Отчет студента группы ИСТ-412 Кузина Ивана

По проекту „Обнаружение и распознание лиц с помощью машинного зрения“

Цель проекта

Создать программу, которая сможет обнаруживать лица людей и распознавать их на основе имеющегося датасета с помощью машинного зрения.

Личные цели

Честно говоря, идея создания такой программы появилась у меня достаточно неожиданно. В самом начале первого курса я работал над своим проектом на одной из пар, и передо мной сидел парень, который писал программу для обнаружения лиц. Я подумал, что он очень умный, раз делает такую программу. В тот момент я почувствовал себя немного глупым. Но буквально через пять минут я понял, что не стоит расстраиваться. Я подумал: *«Да ну, не стоит унывать. Уверен, что такую программу можно написать на Python за несколько строчек кода. Как-нибудь позже я сделаю то же самое.»*

Собственно, так всё и получилось. Спустя месяц я наткнулся на интересную статью от Яндекса: https://thecode.media/face-train/. Там было всё подробно описано — от и до. Статья оказалась очень интересной, как, впрочем, и сам журнал "КОД". Вдохновившись, я взялся за реализацию.

План отчета

1. **Описание проекта**
2. **Техническая реализация**
3. **Демонстрация работы программы**
4. **Проблемы, с которыми я столкнулся**
5. **Результаты и выводы**

Описание проекта

Программа написана на Python с использованием нескольких популярных библиотек:

- **`cv2`** для получения изображения с камеры.
- **`numpy`** для работы с данными (используется немного).
- **`pickle`** для преобразования данных в нужный формат.
- **`os`** для работы с файловой системой.
- **`face_recognition`** для распознавания лиц на основе датасета.

Для обнаружения лиц я использовал модель `haarcascade_frontalface_default.xml`, которая входит в библиотеку **`cv2`**. А для распознавания лиц была использована модель из библиотеки **`face_recognition`**.

> **Примечание:** Стоит понимать разницу между терминами *обнаружение* и *распознавание*.
>
> - **Обнаружение лица** — это процесс определения, что на изображении есть лицо.
> - **Распознавание лица** — это процесс идентификации, чьё это лицо, на основе предварительно обученной модели.

Если вкратце описать работу программы, то она включает несколько основных компонентов:

1. **`dataset_generation.py`** — создает датасет, фотографируя лицо пользователя.
2. **`train_model.py`** — обучает модель распознавания лиц на основе созданного датасета с использованием `face_recognition`.
3. **`functions.py`** — содержит вспомогательные функции для работы основной программы.
4. **`face_detect.py`** — основной файл программы, запускающий процесс обнаружения и распознавания лиц.

Файлы и структура проекта организованы так, чтобы было легко разобраться и поддерживать код. Теперь давайте подробнее рассмотрим техническую реализацию.

Техническая реализация

Структура проекта

Проект состоит из нескольких исполняемых файлов и каталогов, которые структурируют все необходимые компоненты для работы системы распознавания лиц.

1. **Исполняемые файлы:**
   - Файлы с основной логикой, такие как `dataset_generation.py`, `train_model.py`, `functions.py`, и `face_detect.py`.
2. **Каталоги:**
   - **`dataset/`**: Этот каталог содержит фотографии для создания датасета. Каждое фото используется для обучения модели распознавания лиц.

- **`models/`**: В этой папке находится модель `haara` (классическая модель для распознавания лиц), а также сохраняются все обученные модели в формате `.pickle`.
- **`.venv/`**: Папка с виртуальной средой, в которой установлены все необходимые зависимости и библиотеки для работы проекта.

```
project/
├── dataset/
│   └── (здесь хранятся фотографии для датасета)
├── models/
│   ├── haarcascade_frontalface_default.xml
│   └── (здесь сохраняются обученные модели)
├── .venv/
│   └── (виртуальная среда с установленными зависимостями)
├── dataset_generation.py
├── train_model.py
├── functions.py
└── face_detect.py
```

Рис. 1 Наглядная схема проекта

Код программ

`dataset_generation.py` — Создание датасета

1. **Импорт библиотек и инициализация модели:**
   В самом начале импортируются необходимые библиотеки и загружается модель `haarcascade_frontalface_default.xml`, которая используется для обнаружения лиц:

```python
import cv2
import os

path = os.path.dirname(os.path.abspath(__file__))
detector = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
```

Файл dataset_generation.py. Вставка из кода №1

2. **Подключение камеры и настройка параметров:**
   Программа запрашивает имя пользователя, инициализирует счётчик фотографий `i` и отступы для рамки лица. Затем создаётся объект для работы с камерой:

```python
i = 0
offset = 10
name=input('Введите номер пользователя: ')
video = cv2.VideoCapture(0)
```

Файл dataset_generation.py. Вставка из кода №2

3. **Основной цикл работы камеры:**
   Программа непрерывно захватывает кадры с камеры. Изображение преобразуется в оттенки серого, и модель ищет лица на изображении. Если лица не обнаружены, программа выводит сообщение и продолжает ждать:

```
while True:
    ret, im = video.read()
    gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    faces=detector.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5, minSize=(100, 100))

    if len(faces) == 0:
        print('Лицо не обнаружено, ожидание')
        cv2.imshow('im', im)
        cv2.waitKey(1)
        continue
```

Файл dataset_generation.py. Вставка из кода №3

4. **Обработка лиц:**
   Если лицо найдено, программа:

   - Увеличивает счётчик `i`.
   - Вырезает изображение лица с небольшим запасом.
   - Сохраняет изображение в папку `dataset/`.
   - Отображает рамку вокруг лица и само лицо:

```
for (x, y, w, h) in faces:
    i = i + 1
    face_img = gray[y-offset:y+h+offset,x-offset:x+w+offset]
    if face_img.size > 0:
        cv2.imwrite('dataset/face-'+name+'.'+str(i)+'.jpg', face_img)
        cv2.rectangle(im,(x-offset,y-offset),(x+w+offset,y+h+offset),(225,0,0),2)
        cv2.imshow('im', face_img)
        cv2.waitKey(70)
if i > 200:
    break
```

Файл dataset_generation.py. Вставка из кода №4

5. **Завершение работы:**
   Как только собрано 200 изображений, программа завершает работу с камерой:

```
video.release()
cv2.destroyAllWindows()
```

Файл dataset_generation.py. Вставка из кода №5

`train_model.py` — Создание модели на основе датасета

Этот файл отвечает за обработку фотографий из папки `dataset` и создание модели, которая сможет распознавать лица.

Импорт модулей:

- `os` — для работы с файловой системой.

- `face_recognition` — для загрузки изображений и создания кодировок лиц.
- `pickle` — для сериализации данных (сохранения модели).

```
import os
import face_recognition
import pickle
```

Файл train_model.py. Вставка из кода №6

1. **Проверка наличия папки `dataset`:**
   В начале работы программы проверяется, существует ли папка `dataset`, где должны храниться фотографии для обучения. Если папка не найдена, выводится сообщение об ошибке.

```
def train_model_with_dataset():
    if not os.path.exists('dataset'):
        print('Нет папки dataset')
        return
```

Файл train_model.py. Вставка из кода №7

2. **Создание объекта для хранения кодировок:**
   Программа создает объект `all_encodings`, который будет содержать имена людей и их кодировки.

```
all_encodings = {
    'names': [],
    'encodings': []
}
```

Файл train_model.py. Вставка из кода №8

3. **Загрузка и обработка фотографий из папки `dataset`:**
   Программа перебирает файлы в папке `dataset`. Для каждого изображения с расширениями `.png`, `.jpg`, `.jpeg` выполняется следующее:
   - Загружается изображение с помощью библиотеки `face_recognition`.
   - Получается кодировка лица с изображения. Если кодировка успешно создана:
     - Извлекается имя человека (из имени файла).
     - Кодировка и имя добавляются в объект `all_encodings`.

Файл train_model.py. Вставка из кода №9

```
for image in os.listdir('dataset'):
    if image.endswith(('.png', '.jpg', '.jpeg')):
        face_img = face_recognition.load_image_file(f'dataset/{image}')
        face_encodings = face_recognition.face_encodings(face_img)

        if face_encodings:
            face_enc = face_encodings[0]

            person_name = image.split('.')[0].split('-')[1]

            all_encodings['nar Щелкните, чтобы свернуть диапазон.
            all_encodings['encodings'].append(face_enc)
            print(f"Лицо добавлено для {person_name} из файла: {image}")
```

4. **Сохранение кодировок в файл:**
   После обработки всех фотографий создается файл `all_encodings.pickle`, в который записываются все имена и кодировки лиц.

```
with open('all_encodings.pickle', 'wb') as file:
    file.write(pickle.dumps(all_encodings))
```

Файл train_model.py. Вставка из кода №10

5. **Запуск функции `train_model_with_dataset`:**
   Основная функция `train_model_with_dataset` запускает процесс обучения, обрабатывая датасет и сохраняя кодировки для дальнейшего использования в системе распознавания лиц.

Файл train_model.py. Вставка из кода №11

```
def main():
    train_model_with_dataset()

if __name__ == '__main__':
    main()
```

Итог

Файл `train_model.py` выполняет обработку всех фотографий из папки `dataset`, создаёт список кодировок лиц и сохраняет их в формате `pickle`. Этот файл является промежуточным шагом между созданием датасета и запуском основного кода обнаружения и распознавания лиц.

`functions.py` — функции для распознания лиц

В данном файле содержатся функции, которые отвечают за процесс распознавания лиц на изображениях. Рассмотрим их более подробно.

1. Загрузка обученной модели

Функция `download_model()` загружает обученные кодировки лиц из файла, расположенного в папке `models`. Кодировки сохраняются в формате `pickle` и используются для дальнейшего сравнения с лицами, обнаруженными на новых изображениях. Вот как выглядит код этой функции:

```
def download_model(model):
    with open(f'models/{model}_encodings.pickle', 'rb') as file:
        data = pickle.load(file)

    return data
```

Файл functions.py. Вставка из кода №12

Этот шаг необходим для загрузки данных, которые будут использованы при распознавании лиц в процессе работы программы.

2. Обнаружение и распознавание лиц

Основная функция для распознавания лиц — это `detect_face()`. Она принимает на вход изображение и ранее сохраненные кодировки лиц, а также пороговое значение `tolerance` для определения степени схожести лиц. Алгоритм распознавания выполняется в несколько этапов:

1. **Преобразование изображения в оттенки серого**
   Для ускорения обработки изображение сначала переводится в черно-белый формат:

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Файл functions.py. Вставка из кода №13

2. **Обнаружение лиц с помощью каскада Хаара**
   На изображении выполняется поиск лиц с использованием каскадного классификатора Хаара:

```
faces = faceCascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
match = None
```

Файл functions.py. Вставка из кода №13

   Если лица обнаружены, программа рисует прямоугольники вокруг них и извлекает соответствующие области из изображения.

3. **Извлечение кодировки лица**
   Для каждого обнаруженного лица генерируется его кодировка с использованием библиотеки `face_recognition`:

```
face_encodings = face_recognition.face_encodings(image, known_face_locations=[(y, x+w, y+h, x)])
```

Файл functions.py. Вставка из кода №14

   Кодировка лица представляет собой уникальное числовое значение, которое используется для дальнейшего сравнения с другими лицами.

4. **Сравнение с известными кодировками**
   После получения кодировки лица, она сравнивается с заранее сохраненными кодировками:

```
distances = face_recognition.face_distance(known_encodings['encodings'], face_encoding)

min_distance = min(distances)
min_distance_idx = np.argmin(distances)
```

Файл functions.py. Вставка из кода №15

   Для этого вычисляются "расстояния" между кодировкой нового лица и кодировками из базы данных. Чем меньше расстояние, тем более схожи лица.

5. **Пороговое значение для распознавания**
   Если минимальное расстояние между кодировками меньше или равно заданному порогу `tolerance`, то лицо считается распознанным:

```python
if min_distance <= tolerance:
    match = known_encodings['names'][min_distance_idx]
    print(f'Match found: {match} with distance: {min_distance}')
    return match, min_distance, (x, y, w, h)
```

Файл functions.py. Вставка из кода №16

6. **Обработка неизвестных лиц**
   Если расстояние больше порога, лицо считается неизвестным, и программа выводит сообщение:

```python
print('Unknown face')
return None
```

Файл functions.py. Вставка из кода №17

Этот процесс позволяет системе распознавать лица на изображениях, сравнивать их с уже сохраненными и идентифицировать людей на основе заранее подготовленного набора данных.

Результат работы функции

Функция `detect_face()` возвращает:

- Имя распознанного человека (если лицо найдено и совпадает с известными кодировками).
- Минимальное расстояние между кодировками (для оценки качества совпадения).
- Координаты прямоугольника, в котором было найдено лицо (для отображения на изображении).

Если лицо не найдено или не совпало с известными, функция возвращает `None`.

Итоги

Таким образом, файл `functions.py` реализует важную часть проекта, отвечающую за распознавание лиц. Он использует проверенную методику обнаружения лиц с помощью каскадов Хаара и сравнивает их с обученными кодировками для точного распознавания.

`face_detect.py` — запуск программы

Файл `face_detect.py` отвечает за непосредственно распознавание лиц в реальном времени. В нем используется видеопоток с камеры, чтобы обнаружить лицо и сравнить его с ранее обученными кодировками. Рассмотрим код по шагам.

Описание работы

1. **Загрузка модели** В самом начале запускается функция `download_model('all')`, которая загружает все кодировки, сохраненные ранее в файле `all_encodings.pickle`. Эти кодировки будут использоваться для сравнения с лицами, которые мы будем обнаруживать на изображениях с камеры:

```
known_encodings = download_model('all')
```

Файл face_detect.py. Вставка из кода №17

2. **Запуск видеозахвата** Для захвата видео с камеры используется `cv2.VideoCapture(0)`. Это создает объект для захвата изображения с первой доступной камеры

3. **Цикл обработки каждого кадра** В цикле `while True:` программа читает каждый кадр с камеры. Если кадр не получен, программа завершает работу. На каждом кадре происходит следующее:

    - Картинка передается в функцию `detect_face()` для обнаружения лиц и их сравнения с кодировками:

```
result = detect_face(image, known_encodings)
```

Файл face_detect.py. Вставка из кода №18

    - Если лицо распознано, программа отображает имя распознанного человека рядом с лицом, используя `cv2.putText()`:

Файл face_detect.py. Вставка из кода №19

```
if result and result[0] is not None:
    match, distance, (x, y, w, h) = result
    print(f"Match found: {match}")
    cv2.putText(image, f'{match}', (x + 10, y - 10), font, 1.1, (0, 255, 0), 4)
```

4. **Отображение изображения** Каждое изображение, на котором возможно обнаружено лицо, показывается в окне с помощью `cv2.imshow()`. Если пользователь нажимает клавишу 'q' (код клавиши 113), программа завершает работу:

```
cv2.imshow('im', image)
if cv2.waitKey(20) == 113:
    break
```

Файл face_detect.py. Вставка из кода №20

5. **Закрытие видеопотока** После завершения работы программы видеопоток освобождается, а все окна закрываются:

```
video.release()
cv2.destroyAllWindows()
```

Файл face_detect.py. Вставка из кода №21

Итоги

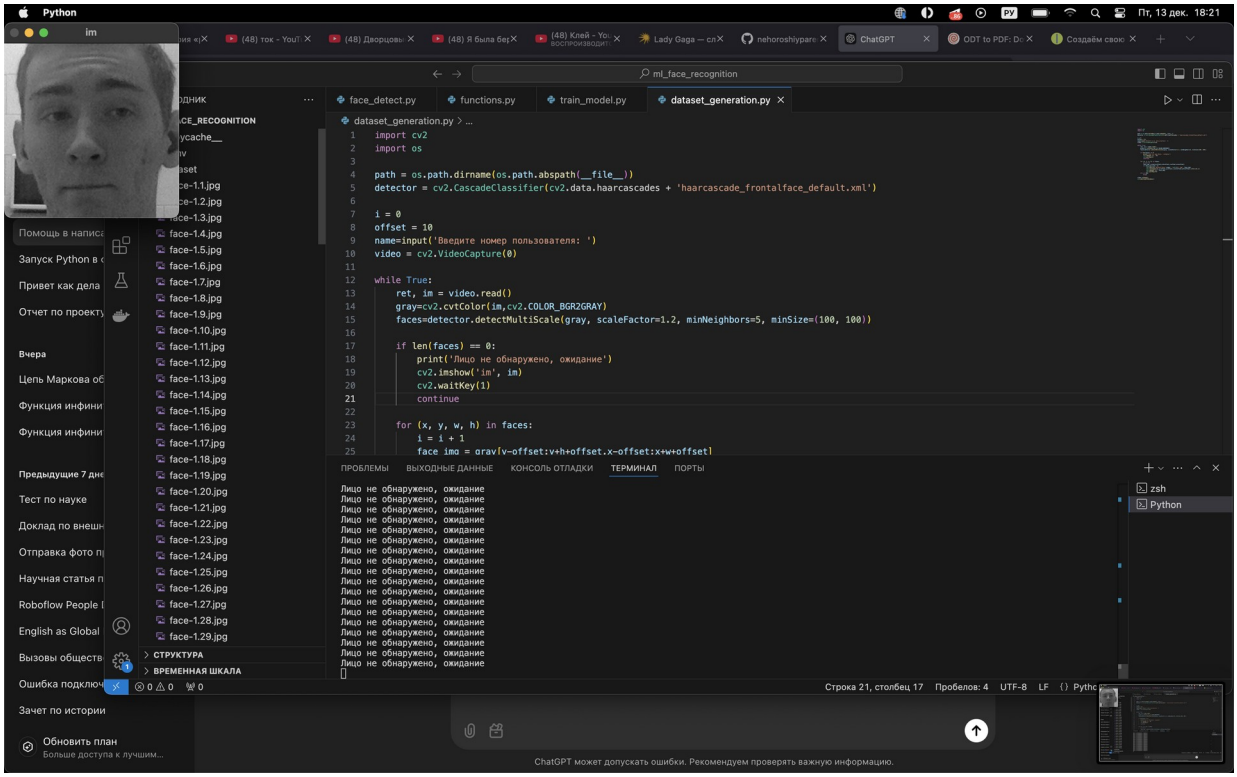
Этот файл запускает процесс распознавания лиц в реальном времени с использованием видеопотока с камеры. Он обращается к уже обученным кодировкам лиц и отображает имя распознанного человека на экране. Таким образом, программа позволяет идентифицировать людей, находящихся в поле зрения камеры, и выводить информацию о них в режиме реального времени.

Демонстрация работы программы

Итак, мы завершили разбор кода программы. Надеюсь, что вы не устали и смогли разобраться в сути.

Теперь давайте быстро пробежимся по демонстрации работы программы.

1. ***Создаем датасет.***
   Для этого запускаем файл `dataset_generation`*, ставим камеру перед своим лицом, и программа делает несколько снимков. Эти фото сохраняются в папку* `dataset`*.*



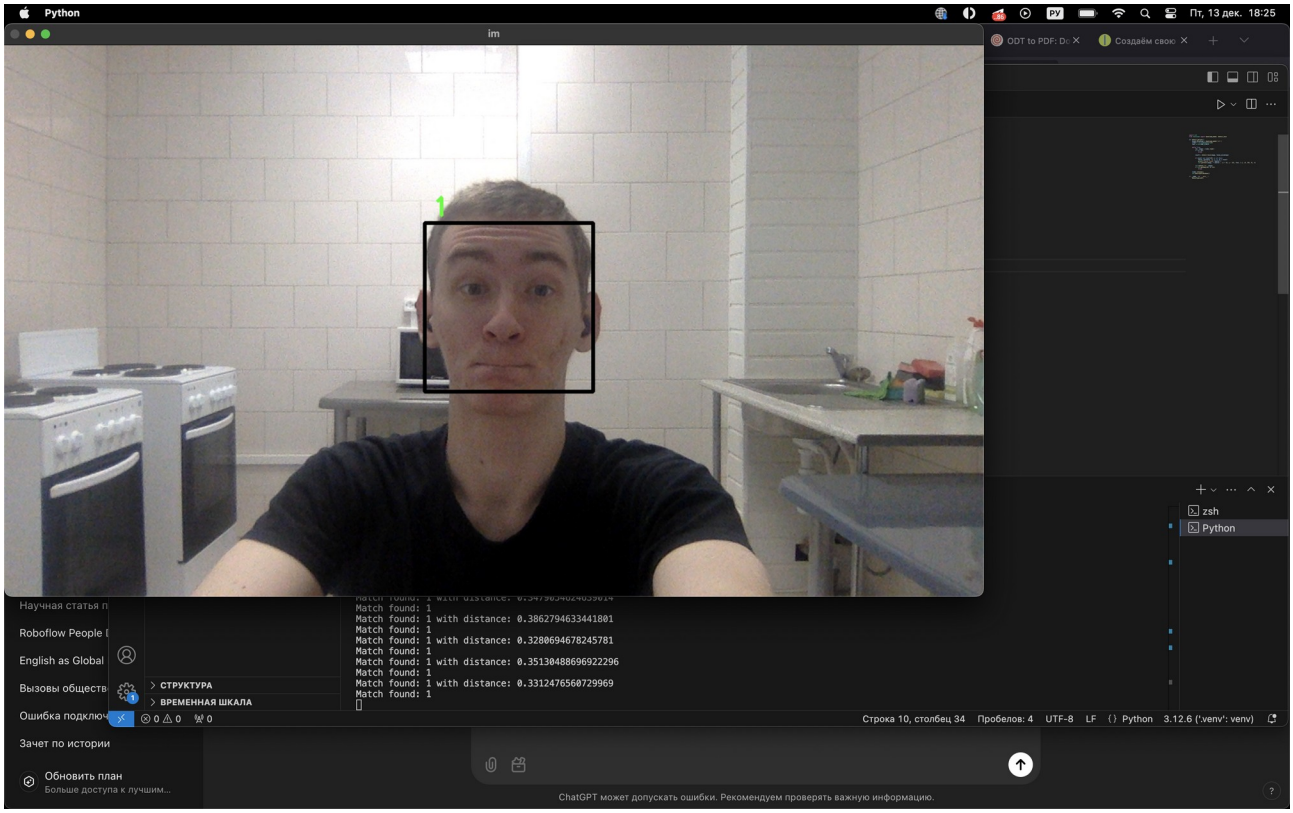
Демонстрация работы программы. Создание датасета

2. **Создаем модель.**
   Следующим шагом запускаем файл `train_model.py`. Он обучает модель на основе созданного нами датасета.

Демонстрация работы программы. Обучение модели

```
Лицо добавлено для 1 из файла: face-1.73.jpg
Лицо добавлено для 1 из файла: face-1.66.jpg
Лицо добавлено для 1 из файла: face-1.72.jpg
Лицо добавлено для 1 из файла: face-1.99.jpg
Лицо добавлено для 1 из файла: face-1.167.jpg
Лицо добавлено для 1 из файла: face-1.173.jpg
Лицо добавлено для 1 из файла: face-1.159.jpg
Лицо добавлено для 1 из файла: face-1.165.jpg
Лицо добавлено для 1 из файла: face-1.171.jpg
Лицо добавлено для 1 из файла: face-1.64.jpg
Лицо добавлено для 1 из файла: face-1.70.jpg
Лицо добавлено для 1 из файла: face-1.58.jpg
Лицо добавлено для 1 из файла: face-1.71.jpg
Лицо добавлено для 1 из файла: face-1.65.jpg
Лицо добавлено для 1 из файла: face-1.170.jpg
Лицо добавлено для 1 из файла: face-1.164.jpg
Лицо добавлено для 1 из файла: face-1.158.jpg
[INFO] Файл all_encodings.pickle успешно создан
```

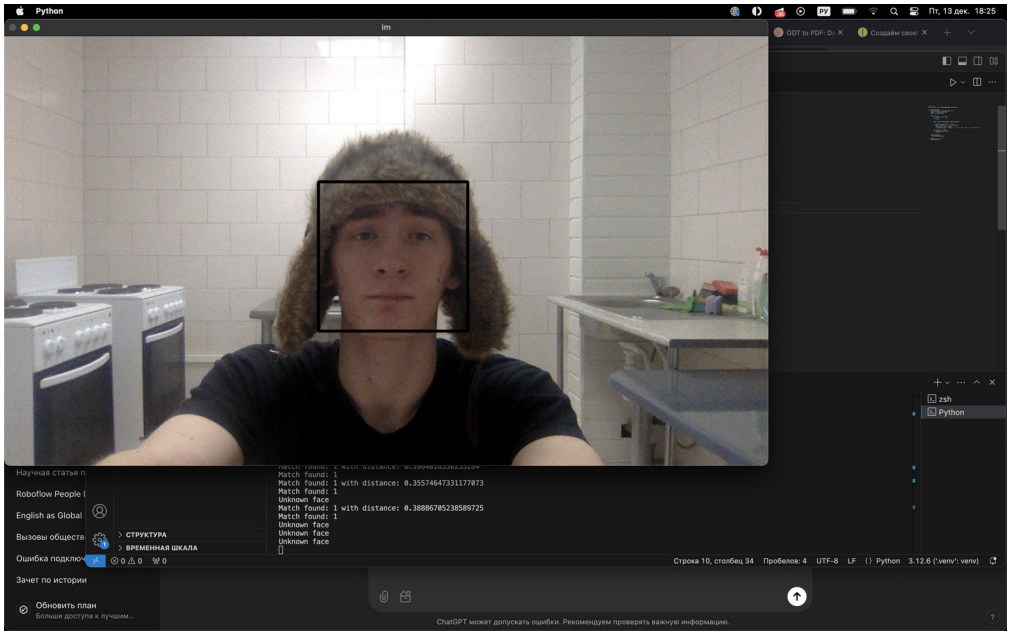
3. **Запускаем распознавание лиц.**
   И наконец, запускаем файл `face_detect.py`, который запускает процесс распознавания лиц в реальном времени.



Демонстрация работы программы. Успешное распознавание

Программа распознает меня довольно хорошо (! ПИШЕТ НАД МОИМ ЛИЦОМ «1» !). Я создал датасет из 100 фотографий, чего вполне достаточно для базового распознавания. Если бы фотографий было больше, распознавание стало бы еще более точным.

Кстати, я заметил интересную вещь: как только я надел шапку, модель стала меня распознавать хуже. Думаю, если увеличить количество фотографий, это поможет избежать таких проблем.



Демонстрация работы программы. Когда я надел шапку, программе стало тяжелее меня распознать

Проблемы, с которыми я столкнулся

В моем проекте была одна серьезная и, как оказалось, ключевая проблема.

Изначально я использовал модель для распознавания из исходного кода статьи — это была всем известная `haarcascade`. Но она, к сожалению, оказалась совершенно неэффективной в плане распознавания конкретных лиц. Поэтому мне пришлось искать другие решения.

В ходе поисков я наткнулся на библиотеку `face_recognition`, которая предложила другой подход к созданию модели. Эффективность этой модели оказалась намного выше, и благодаря ей моя программа заработала так, как я и задумывал.

Кстати, я задумывался о замене `haarcascade` и для обнаружения лиц, но быстро отказался от этой идеи. Она хорошо справляется с этой задачей: работает быстро, не требовательна к ресурсам, и менять ее не было смысла. Проблема заключалась исключительно в распознавании.

Больше никаких серьезных трудностей не возникло. Но я думаю, что эта единственная проблема была крайне важной. Без нее я вряд ли бы так углубился в тему распознавания лиц, и проект мог бы стать одним из тех, которые просто "сделал и забыл". А так — это действительно хороший опыт.

Результаты и выводы

Создавая этот проект, я погрузился в основы машинного зрения и познакомился с библиотекой OpenCV, которая позволяет работать с камерой. Я также узнал о множестве уже существующих моделей для распознавания лиц и получил опыт их использования. Ну и, конечно, пополнил свою коллекцию проектов на Python еще одной работой.

В процессе разработки я наткнулся на фразу, которая меня зацепила:
"Сейчас такое время, когда уже многое создано. Тебе не нужно изобретать с нуля. Гораздо важнее уметь соединять готовые решения и создавать на их основе что-то большее."

Не знаю, является ли это оправданием или просто глупой мыслью, но, мне кажется, в этих словах есть смысл. Этот проект стал для меня подтверждением того, что даже использование готовых инструментов может стать ценным опытом, если правильно их объединить и применить.

Использованные источники:

1. Серия статей на Яндекс КОД: https://thecode.media/face-train/

2. Рандомное видео с ютуба про библиотеку Face recognition: https://www.youtube.com/watch?v=Hk6SZu0V1L8

3. Мой лучший интернет друг: https://chatgpt.com/

ССЫЛКА НА РЕПОЗИТОРИЙ:

https://github.com/nehoroshiyparen/face_recognition