Проект в рамках добровольной помощи поисково-спасательному отряду LizaAlert

Назначение: Данный проект предназначен для загрузки отснятых с квадракоптера фотографий и загрузки их в систему детекции наличия людей на снимках

Папки в проекте:
docs — документация
img — картинки, пояснения к файлам документации
client — скрипты клиентской части
server — скрипты серверной части
etc — файлы конфигурации, если понадобятся

Постановка задачи

Клиентская часть

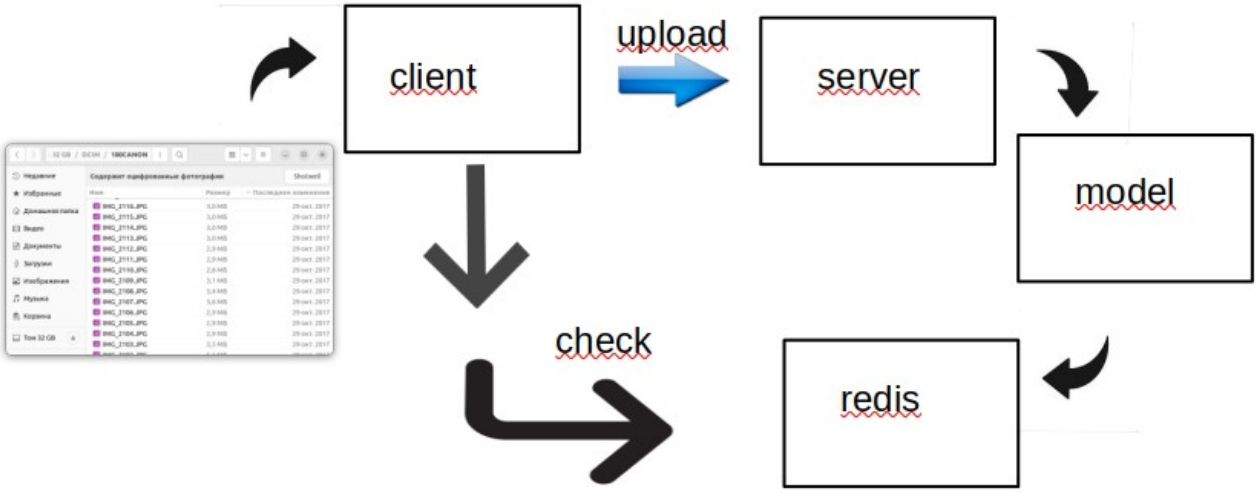
На ноутбуке, обрабатывается или просто записываютсяфотографии с квадрокоптера. Фотографий порядка 4098 шт.

Скрипт клиентской части должен загрузить эти фотографии на сервер и выдать результат, какие из фотографий могут содержать изображение человека. Желательно сделать человекоудобный интерфейс пользователя.

Серверная часть

Скрипт серверной части принимает фотографии, вызывает скрипт модели, которая обрабатывает фотографии и возвращает результат клиентскому скрипту.

Алгоритм взаимодействия скриптов



На рисунке изображен порядок взаимодействия компонентов системы:

1. скрипт загрузчик
- указывается директория с фотками
- скрипт создает UID (просто число) и загружает фотографии по адресу:

api.server.com/upload/uid
- UID может быть в формате REGNUM_TIMESTAMP
- далее опрашивает каждые 1-2 сек сервер по адресу: api.server.com/result/uid

2. скрипт сервера upload.py запускается при обращении метода POST на адрес api.server.com/upload/uid
- загружает фотку в каталог /tmp/uid (настраивается через config.py)
- по окончанию загрузки асинхронно вызывает скрипт modelLoader.py и передает ему в качестве аргумента uid

3. скрипт modelLoader.py считывает по одному имени файла из директории /tmp/uid , отправляет либо саму фотку в модель, либо имя файла.... требуется уточнить АПИ

4. скрипт modelLoader.py получив результат анализа фотографии формирует данные в в виде JSON и заносит их в редис список с ключом UID. Устанавливается время жизни списка в редис 1 час (настраивается в конфиге).

5. бэкграундовский скрипт (демон) каждые 0.5 сек опрашивает редис, если в очереди есть данные, то выбирает список фотографий. Вызывает ML модель, которая осуществляет детекцию изображения. Ключ in

6. бэкграундовский скрипт (демон) пишет в редис в результаты ML предсказания по ключу out_uid

7. Клиентский скрипт опрашивает каждые 1-2 сек сервер по адресу: api.server.com/result/uid

8. скрипт result.py срабатывает на GET запрос api.server.com/result/uid, читает данные из редиса по ключу UID, и если result не равен 0.0 то формирует JSON формата:

```
{ processed : N, count: Nmax, items: [
{photo:IMG_1020.jpg,  x: XX, y: YY }, . . .
]}
```

9. Клиентский скрипт, периодически отправляя GET запрос на адрес api.server.com/result/uid получает JSON. Данные JSON отображаются в клиентской программе. Формируется ссылка или кнопка «посмотреть». *Тут надо подумать как лучше сделать.*

При получении всего обработанного списка: значение processed равно значению count, выдает надпись: Обработка закончилась и кнопка «удалить фото с сервера».

10. По нажатию кнопки удалить отправляется DELETE запрос на адрес api.server.com/uid по которому запускается скрипт delete.py, который удаляет директорию со всеми загруженными фотографиями.