



мне необходимо написать программу распознавания лиц алгоритмом DCT. используй самую простую его реализацию. напиши python функцию которая будет получать на вход пути к тестовому изображению, пути к изображениям эталонов и параметры алгоритма, а возвращать ответ - тот же человек или нет.

```
import numpy as np
import cv2
from scipy.fftpack import dct

def face_recognition_dct(test_image_path, reference_images_paths, block_size=8, num_coefficients=100, threshold=0.5):
    """
    Распознавание лиц с использованием DCT.

    Параметры:
    - test_image_path: путь к тестовому изображению
    - reference_images_paths: список путей к эталонным изображениям
    - block_size: размер блока для DCT (по умолчанию 8x8)
    - num_coefficients: количество коэффициентов DCT для сравнения
    - threshold: порог схожести (от 0 до 1)

    Возвращает:
    - True если найдено совпадение, False если нет
    """

    def extract_dct_features(image_path):
        # Загрузка и предобработка изображения
        img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if img is None:
            raise ValueError(f"Не удалось загрузить изображение: {image_path}")

        # Изменение размера для унификации
        img = cv2.resize(img, (128, 128))

        # Разбиение на блоки и применение DCT
        features = []
        h, w = img.shape

        for i in range(0, h - block_size + 1, block_size):
            for j in range(0, w - block_size + 1, block_size):
                block = img[i:i+block_size, j:j+block_size].astype(float)
```

```

        # Применение 2D DCT
        dct_block = dct(dct(block.T, norm='ortho').T, norm='ortho')

        # Извлечение первых коэффициентов (зигзаг-порядок упрощен)
        coeffs = []
        for k in range(min(num_coefficients, block_size)):
            for l in range(min(num_coefficients - k, block_size)):
                if k + l < num_coefficients:
                    coeffs.append(dct_block[k, l])

        features.extend(coeffs[:num_coefficients])

    return np.array(features)

def cosine_similarity(vec1, vec2):
    # Вычисление косинусного сходства
    dot_product = np.dot(vec1, vec2)
    norm1 = np.linalg.norm(vec1)
    norm2 = np.linalg.norm(vec2)
    return dot_product / (norm1 * norm2) if norm1 > 0 and norm2 > 0 else 0

# Извлечение признаков тестового изображения
test_features = extract_dct_features(test_image_path)

# Сравнение с эталонами
for ref_path in reference_images_paths:
    ref_features = extract_dct_features(ref_path)

    # Приведение к одинаковой длине
    min_len = min(len(test_features), len(ref_features))
    similarity = cosine_similarity(test_features[:min_len], ref_features[:min_len])

    if similarity >= threshold:
        return True

return False

# Пример использования:
# result = face_recognition_dct(
#     'test_face.jpg',
#     ['reference1.jpg', 'reference2.jpg'],
#     block_size=8,
#     num_coefficients=15,
#     threshold=0.85
# )
# print("Тот же человек" if result else "Другой человек")

```

Функция работает следующим образом:

Основные этапы

- Загружает изображения в градациях серого и приводит к размеру 128×128
- Разбивает изображение на блоки 8×8 пикселей
- Применяет двумерное дискретное косинусное преобразование к каждому блоку
- Извлекает первые коэффициенты DCT в качестве признаков
- Сравнивает признаки с помощью косинусного сходства
- Возвращает результат на основе порога

Параметры для настройки

- **block_size**: размер блока (обычно 8 или 16)
- **num_coefficients**: количество коэффициентов DCT (больше = точнее, но медленнее)
- **threshold**: порог схожести от 0 до 1 (выше = строже)

Хотите изменить параметры для большей точности или скорости работы?

во-первых, добавь в качестве возвращаемого значения максимальный результат сходства.
во-вторых, напиши алгоритм самостоятельно без использования scipy.

```
import numpy as np
import cv2

def face_recognition_dct(test_image_path, reference_images_paths, block_size=8, num_coef=10):
    """
    Распознавание лиц с использованием DCT.

    Параметры:
    - test_image_path: путь к тестовому изображению
    - reference_images_paths: список путей к эталонным изображениям
    - block_size: размер блока для DCT (по умолчанию 8x8)
    - num_coefficients: количество коэффициентов DCT для сравнения
    - threshold: порог схожести (от 0 до 1)

    Возвращает:
    - (True/False, max_similarity): совпадение и максимальное сходство
    """

    def dct_2d(block):
        """Вычисление 2D DCT для блока"""
        N = block.shape[0]
        M = block.shape[1]
        dct_matrix = np.zeros((N, M))

        for u in range(N):
            for v in range(M):
```

```

        alpha_u = np.sqrt(1/N) if u == 0 else np.sqrt(2/N)
        alpha_v = np.sqrt(1/M) if v == 0 else np.sqrt(2/M)

        sum_val = 0.0
        for i in range(N):
            for j in range(M):
                sum_val += block[i, j] * \
                    np.cos((2*i + 1) * u * np.pi / (2*N)) * \
                    np.cos((2*j + 1) * v * np.pi / (2*M))

        dct_matrix[u, v] = alpha_u * alpha_v * sum_val

    return dct_matrix

def extract_dct_features(image_path):
    # Загрузка и предобработка изображения
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise ValueError(f"Не удалось загрузить изображение: {image_path}")

    # Изменение размера для унификации
    img = cv2.resize(img, (128, 128))

    # Разбиение на блоки и применение DCT
    features = []
    h, w = img.shape

    for i in range(0, h - block_size + 1, block_size):
        for j in range(0, w - block_size + 1, block_size):
            block = img[i:i+block_size, j:j+block_size].astype(float)

            # Применение 2D DCT
            dct_block = dct_2d(block)

            # Извлечение первых коэффициентов (зигзаг-порядок упрощен)
            coeffs = []
            for k in range(min(num_coefficients, block_size)):
                for l in range(min(num_coefficients - k, block_size)):
                    if k + l < num_coefficients:
                        coeffs.append(dct_block[k, l])

            features.extend(coeffs[:num_coefficients])

    return np.array(features)

def cosine_similarity(vec1, vec2):
    # Вычисление косинусного сходства
    dot_product = np.dot(vec1, vec2)
    norm1 = np.linalg.norm(vec1)
    norm2 = np.linalg.norm(vec2)
    return dot_product / (norm1 * norm2) if norm1 > 0 and norm2 > 0 else 0

# Извлечение признаков тестового изображения
test_features = extract_dct_features(test_image_path)

# Сравнение с эталонами

```

```

max_similarity = 0.0

for ref_path in reference_images_paths:
    ref_features = extract_dct_features(ref_path)

    # Приведение к одинаковой длине
    min_len = min(len(test_features), len(ref_features))
    similarity = cosine_similarity(test_features[:min_len], ref_features[:min_len])

    max_similarity = max(max_similarity, similarity)

is_match = max_similarity >= threshold

return is_match, max_similarity

# Пример использования:
# is_match, similarity = face_recognition_dct(
#     'test_face.jpg',
#     ['reference1.jpg', 'reference2.jpg'],
#     block_size=8,
#     num_coefficients=15,
#     threshold=0.85
# )
# print(f"Результат: {'Тот же человек' if is_match else 'Другой человек'}")
# print(f"Максимальное сходство: {similarity:.4f}")

```

Теперь функция возвращает кортеж из двух значений: результат сравнения и максимальное сходство среди всех эталонов, а DCT реализован вручную через двойную сумму косинусов.

алгоритм получает слишком высокое сходство для разных людей, какие параметры мне необходимо изменить?

Для снижения ложных совпадений попробуйте изменить следующие параметры:

Основные настройки

- **threshold:** увеличьте до 0.90-0.95 (вместо 0.85)
- **num_coefficients:** увеличьте до 20-30 (больше признаков = точнее)
- **block_size:** попробуйте 16 вместо 8 (больше информации в блоке)

Дополнительные улучшения

Добавьте нормализацию изображений перед обработкой:

```

# В функции extract_dct_features после resize добавьте:
img = cv2.equalizeHist(img) # Выравнивание гистограммы

```

```
img = (img - np.mean(img)) / np.std(img) # Нормализация
```

Замените косинусное сходство на евклидово расстояние:

```
def euclidean_distance(vec1, vec2):  
    return np.linalg.norm(vec1 - vec2)  
  
# В основном цикле:  
distance = euclidean_distance(test_features[:min_len], ref_features[:min_len])  
# Конвертируйте в сходство: similarity = 1 / (1 + distance)
```

Рекомендуемая комбинация

```
result = face_recognition_dct(  
    'test_face.jpg',  
    ['reference1.jpg', 'reference2.jpg'],  
    block_size=16,  
    num_coefficients=25,  
    threshold=0.92  
)
```

Какой результат сходства вы получаете для разных людей сейчас?

**внеси предложенные изменения в алгоритм.
попробуй оптимизировать его
производительность. он работает очень долго.**

```
import numpy as np  
import cv2  
  
def face_recognition_dct(test_image_path, reference_images_paths, block_size=16, num_coef  
    """  
    Распознавание лиц с использованием DCT (оптимизированная версия).  
  
    Параметры:  
    - test_image_path: путь к тестовому изображению  
    - reference_images_paths: список путей к эталонным изображениям  
    - block_size: размер блока для DCT (по умолчанию 16x16)  
    - num_coefficients: количество коэффициентов DCT для сравнения  
    - threshold: порог схожести (от 0 до 1)  
  
    Возвращает:  
    - (True/False, max_similarity): совпадение и максимальное сходство  
    """  
  
    def create_dct_matrix(N):  
        """Предвычисление DCT матрицы для ускорения"""  
        dct_mat = np.zeros((N, N))  
        for k in range(N):
```

```

        alpha = np.sqrt(1/N) if k == 0 else np.sqrt(2/N)
        for n in range(N):
            dct_mat[k, n] = alpha * np.cos((2*n + 1) * k * np.pi / (2*N))
    return dct_mat

def dct_2d_fast(block, dct_mat):
    """Быстрое вычисление 2D DCT через матричное умножение"""
    return dct_mat @ block @ dct_mat.T

def extract_dct_features(image_path, dct_mat):
    # Загрузка и предобработка изображения
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise ValueError(f"Не удалось загрузить изображение: {image_path}")

    # Изменение размера для унификации
    img = cv2.resize(img, (128, 128))

    # Выравнивание гистограммы и нормализация
    img = cv2.equalizeHist(img)
    img = img.astype(float)
    img = (img - np.mean(img)) / (np.std(img) + 1e-8)

    # Разбиение на блоки и применение DCT
    features = []
    h, w = img.shape

    # Вычисление количества коэффициентов для зигзаг-извлечения
    zigzag_size = int(np.ceil(np.sqrt(num_coefficients * 2)))

    for i in range(0, h - block_size + 1, block_size):
        for j in range(0, w - block_size + 1, block_size):
            block = img[i:i+block_size, j:j+block_size]

            # Применение 2D DCT (быстрая версия)
            dct_block = dct_2d_fast(block, dct_mat)

            # Извлечение коэффициентов в зигзаг-порядке (упрощенно)
            coeffs = []
            for k in range(min(zigzag_size, block_size)):
                for l in range(min(zigzag_size - k, block_size)):
                    if len(coeffs) < num_coefficients:
                        coeffs.append(dct_block[k, l])

            features.extend(coeffs)

    return np.array(features)

def similarity_metric(vec1, vec2):
    """Комбинированная метрика: евклидово расстояние + косинусное сходство"""
    # Косинусное сходство
    dot_product = np.dot(vec1, vec2)
    norm1 = np.linalg.norm(vec1)
    norm2 = np.linalg.norm(vec2)
    cosine_sim = dot_product / (norm1 * norm2 + 1e-8)

```

```

# Нормализованное евклидово расстояние
euclidean_dist = np.linalg.norm(vec1 - vec2)
max_dist = np.sqrt(len(vec1)) * 2 # Примерный максимум
euclidean_sim = 1 - (euclidean_dist / max_dist)

# Среднее двух метрик
return (cosine_sim * 0.7 + euclidean_sim * 0.3)

# Предвычисление DCT матрицы
dct_mat = create_dct_matrix(block_size)

# Извлечение признаков тестового изображения
test_features = extract_dct_features(test_image_path, dct_mat)

# Сравнение с эталонами
max_similarity = 0.0

for ref_path in reference_images_paths:
    ref_features = extract_dct_features(ref_path, dct_mat)

    # Приведение к одинаковой длине
    min_len = min(len(test_features), len(ref_features))
    similarity = similarity_metric(test_features[:min_len], ref_features[:min_len])

    max_similarity = max(max_similarity, similarity)

is_match = max_similarity >= threshold

return is_match, max_similarity

# Пример использования:
# is_match, similarity = face_recognition_dct(
#     'test_face.jpg',
#     ['reference1.jpg', 'reference2.jpg'],
#     block_size=16,
#     num_coefficients=25,
#     threshold=0.92
# )
# print(f"Результат: {'Тот же человек' if is_match else 'Другой человек'}")
# print(f"Максимальное сходство: {similarity:.4f}")

```

Ключевые оптимизации

- **Предвычисление DCT матрицы:** вместо вычисления косинусов в циклах создается матрица один раз
- **Матричное умножение:** DCT вычисляется через $A @ B @ A.T$, что в ~100 раз быстрее
- **Комбинированная метрика:** использует косинусное сходство (70%) и евклидово расстояние (30%) для лучшей точности
- **Нормализация изображений:** выравнивание гистограммы и стандартизация уменьшают ложные совпадения

Насколько быстрее работает теперь?