

Relatório de desenvolvimento usando LBPH

Instituto Federal Fluminense (IFF)

Campus Macaé

Aluno: Gustavo Lira

Coordenador: Lucas Scotta

Resumo:.....	3
Introdução ao LBPH.....	3
Vantagens do LBPH:.....	3
Limitações do LBPH:.....	3
O programa utiliza as seguintes bibliotecas:	4
OpenCV:	4
dlib:	4
NumPy:.....	4
scikit-learn:	4
OS:.....	4
Como Funciona o Programa O programa é dividido em etapas principais:.....	4
Carregamento das Imagens:.....	4
Detecção e Alinhamento de Faces:	5
Pré-processamento:	5
Treinamento do Modelo LBPH:.....	5
Reconhecimento Facial:	5
Como Funciona o Treinamento.....	6
O treinamento do modelo LBPH envolve as seguintes etapas:	6
Carregamento das Imagens:.....	6
Detecção de Faces:	6
Pré-processamento:	6
Treinamento do Modelo:.....	6
Salvamento do Modelo:.....	6
Problemas e Soluções.....	6
Nenhuma Face Detectada:	6
Baixa Precisão no Reconhecimento:	6
Valores de Confiança Altos:	7

Resumo:

O método de reconhecimento facial foi aprimorado ao substituir a abordagem anterior, que comparava uma única foto da pessoa com a imagem capturada pela webcam. Esse método demandava um processamento de imagem intensivo, o que impactava a eficiência do sistema. A nova abordagem utiliza o algoritmo LBPH (Local Binary Patterns Histograms), que realiza cálculos baseados em histogramas, permitindo um reconhecimento mais preciso e eficiente.

Agora, ao invés de usar apenas uma foto, o sistema utiliza um diretório contendo várias imagens do rosto da pessoa. Isso aumenta significativamente a precisão do reconhecimento facial, pois há mais informações disponíveis sobre as características faciais. Além disso, mesmo com um diretório dedicado para cada pessoa, houve uma redução drástica no uso de processamento de imagem, já que o algoritmo é treinado com os dados antes da execução do programa, tornando-o uma alternativa viável e eficiente.

No entanto, um desafio encontrado foi a qualidade limitada da captura de imagens pela webcam, que possui uma resolução baixa. Isso dificulta o reconhecimento facial devido à falta de detalhes nas imagens capturadas, o que pode comprometer a precisão do sistema em condições fora de laboratório.

Introdução ao LBPH

O LBPH (Local Binary Patterns Histograms) é um algoritmo de reconhecimento facial baseado em características locais da imagem. Ele funciona analisando padrões locais de textura em uma imagem, convertendo-os em histogramas que representam a distribuição desses padrões. O LBPH é amplamente utilizado por sua simplicidade, eficiência e capacidade de reconhecer faces mesmo em condições de iluminação variáveis.

Vantagens do LBPH:

Robustez a variações de iluminação: O LBPH é menos sensível a mudanças na iluminação em comparação com outros métodos.

Simplicidade: Fácil de implementar e requer menos recursos computacionais.

Eficiência: Funciona bem em sistemas com recursos limitados.

Limitações do LBPH:

Dependência da qualidade das imagens: Imagens de baixa qualidade podem reduzir a precisão.

Dificuldade com variações extremas: Pode ter problemas com grandes variações de pose ou expressões faciais.

O programa utiliza as seguintes bibliotecas:

OpenCV:

Para carregar, processar e exibir imagens.

Implementação do algoritmo LBPH (`cv2.face.LBPHFaceRecognizer_create`).

dlib:

Para detecção de faces usando o detector HOG (Histogram of Oriented Gradients).

Oferece alta precisão na detecção de faces.

NumPy:

Para manipulação de arrays e operações matemáticas.

scikit-learn:

Para codificação dos rótulos (nomes das pessoas) em números usando `LabelEncoder`.

OS:

Para navegar pelos diretórios e carregar as imagens.

Como Funciona o Programa O programa é dividido em etapas principais:

Carregamento das Imagens:

As imagens são carregadas de diretórios organizados por pessoa.

Cada subdiretório contém imagens de uma pessoa, e o nome do subdiretório é usado como rótulo.

Detecção e Alinhamento de Faces:

O detector de faces do dlib é usado para localizar rostos nas imagens.

As faces detectadas são cortadas e redimensionadas para um tamanho padrão.

Pré-processamento:

As imagens são convertidas para escala de cinza.

A equalização de histograma é aplicada para melhorar o contraste.

A suavização (blur) é usada para reduzir ruídos.

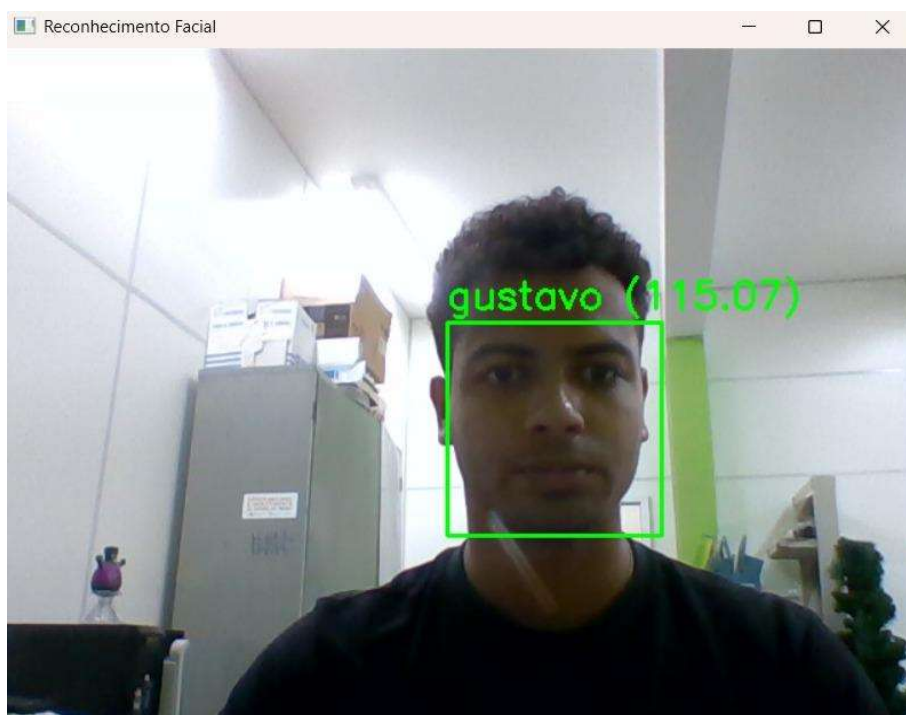
Treinamento do Modelo LBPH:

O modelo LBPH é treinado usando as imagens pré-processadas e seus rótulos correspondentes.

Reconhecimento Facial:

O programa captura frames da webcam, detecta faces e as reconhece usando o modelo treinado.

Se a confiança do reconhecimento for alta, o nome da pessoa é exibido. Caso contrário, a face é marcada como "Desconhecido".



Como Funciona o Treinamento

O treinamento do modelo LBPH envolve as seguintes etapas:

Carregamento das Imagens:

As imagens são carregadas e convertidas para escala de cinza.

Detecção de Faces:

O detector de faces do dlib localiza as regiões das faces nas imagens.

Pré-processamento:

As faces detectadas são redimensionadas e pré-processadas (equalização de histograma e suavização).

Treinamento do Modelo:

O modelo LBPH é treinado usando as faces pré-processadas e seus rótulos correspondentes.

O modelo gera um conjunto de histogramas que representam as características das faces.

Salvamento do Modelo:

O modelo treinado pode ser salvo em disco para uso futuro.

Problemas e Soluções

Nenhuma Face Detectada:

Causa: As imagens não contêm rostos claramente visíveis.

Solução: Verifique as imagens e certifique-se de que os rostos estejam visíveis e bem iluminados.

Baixa Precisão no Reconhecimento:

Causa: Imagens de baixa qualidade ou falta de variedade nas imagens de treinamento.

Solução:

Adicione mais imagens de treinamento com diferentes ângulos e expressões.

Aplique técnicas de pré-processamento, como equalização de histograma e suavização.

Valores de Confiança Altos:

Causa: O modelo não está confiante nas previsões.

Solução:

Reduza o valor de `confidence_threshold`.

Melhore a qualidade das imagens de treinamento.

Codigo :

```
import os
import cv2
import dlib
import numpy as np
from sklearn.preprocessing import LabelEncoder

# Função para carregar as imagens e labels
def load_images_and_labels(data_dir):
    images = []
    labels = []
    for person_name in os.listdir(data_dir):
        person_dir = os.path.join(data_dir, person_name)
        if os.path.isdir(person_dir):
            for image_name in os.listdir(person_dir):
                image_path = os.path.join(person_dir, image_name)
                try:
                    # Carregar a imagem em escala de cinza
                    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
                    if image is not None:
                        images.append(image)
                        labels.append(person_name)
                    else:
                        print(f"[ERRO] Não foi possível carregar a imagem:
{image_path}")
                except Exception as e:
                    print(f"[ERRO] Erro ao processar a imagem {image_path}:
{e}")
    return images, labels

# Função para detectar e alinhar faces
def detect_and_align_faces(images, face_detector, target_size=(150, 150)):
    aligned_faces = []
    valid_indices = [] # Índices das imagens que contêm faces detectadas
    for idx, image in enumerate(images):
        # Detectar faces na imagem
        faces = face_detector(image)
        if len(faces) > 0:
            # Pegar a primeira face detectada
            x, y, w, h = faces[0].left(), faces[0].top(), faces[0].width(),
faces[0].height()
            face_roi = image[y:y+h, x:x+w]
            # Redimensionar a face para o tamanho desejado
            face_roi = cv2.resize(face_roi, target_size)
            aligned_faces.append(face_roi)
            valid_indices.append(idx)
```

```

        else:
            print(f"[AVISO] Nenhuma face detectada na imagem {idx}.")
    return aligned_faces, valid_indices

# Função para aplicar equalização de histograma e suavização
def preprocess_images(images):
    preprocessed_images = []
    for image in images:
        # Aplicar equalização de histograma
        equalized_image = cv2.equalizeHist(image)
        # Aplicar suavização (blur) para reduzir ruído
        blurred_image = cv2.GaussianBlur(equalized_image, (5, 5), 0)
        preprocessed_images.append(blurred_image)
    return preprocessed_images

# Função para treinar o modelo LBPH
def train_lbph_recognizer(images, labels):
    if len(images) == 0:
        raise ValueError("Nenhuma imagem válida foi carregada. Verifique os dados de entrada.")
    recognizer = cv2.face.LBPHFaceRecognizer_create() # Requer opencv-contrib-python
    recognizer.train(images, np.array(labels))
    return recognizer

# Função para reconhecer faces
def recognize_faces(recognizer, face_detector, label_encoder, frame, confidence_threshold=60):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_detector(gray)
    for face in faces:
        x, y, w, h = face.left(), face.top(), face.width(), face.height()
        face_roi = gray[y:y+h, x:x+w]
        # Redimensionar e pré-processar a face detectada
        face_roi = cv2.resize(face_roi, (150, 150))
        face_roi = cv2.equalizeHist(face_roi)
        face_roi = cv2.GaussianBlur(face_roi, (5, 5), 0)
        # Reconhecer a face
        label, confidence = recognizer.predict(face_roi)
        # Se a confiança for menor que o limite, reconhece a pessoa
        if confidence < confidence_threshold:
            person_name = label_encoder.inverse_transform([label])[0]
        else:
            person_name = "Desconhecido"
        # Desenhar retângulo e texto
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

```

```

        cv2.putText(frame, f"{person_name} ({confidence:.2f})", (x, y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
    return frame

# Diretório onde as imagens estão armazenadas
data_dir = r"C:\Users\gustavo\Desktop\testes hub-guard\hub-guard detect\faces"

# Carregar imagens e labels
images, labels = load_images_and_labels(data_dir)

# Verificar se há imagens carregadas
if len(images) == 0:
    print("Nenhuma imagem válida foi carregada. Verifique o diretório e os arquivos.")
    exit()

# Inicializar o detector de faces do dlib
face_detector = dlib.get_frontal_face_detector()

# Detectar e alinhar faces
aligned_faces, valid_indices = detect_and_align_faces(images, face_detector)

# Filtrar os rótulos para incluir apenas os correspondentes às imagens com faces detectadas
labels = [labels[idx] for idx in valid_indices]

# Verificar se há faces detectadas
if len(aligned_faces) == 0:
    print("Nenhuma face foi detectada nas imagens carregadas. Verifique as imagens.")
    exit()

# Pré-processar as imagens (equalização de histograma e suavização)
aligned_faces = preprocess_images(aligned_faces)

# Codificar labels para números
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)

# Treinar o reconhecedor LBPH
try:
    reconhecer = train_lbph_recognizer(aligned_faces, labels_encoded)
    print(f"Modelo treinado com {len(aligned_faces)} imagens.")
except ValueError as e:
    print(e)
    exit()

```

```
# Inicializar a webcam
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Reconhecer faces no frame
    frame = recognize_faces(recognizer, face_detector, label_encoder, frame)

    # Mostrar o frame
    cv2.imshow("Reconhecimento Facial", frame)

    # Parar o loop se a tecla 'q' for pressionada
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Liberar recursos
cap.release()
cv2.destroyAllWindows()
```