando o **DeepFace**.
- **recognize_face():** Executa o reconhecimento facial utilizando o **modelo LBPH**.

O resultado é exibido em uma messagebox, que mostra as emoções detectadas e a identificação do rosto, junto com o grau de confiança do reconhecimento.

Função display_image

```python
def display_image(self, image):
    img_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    img_pil = Image.fromarray(img_rgb)
    img_pil = img_pil.resize((375, 500), Image.ANTIALIAS)
    img_tk = ImageTk.PhotoImage(img_pil)
    self.panel.config(image=img_tk)
    self.panel.image = img_tk
```

Este método é responsável por exibir a imagem selecionada ou processada na interface. Ele converte a imagem de formato OpenCV para PIL (Python Imaging Library) e, em seguida, para um formato compatível com o Tkinter.

Bloco if __name__ == "__main__":

```python
if __name__ == "__main__":
    root = Tk()
    app = ImageApp(root)
    root.mainloop()
```

Este bloco é fundamental para iniciar a execução da interface gráfica.

- **root = Tk()**: Cria a janela principal da interface gráfica utilizando o Tkinter.
- **app = ImageApp(root)**: Inicializa a aplicação, criando uma instância da classe ImageApp e passando a janela principal como parâmetro.
- **root.mainloop()**: Inicia o loop principal da interface gráfica, que mantém a janela aberta e responde às interações do usuário.

- **Dificuldades Encontradas e Soluções**

Erro ao Avaliar Emoções

- **Problema:**
    - Inicialmente, o método recognize_face() não estava implementado corretamente na classe ImageManager, o que gerava o erro "'ImageManager' object has no attribute 'recognize_face'".
- **Solução:**
    - Foi implementado o método recognize_face() no arquivo poo.py, garantindo que a função pudesse ser utilizada pela interface.

Treinamento Incompleto do Modelo LBPH

- **Problema:**
    - O treinamento do modelo LBPH estava sendo feito apenas com algumas imagens, o que reduzia a precisão do reconhecimento facial.
- **Solução:**
    - O código foi ajustado para treinar o modelo LBPH com todas as imagens presentes na pasta Pessoas.