perplexity

у меня есть несколько вычисляющих векторы признаков функций. напиши программу. используй из этих функций только способ вычисления дискретного косинусного преобразования. НЕ ИЗМЕНЯХ ИХ АЛГОРИТМ РАБОТЫ - он корректен.

1. программа имеет графический интерфейс, где можно выбрать директорию, которая содержит НЕСКОЛЬКО директорий с фотографиями. 1 директория внутри родительской (выбранной) содержит несколько фото одного человека.
2. перед нажатием кнопки старт должны быть настройка параметров алгоритма. я хочу чтобы стандартно было задано: порог - 0.7, размер блока - 8, число коэффициентов - 24
3. затем, по нажатии кнопки старт должна запускаться обработка всех фото и формирование векторов признаков для каждого человека. эти векторы признаков необходимо хранить, чтобы затем иметь возможность распознавать людей.
4. затем должна появляться возможность загрузить любое изображение (эту процедуру должна быть возможность выполнить многократно для одного запуска программы), которое затем сравнивается со всеми доступными векторами признаков и определяется, какому человеку принадлежит фото.
5. я хочу, чтобы при загрузке фото отображалось, на какой эталон больше всего похоже фото. графически отображались два фото рядом друг с другом!
6. программа должна иметь встроенную функцию тестировать саму себя. дополнительная кнопка - на вход принимается директория с изображениями. названия файлов в ней соответствуют названиям директорий с эталонами. например, фото ref-1.jpg точно относится к человеку, фото которого лежат в директории ref-1. должен вычисляться процент правильно распознанных фото.

функции, которые ты должен взять за основу:

```
import numpy as np
import cv2

def face_recognition_dct(
    test_image_path,
    reference_images_paths,
    block_size=16,
    num_coefficients=25,
```

```
    threshold=0.92,
):
    """
    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}")

```
import numpy as np
import cv2
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from PIL import Image, ImageTk
import os
from pathlib import Path
import threading


class FaceRecognitionApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Система распознавания лиц - DCT")
        self.root.geometry("1200x800")

        self.reference_features = {}
        self.reference_sample_paths = {}
        self.dct_mat = None
        self.processing = False

        self.create_widgets()
```

```
    def create_widgets(self):
        # Фрейм параметров
        params_frame = ttk.LabelFrame(self.root, text="Параметры алгоритма", padding=10)
        params_frame.pack(fill="x", padx=10, pady=5)

        ttk.Label(params_frame, text="Порог:").grid(row=0, column=0, padx=5, pady=5, stic
        self.threshold_var = tk.DoubleVar(value=0.7)
        ttk.Entry(params_frame, textvariable=self.threshold_var, width=10).grid(row=0, co

        ttk.Label(params_frame, text="Размер блока:").grid(row=0, column=2, padx=5, pady=
        self.block_size_var = tk.IntVar(value=8)
        ttk.Entry(params_frame, textvariable=self.block_size_var, width=10).grid(row=0, c

        ttk.Label(params_frame, text="Число коэффициентов:").grid(row=0, column=4, padx=5
        self.num_coeffs_var = tk.IntVar(value=24)
        ttk.Entry(params_frame, textvariable=self.num_coeffs_var, width=10).grid(row=0, c

        # Фрейм обучения
        train_frame = ttk.LabelFrame(self.root, text="Обучение", padding=10)
        train_frame.pack(fill="x", padx=10, pady=5)

        ttk.Button(train_frame, text="Выбрать директорию с эталонами", command=self.selec
        self.ref_dir_label = ttk.Label(train_frame, text="Не выбрана")
        self.ref_dir_label.pack(side="left", padx=5)

        ttk.Button(train_frame, text="Старт обучения", command=self.start_training).pack(

        self.train_status_label = ttk.Label(train_frame, text="")
        self.train_status_label.pack(side="left", padx=5)

        # Фрейм распознавания
        recog_frame = ttk.LabelFrame(self.root, text="Распознавание", padding=10)
        recog_frame.pack(fill="x", padx=10, pady=5)

        ttk.Button(recog_frame, text="Загрузить изображение для распознавания", command=s
        self.recog_status_label = ttk.Label(recog_frame, text="")
        self.recog_status_label.pack(side="left", padx=5)

        # Фрейм тестирования
        test_frame = ttk.LabelFrame(self.root, text="Тестирование", padding=10)
        test_frame.pack(fill="x", padx=10, pady=5)

        ttk.Button(test_frame, text="Выбрать директорию для тестирования", command=self.t
        self.test_status_label = ttk.Label(test_frame, text="")
        self.test_status_label.pack(side="left", padx=5)

        # Фрейм результатов
        results_frame = ttk.LabelFrame(self.root, text="Результаты сравнения", padding=10
        results_frame.pack(fill="both", expand=True, padx=10, pady=5)

        # Контейнер для изображений
        images_container = ttk.Frame(results_frame)
        images_container.pack(fill="both", expand=True)

        # Тестовое изображение
```

```
        test_img_frame = ttk.Frame(images_container)
        test_img_frame.pack(side="left", fill="both", expand=True, padx=5)
        ttk.Label(test_img_frame, text="Тестовое изображение").pack()
        self.test_image_label = ttk.Label(test_img_frame)
        self.test_image_label.pack(fill="both", expand=True)

        # Эталонное изображение
        ref_img_frame = ttk.Frame(images_container)
        ref_img_frame.pack(side="left", fill="both", expand=True, padx=5)
        ttk.Label(ref_img_frame, text="Наиболее похожий эталон").pack()
        self.ref_image_label = ttk.Label(ref_img_frame)
        self.ref_image_label.pack(fill="both", expand=True)

        # Текст результата
        self.result_text = tk.Text(results_frame, height=5, wrap="word")
        self.result_text.pack(fill="x", padx=5, pady=5)

    def create_dct_matrix(self, N):
        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(self, block, dct_mat):
        return dct_mat @ block @ dct_mat.T

    def extract_dct_features(self, image_path, dct_mat, block_size, num_coefficients):
        img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if img is None:
            raise ValueError(f"Не удалось загрузить изображение: {image_path}")

        img = cv2.resize(img, (128, 128))
        img = cv2.equalizeHist(img)
        img = img.astype(float)
        img = (img - np.mean(img)) / (np.std(img) + 1e-8)

        features = []
        h, w = img.shape
        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]
                dct_block = self.dct_2d_fast(block, dct_mat)

                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(self, vec1, vec2):
        dot_product = np.dot(vec1, vec2)
        norm1 = np.linalg.norm(vec1)
        norm2 = np.linalg.norm(vec2)
        cosine_sim = dot_product / (norm1 * norm2 + 1e-8)

        euclidean_dist = np.linalg.norm(vec1 - vec2)
        max_dist = np.sqrt(len(vec1)) * 2
        euclidean_sim = 1 - (euclidean_dist / max_dist)

        return cosine_sim * 0.7 + euclidean_sim * 0.3

    def select_reference_directory(self):
        directory = filedialog.askdirectory(title="Выберите директорию с эталонами")
        if directory:
            self.reference_directory = directory
            self.ref_dir_label.config(text=os.path.basename(directory))

    def start_training(self):
        if not hasattr(self, 'reference_directory'):
            messagebox.showerror("Ошибка", "Выберите директорию с эталонами")
            return

        if self.processing:
            messagebox.showwarning("Предупреждение", "Обработка уже выполняется")
            return

        def train():
            self.processing = True
            self.train_status_label.config(text="Обучение...")

            try:
                block_size = self.block_size_var.get()
                num_coefficients = self.num_coeffs_var.get()

                self.dct_mat = self.create_dct_matrix(block_size)
                self.reference_features = {}
                self.reference_sample_paths = {}

                person_dirs = [d for d in Path(self.reference_directory).iterdir() if d.i

                for person_dir in person_dirs:
                    person_name = person_dir.name
                    image_files = list(person_dir.glob("*.jpg")) + list(person_dir.glob('

                    if not image_files:
                        continue

                    person_features = []
                    for img_path in image_files:
                        try:
                            features = self.extract_dct_features(str(img_path), self.dct_
                            person_features.append(features)
                        except Exception as e:
                            print(f"Ошибка обработки {img_path}: {e}")
```

```
                if person_features:
                    self.reference_features[person_name] = person_features
                    self.reference_sample_paths[person_name] = str(image_files[0])

            self.train_status_label.config(text=f"Обучение завершено. Загружено {len(
        except Exception as e:
            messagebox.showerror("Ошибка", f"Ошибка обучения: {str(e)}")
            self.train_status_label.config(text="Ошибка обучения")
        finally:
            self.processing = False

    threading.Thread(target=train, daemon=True).start()

def recognize_face(self):
    if not self.reference_features:
        messagebox.showerror("Ошибка", "Сначала выполните обучение")
        return

    image_path = filedialog.askopenfilename(title="Выберите изображение", filetypes=[
    if not image_path:
        return

    try:
        block_size = self.block_size_var.get()
        num_coefficients = self.num_coeffs_var.get()
        threshold = self.threshold_var.get()

        test_features = self.extract_dct_features(image_path, self.dct_mat, block_siz

        max_similarity = 0.0
        best_match = None

        for person_name, features_list in self.reference_features.items():
            for ref_features in features_list:
                min_len = min(len(test_features), len(ref_features))
                similarity = self.similarity_metric(test_features[:min_len], ref_feat

                if similarity > max_similarity:
                    max_similarity = similarity
                    best_match = person_name

        # Отображение результатов
        self.display_comparison(image_path, best_match, max_similarity, threshold)

    except Exception as e:
        messagebox.showerror("Ошибка", f"Ошибка распознавания: {str(e)}")

def display_comparison(self, test_path, best_match, similarity, threshold):
    # Загрузка и отображение тестового изображения
    test_img = Image.open(test_path)
    test_img.thumbnail((400, 400))
    test_photo = ImageTk.PhotoImage(test_img)
    self.test_image_label.config(image=test_photo)
    self.test_image_label.image = test_photo
```

```
        # Загрузка и отображение эталонного изображения
        if best_match and best_match in self.reference_sample_paths:
            ref_img = Image.open(self.reference_sample_paths[best_match])
            ref_img.thumbnail((400, 400))
            ref_photo = ImageTk.PhotoImage(ref_img)
            self.ref_image_label.config(image=ref_photo)
            self.ref_image_label.image = ref_photo

        # Текстовый результат
        is_match = similarity >= threshold
        result_text = f"Результат: {'Совпадение' if is_match else 'Не совпадает'}\n"
        result_text += f"Наиболее похож на: {best_match if best_match else 'Неизвестно'}\
        result_text += f"Сходство: {similarity:.4f}\n"
        result_text += f"Порог: {threshold:.4f}"

        self.result_text.delete(1.0, tk.END)
        self.result_text.insert(1.0, result_text)

    def test_system(self):
        if not self.reference_features:
            messagebox.showerror("Ошибка", "Сначала выполните обучение")
            return

        test_directory = filedialog.askdirectory(title="Выберите директорию для тестирова
        if not test_directory:
            return

        def test():
            self.test_status_label.config(text="Тестирование...")

            try:
                block_size = self.block_size_var.get()
                num_coefficients = self.num_coeffs_var.get()
                threshold = self.threshold_var.get()

                test_files = list(Path(test_directory).glob("*.jpg")) + list(Path(test_di

                correct = 0
                total = 0

                for test_file in test_files:
                    true_label = test_file.stem

                    try:
                        test_features = self.extract_dct_features(str(test_file), self.dc

                        max_similarity = 0.0
                        best_match = None

                        for person_name, features_list in self.reference_features.items()
                            for ref_features in features_list:
                                min_len = min(len(test_features), len(ref_features))
                                similarity = self.similarity_metric(test_features[:min_le

                                if similarity > max_similarity:
                                    max_similarity = similarity
```

```
                                    best_match = person_name

                            if best_match == true_label and max_similarity >= threshold:
                                correct += 1

                            total += 1
                        except Exception as e:
                            print(f"Ошибка обработки {test_file}: {e}")

                    accuracy = (correct / total * 100) if total > 0 else 0

                    self.test_status_label.config(text=f"Точность: {accuracy:.2f}% ({correct}
                    messagebox.showinfo("Результаты тестирования", f"Точность: {accuracy:.2f}

                except Exception as e:
                    messagebox.showerror("Ошибка", f"Ошибка тестирования: {str(e)}")
                    self.test_status_label.config(text="Ошибка тестирования")

            threading.Thread(target=test, daemon=True).start()


if __name__ == "__main__":
    root = tk.Tk()
    app = FaceRecognitionApp(root)
    root.mainloop()
```

Программа полностью использует ваш алгоритм DCT без изменений и предоставляет графический интерфейс со всеми запрошенными функциями:

Основные возможности

- Настройка параметров алгоритма перед запуском
- Обучение на директории с эталонными изображениями
- Распознавание загруженных изображений с визуальным сравнением
- Автоматическое тестирование системы с расчетом точности
- Многократная загрузка изображений для распознавания без перезапуска

Какие параметры вы планируете использовать для реальных задач распознавания?