Программа Верификатор паспорта РФ

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Москва, 2023

Содержание:

Назначение

Сервис предназначен для автоматизированного визуального распознавания информации в документах, представленных в виде сканов или фотографий. Распознавание выполняется на ЦПУ с использованием нейросетевых модулей, каждый из которых решает свою профильную задачу. Применение данного сервиса позволяет снизить вовлеченность человека в процесс анализа пакета документов, и, как следствие, добиться:

- снижения количества допускаемых в пакете документов ошибок
- предотвращения попадания сторонних документов в пакет
- ускорения процесса обработки и оцифровки документов
- предотвращения подделки документов
- уменьшения затрачиваемых на проверку человеко-часов

На текущий момент сервис поддерживает распознавание паспорта гражданина Российской Федерации, однако по запросу клиента возможно добавление распознавания произвольных документов, специфичных для конкретного бизнеса.

Основные понятия

- API – программный интерфейс, который позволяет интегрировать сервис с другими информационными системами
- НЕЙРОСЕТЕВОЙ МОДУЛЬ - часть программы, использующая нейронные сети для решения поставленной задачи
- РАСПОЗНАВАНИЕ ТЕКСТА - перевод текста, представленного в виде изображения в каком-либо из допустимых форматов данных для хранения изображений в вычислительных машинах в строку символов
- ДЕТЕКТОР - часть программы, как правило имеющая в своей основе нейросетевой модуль, выполняющая задачу по поиску какого-либо объекта на изображении

Архитектура решения

Решение представляет собой набор микросервисов, каждый выполняющий свою задачу в рамках отдельного контейнера. Микросервисы коммуницируют друг с другом через HTTP-запросы, передавая информацию, необходимую для обработки поступившего пакета документов.

В решении представлены следующие микросервисы:

- Package Manager - принимает на вход пакет документов и управляет распознаванием в нем информации, детектирует и подготавливает документы к дальнейшей обработке, а также возвращает результат обработки клиентскому приложению
- Passport RF - выполняет непосредственное распознавание информации в паспорте гражданина Российской Федерации
- Redis - сервис хранения очередей задач на обработку присланных пакетов документов, задачи хранятся в зашифрованном виде с целью предотвращения свободного чтения данных в случае несанкционированного доступа к хранилищу

В случае необходимости добавления распознавания специфичных для конкретного бизнеса документов добавляется отдельный микросервис, решающий данную задачу.

Список нейросетевых модулей

- ДЕТЕКТОР ДОКУМЕНТОВ – находит документ на изображении, возвращает параметры описывающего документ прямоугольника
- ДЕТЕКТОР ЛИЦ - находит лицо на изображении, возвращает параметры описывающего лицо прямоугольника
- ПРЕОБРАЗОВАТЕЛЬ ПЕРСПЕКТИВЫ – находит углы документа
- ДЕТЕКТОР ТЕКСТА – находит описывающие текст прямоугольники на изображении документа

- РАСПОЗНАВАТЕЛЬ – переводит входное изображение в строку текста
- ВЕРИФИКАТОР – преобразует входное изображение лица в вектор чисел, уникально идентифицирующий изображенного на нем человека

Используемые технологии

Основа решения написана с использованием интерпретируемого языка программирования Python версии 3.9. Также используются следующие дополнительные модули:

- OpenCV - используется для обработки изображений и для постобработки результатов работы нейросетевых модулей
- Numpy - используется для работы с числовыми массивами и матричными операциями
- ONNX Runtime - используется для выполнения обработки используемых нейросетевых модулей
- Cryptography - используется для шифрования данных в процессе обработки
- Async IO - используется для выполнения асинхронных операций
- aiohttp - используется для асинхронного выполнения HTTP-запросов
- Flask - фреймворк для реализации веб-сервиса
- Redis - фреймворк для коммуникации с хранилищем очередей задач
- RQ - используется для работы с очередями задач

Установка

Перечень документации, с которой необходимо ознакомиться

Пользователь должен ознакомиться с данным руководством пользователя, а также по необходимости со следующими ресурсами:

- Docker (https://docs.docker.com/get-started/overview/)
- Docker Compose (https://docs.docker.com/compose/)
- Debian OS (https://www.debian.org/doc/)
- или Ubuntu OS (https://help.ubuntu.com/)
- Kubernetes (в случае использования данного решения в кластере вычислительных машин) ((https://kubernetes.io/ru/docs/home/)

Требования к вычислительной машине

Для эффективной работы решения необходимо

- 8 vCPU
- 12 RAM DDR4
- 48 Гб SSD

Начало работы

В первую очередь необходимо получить доступ к репозиторию docker-образов, на котором и расположено решение. Сам репозиторий расположен в Яндекс.Облаке. Клиенту передается JSON-файл, содержащий ключ авторизации. Чтобы авторизоваться, клиенту нужно ввести следующую команду в bash-терминале:

```
cat key.json | docker login --username json_key --password-stdin cr.yandex
```

После этого клиент может просто запустить приложение. Клиенту заранее передается файл docker-compose.yml, содержащий параметры запуска решения. Используя его, находясь в одной папке с ним, клиент может запустить решение следующей командой:

```
docker-compose up -d
```

После этого нужно будет подождать какое-то время прежде чем будут загружены docker-образы самого решения, после чего оно станет доступным на машине на 80-ом порту. Для выключения сервиса необходимо использовать следующую команду:

```
docker-compose down
```

Порт запуска можно изменить, поменяв соответствующую строку в docker-compose.yml файле. Другие настройки менять не рекомендуется, т.к. это может спровоцировать нарушение работы самого решения.

Ограничения

- Максимальный размер пакета: 16 Мб
- Максимальное количество запросов / минуту: 25-30 шт.

Эксплуатация

Выполнение задачи по обработке пакета документов происходит асинхронно:

1. запросом на одну точку передается пакет документом и тем самым создается задача на обработку (<SERVICE_IP_AND_PORT>/api/v1/docs),

2. затем запросами на другую точку опрашивается состояние пакета в обработке (<SERVICE_IP_AND_PORT>/api/v1/verifications),

Запрос на обработку пакета (он же “создание задачи на обработку”)

Пакет документов передается в формате FormData, документы пакета могут находиться как в одном лишь архиве, так и распределены по самой FormData.

В пакете может также содержаться файл, содержащий метаданные о данном пакете, например уникальный идентификатор пакета (PACKAGE_ID), тип пакета (PACKAGE_TYPE) или версия пакета (PACKAGE_VERSION). Также могут быть указаны документы, присутствие которых ожидается в данном пакете (DOCS).

Примеры запроса на обработку пакета с несколькими файлами

Bash (via curl):

```
curl -X POST \
   -H "Content-type: multipart/form-data" \
   --form 'passport.jpg=@"/home/user/passport.jpg"' \
   --form 'photo.jpeg=@"/home/user/photo.jpeg"' \
   --form 'another_photo.jpeg=@"/home/user/another_photo.jpeg"' \
   '<SERVICE_IP_AND_PORT>/api/v1/docs'
```

JavaScript:

```
<input type="file" id="files" name="document" multiple/>
<script>
    async function handleFileSelect(evt) {
        let files = evt.target.files; // FileList object
        const ctrl = new AbortController() // timeout
        setTimeout(() => ctrl.abort(), 10000);
```

```
        let input = document.getElementById("files");
        let data = new FormData();
        // files is a FileList of File objects. List some properties.
        for (let i = 0, f; f = files[i]; i++) {
            data.append(f.name, input.files[i]);
        }
        fetch('/api/v1/docs', {method: "POST", body: data, signal: ctrl.signal}).then(response
=> response.text())
        .then(data => {
            <RECEIVED_DATA_PROCESSING_CODE>
        })
        .catch(error => { console.log(error) });
    }
    document.getElementById('files').addEventListener('change', handleFileSelect, false);
</script>
```

Пример запроса на обработку пакета файлом с одним архивом

Bash (via curl):

```
curl -X POST \
    -H "Content-type: multipart/form-data" \
    --form 'archive.zip=@"/home/user/archive.zip"' \
    '<SERVICE_IP_AND_PORT>/api/v1/docs'
```

В качестве ответа сервис возвращает ID созданный (или переданный в мета-файле в пакете) идентифицирующий пакет документа в процессе обработки. Используя этот ID можно запрашивать статус обработки данного пакета у сервиса.

Запрос статуса обработки пакета и получение результата

Bash (via curl):

```
curl -X GET \
    '<SERVICE_IP_AND_PORT>/api/v1/verifications?
    packages=<LIST_OF_PACKAGE_IDS_SEPARATED_BY_COMMAS>'
```

Результат обработки приходит в формате JSON (application/json).

```
{
   "<PACKAGE_ID>": {
      "package_meta": {
         "package_id": "<PACKAGE_ID>",
         "processing_status": "<PROCESSED | IN_PROGRESS | NOT_FOUND | ERROR>",
         "docs_summary": [{
               "doc_type": "<DOCUMENT_TYPE>",
               "file_name": "<FILENAME_OF_THE_DOCUMENT>",
               "status": "<EITHER_VALID_OR_ERROR>"
               "info": {
                  "FACE_INFO": [{
                        "probability": <PROBABILITY_FACE_BELONGS_TO_PACKAGE>,
                        "xbr": <XBR_FACE_COORD>,
                        "xtl": <XTL_FACE_COORD>,
                        "ybr": <YBR_FACE_COORD>,
                        "ytl": <YTL_FACE_COORD>
                  }]
               "FAILED_FACE_COMPARISON": [{
                     "filename": "<FILENAME_WHERE_UNEQUAL_FACE_IS>",
                     "score": <FLOAT_COMPARISON_SCORE_OF_THE_FACES>
               }]
         }],
      },
      "docs": [{
            "TYPE": "<DOCUMENT_TYPE>",
            "FILENAME": "<NAME_OF_THE_FILE_CONTAINING_THE_DOCUMENT>",
            "DATA": {
               "<DATA_FIELD>": {
                  "SCORE": <SCORE_OF_CONFIDENCE_AS_FLOAT>,
                  "VALUE": "<TEXT_OF_THE_FIELD>"
               },
               "<DATA_FIELD>": {
                  "SCORE": <SCORE_OF_CONFIDENCE_AS_FLOAT>,
                  "VALUE": "<TEXT_OF_THE_FIELD>"
               },
            },
            "META": {
               "<META_PROPERTY>": <VALUE>
            },
      }]
   }
}
```