Relatório: Implementação e Comparação de Algoritmos de Reconhecimento

Instituto Federal Fluminense (IFF)

Campus Macaé

Aluno: Gustavo Lira

Coordenador: Lucas Scotta

1. Introdução

Este relatório apresenta as modificações realizadas no código de reconhecimento facial, substituindo o método baseado na diferença absoluta de pixels pelo algoritmo **LBPH (Local Binary Patterns Histograms)**. Além disso, realizamos uma comparação com a biblioteca **Dlib**, que utiliza redes neurais profundas para reconhecimento facial.

O objetivo dessas mudanças é aprimorar a precisão do reconhecimento, especialmente em cenários com variações de iluminação, e avaliar o desempenho dos dois métodos.

2. O Algoritmo LBPH

2.1 O que é o LBPH?

O **LBPH** é um algoritmo de reconhecimento facial baseado nos padrões locais binários (**LBP**), utilizado para extrair características únicas de uma imagem. Ele se destaca por sua robustez em relação a variações de iluminação e por sua eficiência computacional, o que o torna uma opção viável para aplicações em tempo real.

2.2 Como funciona o LBPH?

O processo do **LBPH** pode ser dividido em três etapas principais:

1. Extração de Características (LBP):

- A imagem é dividida em pequenas regiões (por exemplo, blocos de 3x3 pixels).
- Para cada pixel central de uma região, os pixels vizinhos são comparados com ele.
- Se um pixel vizinho for maior ou igual ao pixel central, ele recebe o valor **1**; caso contrário, recebe o valor **0**.
- Esses valores binários são combinados para formar um número decimal, que representa o **padrão local binário (LBP)** da região.

2. Construção do Histograma:

- Um histograma é gerado para cada região da imagem, contando a frequência de cada padrão LBP.

- Os histogramas de todas as regiões são concatenados, formando um **vetor de características** que representa a imagem.

3. Reconhecimento:

- Durante o treinamento, o algoritmo armazena os vetores de características das imagens conhecidas.
- Durante o reconhecimento, o vetor de características da imagem de teste é comparado com os vetores armazenados utilizando uma métrica de distância (como **distância euclidiana** ou **chi-quadrado**).
- A imagem de teste é então associada à imagem conhecida com a menor distância, determinando assim a identidade do rosto.

2.3 Vantagens do LBPH

- **Robustez à iluminação:** Como o LBPH trabalha com comparações relativas entre pixels, ele é menos sensível a variações de luz.
- **Eficiência computacional:** O cálculo dos padrões binários é simples e rápido, tornando-o ideal para aplicações em tempo real.
- **Simplicidade:** O algoritmo é fácil de implementar e consome menos recursos computacionais em comparação com métodos baseados em redes neurais.

3. Implementação do LBPH no Código

3.1 Mudanças no Código Substituição do Método de Comparação:

O método anterior, que comparava imagens usando a diferença absoluta de pixels, foi substituído pelo LBPH.

O OpenCV fornece uma implementação pronta do LBPH através da classe cv2.face.LBPHFaceRecognizer_create().

Treinamento do Modelo:

As imagens conhecidas são usadas para treinar o modelo LBPH.

O modelo é salvo para evitar a necessidade de retreinamento a cada execução.

Reconhecimento:

Durante o reconhecimento, o modelo LBPH é usado para prever a identidade da face detectada.

3.2 Código Atualizado

```
import cv2
import os
import numpy as np
from datetime import datetime

# Caminhos
path_known_faces = r"C:\Users\gustavo\Desktop\test\imagens_conhecidas"
path_unknown_faces = r"C:\Users\gustavo\Desktop\test\rostos_desconhecidos"

# Inicializar o reconhecedor LBPH
recognizer = cv2.face.LBPHFaceRecognizer_create()

# Listas de rostos conhecidos
known_face_names = []

# Função para carregar rostos conhecidos e treinar o modelo
def load_and_train_known_faces():
    faces = []
    labels = []
    label_ids = {}
    current_id = 0

    for filename in os.listdir(path_known_faces):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(path_known_faces, filename)
            known_image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
            if known_image is None:
                print(f"Erro ao carregar imagem: {filename}")
                continue

            # Redimensionar a imagem para um tamanho padrão
            known_image_resized = cv2.resize(known_image, (150, 150))

            # Adicionar a imagem e o rótulo
            name = os.path.splitext(filename)[0]
            if name not in label_ids:
                label_ids[name] = current_id
                current_id += 1

            faces.append(known_image_resized)
            labels.append(label_ids[name])
            known_face_names.append(name)

    # Treinar o modelo
```

```
    recognizer.train(faces, np.array(labels))

# Carregar e treinar o modelo
load_and_train_known_faces()
if not known_face_names:
    print("Nenhuma face conhecida foi carregada. Verifique a pasta de imagens.")
    exit()

# Inicializar vídeo e detector Haar Cascade
video_capture = cv2.VideoCapture(0)
face_cascade       =       cv2.CascadeClassifier(cv2.data.haarcascades       +
'haarcascade_frontalface_default.xml')

print("Iniciando reconhecimento facial...")

while True:
    ret, frame = video_capture.read()
    if not ret:
        print("Erro ao capturar frame.")
        break

    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Detectar rostos
    faces = face_cascade.detectMultiScale(
        gray_frame, scaleFactor=1.1, minNeighbors=5, minSize=(100, 100)
    )

    for (x, y, w, h) in faces:
        face_image = gray_frame[y:y+h, x:x+w]
        face_image_resized = cv2.resize(face_image, (150, 150))

        # Reconhecer a face usando LBPH
        label, confidence = recognizer.predict(face_image_resized)
        if confidence < 60:
            name = f"{known_face_names[label]} ({100 - confidence:.2f}%)"
        else:
            name = "Desconhecido"
            if not os.path.exists(path_unknown_faces):
                os.makedirs(path_unknown_faces)
            img_name                                                    =
f"{path_unknown_faces}/desconhecido_{datetime.now().strftime('%Y%m%d_%H%M%S')}
.jpg"
            cv2.imwrite(img_name, frame[y:y+h, x:x+w])

        # Exibir informações no quadro
```

```
        color = (0, 255, 0) if name != "Desconhecido" else (0, 0, 255)
        cv2.rectangle(frame, (x, y), (x+w, y+h), color, 2)
        cv2.putText(frame, name, (x, y - 10), cv2.FONT_HERSHEY_DUPLEX, 0.9,
color, 1)

    # Exibe o quadro
    cv2.imshow("Video", frame)

    # Para sair
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Libera recursos
video_capture.release()
cv2.destroyAllWindows()
```

4. Comparação com Dlib

4.1 O que é o Dlib?

O **Dlib** é uma biblioteca de **machine learning** que oferece diversas ferramentas para **detecção e reconhecimento facial**. Ele utiliza **redes neurais profundas pré-treinadas** para extrair **embeddings faciais**, que são comparados com um banco de dados de rostos conhecidos para identificar indivíduos.

4.2 Vantagens do Dlib

- **Alta Precisão:** Graças ao uso de redes neurais, o Dlib proporciona um reconhecimento facial mais preciso, especialmente em **cenários desafiadores**.
- **Robustez a Variações:** O modelo é menos sensível a mudanças na **pose, iluminação e expressões faciais**, tornando-o mais adaptável a diferentes condições.
- **Facilidade de Uso:** A biblioteca fornece APIs intuitivas que simplificam a implementação de **detecção e reconhecimento facial**.

4.3 Desvantagens do Dlib

- **Alto Custo Computacional:** Comparado ao **LBPH**, o Dlib exige **mais poder de processamento**, podendo demandar hardware mais potente para um desempenho eficiente.

- **Dependência de Dados:** A eficácia do Dlib está diretamente ligada à **qualidade e quantidade** dos dados de treinamento utilizados.

4.4 Implementação com Dlib

A implementação com o **Dlib** envolve o uso de um **modelo pré-treinado** para extrair **embeddings faciais**, que são comparados a um banco de dados para identificar rostos. O processo segue uma lógica semelhante ao do **LBPH**, mas substituindo o reconhecedor tradicional pelo modelo baseado em **deep learning** do Dlib.

5. Testes com a Base de Dados Yale

A **base de dados Yale** contém imagens faciais capturadas sob diferentes **condições de iluminação, expressões faciais e poses**. Ela será utilizada para avaliar a eficácia dos algoritmos **LBPH e Dlib**, testando seu desempenho em **cenários desafiadores**.

5.1 Resultados Esperados

- **LBPH:** Deve apresentar **bons resultados** em ambientes com **variações de iluminação**, mas pode enfrentar dificuldades ao lidar com **expressões faciais extremas ou mudanças de pose**.
- **Dlib:** Espera-se que ofereça **maior precisão** do que o LBPH, principalmente em **situações complexas**, mas com um **custo computacional mais elevado**.

6. Conclusão

A substituição do método de comparação baseado em pixels pelo **LBPH** trouxe melhorias significativas, especialmente na **robustez contra variações de iluminação**. No entanto, para aplicações que exigem **alta precisão e maior flexibilidade**, o **Dlib** pode ser a melhor escolha, apesar de sua demanda por **recursos computacionais mais altos**.

Para uma avaliação mais completa, recomenda-se a realização de **testes adicionais com a base de dados Yale**, comparando os desempenhos dos dois métodos em diferentes cenários.

7. Próximos Passos

- **Realizar testes** com a base de dados Yale.
- **Comparar os resultados** do **LBPH e do Dlib** para medir sua eficácia.
- **Otimizar o código** para melhorar a eficiência computacional.
- **Explorar outras técnicas de reconhecimento facial**, como o uso de **redes neurais convolucionais (CNNs)** para aprimorar o desempenho.