Reconhecimento Facial

Este código implementa um sistema de detecção facial aprimorado utilizando as bibliotecas OpenCV, NumPy, PIL e Matplotlib. O objetivo é detectar rostos em uma imagem, filtrar aqueles que possuem olhos detectados e exibir e salvar a imagem resultante com os rostos detectados destacados.

Bibliotecas Utilizadas

- **cv2**: Biblioteca OpenCV para processamento de imagens e visão computacional.
- **numpy**: Biblioteca para manipulação de arrays e operações numéricas.
- **PIL (Pillow)**: Biblioteca para manipulação de imagens.
- **matplotlib.pyplot**: Biblioteca para plotagem de gráficos e exibição de imagens.
- **pathlib.Path**: Biblioteca para manipulação de caminhos de arquivos de forma orientada a objetos.

Funções Definidas

1. `load_image(img_path)`

Descrição: Carrega uma imagem a partir de um caminho fornecido, converte-a para o formato RGB e, em seguida, para o formato BGR utilizado pelo OpenCV.

Parâmetros:

- `img_path`: Caminho da imagem a ser carregada.

Retorno:

- Imagem no formato BGR se carregada com sucesso.
- `None` se ocorrer algum erro ao carregar a imagem.

```
def load_image(img_path):
    try:
        pil_img = Image.open(img_path).convert('RGB')
        img = np.array(pil_img)
        return cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    except Exception as e:
        print(f"Error loading image: {e}")
        return None
```

2. `detect_faces(img)`

Descrição: Detecta rostos em uma imagem utilizando o classificador Haar Cascade pré-treinado para detecção frontal de faces.

Parâmetros:

- `img`: Imagem no formato BGR.

Retorno:

- Lista de retângulos delimitadores dos rostos detectados, onde cada retângulo é definido por (x, y, largura, altura).

```
def detect_faces(img):
    gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    face_classifier = cv2.CascadeClassifier(cv2.data.haarcascade
    faces = face_classifier.detectMultiScale(
        gray_image,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE
```

```
    )
    return faces
```

3. `detect_eyes(gray_face)`

Descrição: Detecta olhos em uma região de rosto em escala de cinza utilizando o classificador Haar Cascade pré-treinado para detecção de olhos.

Parâmetros:

- `gray_face`: Região do rosto em escala de cinza.

Retorno:

- `True` se pelo menos um olho for detectado.
- `False` caso contrário.

```
def detect_eyes(gray_face):
    eye_classifier = cv2.CascadeClassifier(cv2.data.haarcascades
    eyes = eye_classifier.detectMultiScale(gray_face)
    return len(eyes) >= 1
```

4. `filter_faces_with_eyes(img, faces)`

Descrição: Filtra os rostos detectados, mantendo apenas aqueles nos quais pelo menos um olho é detectado.

Parâmetros:

- `img`: Imagem original no formato BGR.
- `faces`: Lista de retângulos dos rostos detectados.

Retorno:

- Lista de retângulos dos rostos válidos após a filtragem.

```
def filter_faces_with_eyes(img, faces):
    valid_faces = []
    gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    for (x, y, w, h) in faces:
        face_region = gray_image[y:y+h, x:x+w]
        if detect_eyes(face_region):
            valid_faces.append((x, y, w, h))
    return valid_faces
```

5. `draw_faces(img, faces, color=(0, 255, 0))`

Descrição: Desenha retângulos ao redor dos rostos detectados na imagem.

Parâmetros:

- `img`: Imagem na qual desenhar os retângulos.
- `faces`: Lista de retângulos dos rostos detectados.
- `color`: Cor dos retângulos (padrão é verde).

Retorno:

- Imagem com os retângulos desenhados.

```
def draw_faces(img, faces, color=(0, 255, 0)):
    for (x, y, w, h) in faces:
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
    return img
```

Fluxo Principal do Código

1. **Definição do Caminho da Imagem**:

- O caminho para o diretório contendo as imagens é definido utilizando `Path`.
- São procurados arquivos com extensões `.jpg` e `.png`.

```
img_path = Path(r'C:\Users\luish\OneDrive\Área de Trabalho\FIAP
image_files = list(img_path.glob('*.jpg')) + list(img_path.glob
```

```
if not image_files:
    print("No image files found in the directory.")
    exit()
```

Carregamento da Imagem:

- Seleciona o primeiro arquivo de imagem encontrado e tenta carregá-lo utilizando a função `load_image`.
- Se o carregamento falhar, o programa encerra.

```
img_file = str(image_files[0])
print(f"Trying to load the image: {img_file}")

img = load_image(img_file)
if img is None:
    exit()

print("Image loaded successfully.")
```

Detecção de Rostos:

- Utiliza a função `detect_faces` para detectar rostos na imagem.
- Exibe o número de rostos detectados antes da filtragem.

```
faces = detect_faces(img)
print(f"Faces detected before eye filtering: {len(faces)}")
```

Filtragem de Rostos com Olhos:

- Filtra os rostos detectados para manter apenas aqueles com olhos detectados utilizando a função `filter_faces_with_eyes`.
- Exibe o número de rostos após a filtragem.

```
faces = filter_faces_with_eyes(img, faces)
print(f"Faces detected after eye filtering: {len(faces)}")
```

Desenho dos Retângulos nos Rostos:

- Cria uma cópia da imagem original e desenha retângulos ao redor dos rostos detectados utilizando a função `draw_faces`.

```
img_with_faces = draw_faces(img.copy(), faces)
```

Exibição da Imagem Resultante:

- Utiliza o Matplotlib para exibir a imagem com os rostos detectados.

```
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(img_with_faces, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

Salvamento da Imagem:

- Salva a imagem resultante no diretório de origem com o nome `resultado_deteccao_facial_melhorado.jpg`.

```
output_path = img_path / 'resultado_deteccao_facial_melhorado.jp
cv2.imwrite(str(output_path), img_with_faces)
print(f"Image with face detection saved at: {output_path}")
```

Explicação Detalhada

- **Carregamento e Preparação da Imagem**:
  - A imagem é carregada utilizando o PIL e convertida para um array NumPy.
  - É realizada uma conversão de cores de RGB para BGR, pois o OpenCV utiliza o formato BGR.
- **Detecção de Faces**:
  - A imagem é convertida para escala de cinza para simplificar o processamento.
  - O classificador Haar Cascade para detecção de faces frontais é carregado.
  - A função `detectMultiScale` é utilizada para detectar múltiplos rostos na imagem.
- **Detecção de Olhos em Cada Rosto**:

    - Para cada rosto detectado, é extraída a região correspondente na imagem em escala de cinza.
    - É utilizado o classificador Haar Cascade para detecção de olhos.
    - Apenas rostos com pelo menos um olho detectado são mantidos.
- **Desenho e Exibição dos Resultados**:
    - Retângulos são desenhados ao redor dos rostos válidos na imagem.
    - A imagem resultante é exibida utilizando o Matplotlib, com a conversão de BGR para RGB para exibição correta.
    - A imagem é salva no disco para referência futura.

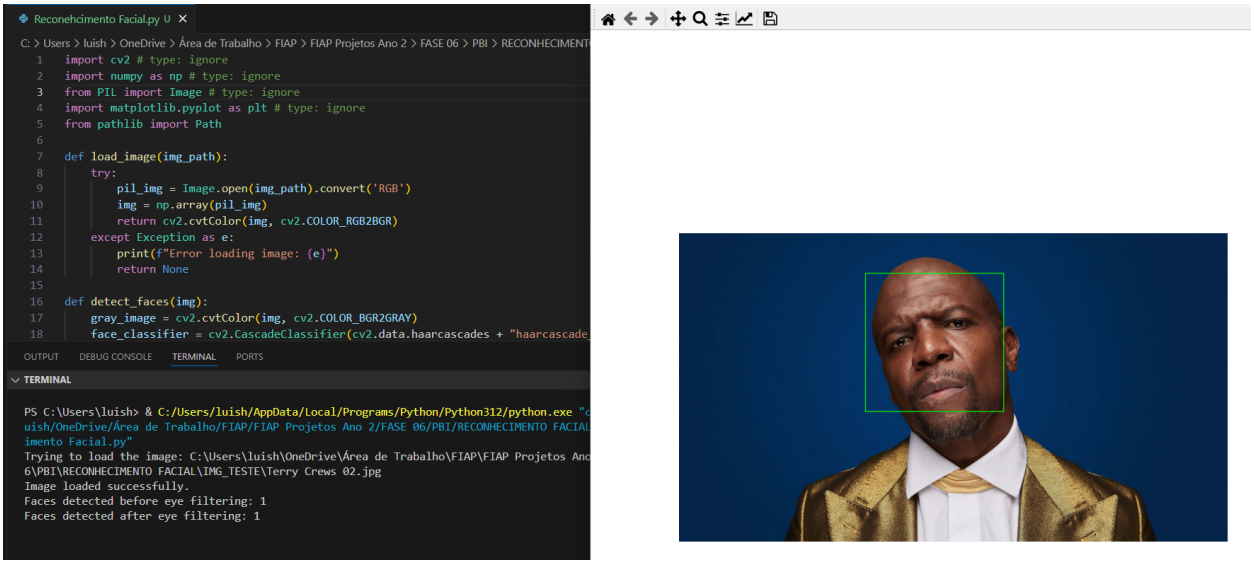
Observações

- **Parâmetros Ajustáveis**:
    - `scaleFactor`: Indica o quanto a imagem é reduzida em cada escala. Valores menores detectam mais faces, mas aumentam o tempo de processamento.
    - `minNeighbors`: Define quantos vizinhos cada retângulo candidato deve ter para retê-lo. Valores maiores resultam em menos detecções, mas aumentam a qualidade.
    - `minSize`: Tamanho mínimo dos objetos a serem detectados. Pode ser ajustado conforme o tamanho esperado dos rostos na imagem.
- **Tratamento de Exceções**:
    - Ao carregar a imagem, possíveis exceções são capturadas e uma mensagem de erro é exibida.
- **Compatibilidade de Caminhos**:
    - O uso de `Path` da biblioteca `pathlib` facilita a manipulação de caminhos de arquivos de forma independente do sistema operacional.
- **Dependências Necessárias**:
    - Certifique-se de que todas as bibliotecas necessárias estejam instaladas:
        - OpenCV (`cv2`)

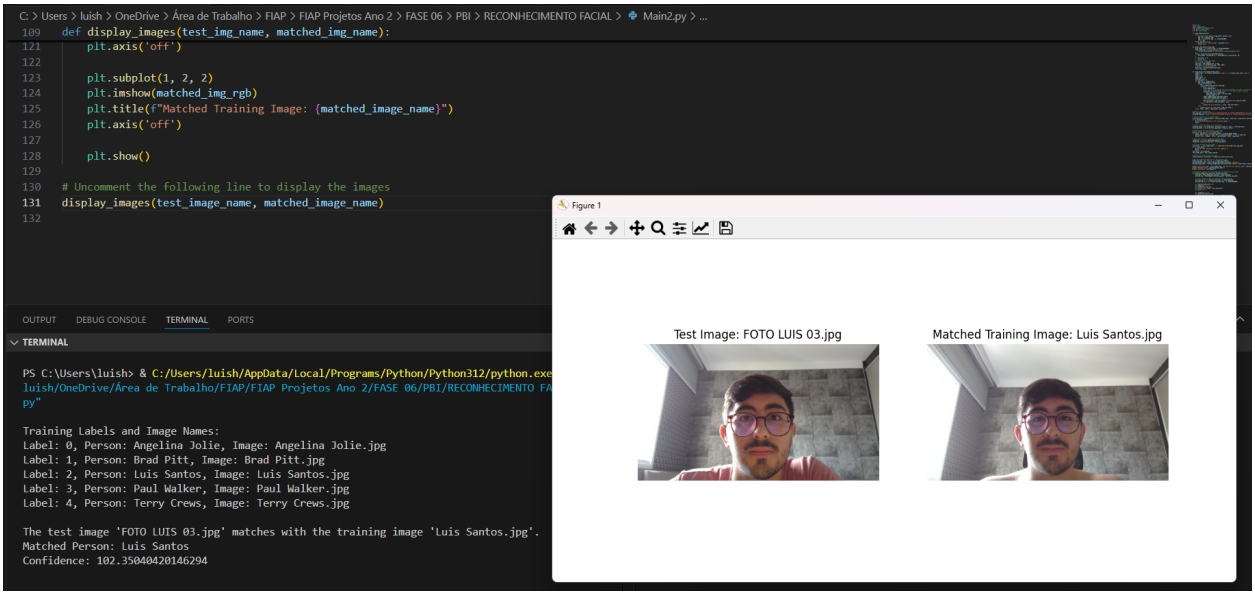
- NumPy
- Pillow (`PIL`)
- Matplotlib

Testes e Resultados

1. Identificação de rostos



2. Teste com Código Final 01



3. Teste com Código Final 02