

Классификатор изображений фруктов и овощей

Годовой проект магистратуры Искусственный интеллект ВШЭ, первый курс.

Команда:

- [Красюков Александр Сергеевич](#) — (tg: @kas_dev)
- [Мевший Илья Павлович](#) — (tg: @Ilya12)
- [Климашевский Илья Алексеевич](#) — (tg: @ferox_Y)
- [Писаренко Сергей Сергеевич](#) — (tg: @SerejP)

Куратор:

[Тимур Ермешев](#) - (tg: @SofaViking)

Описание проекта:

Задача классификации изображений - одна из классических задач компьютерного зрения, которая не теряет своей актуальности, поскольку в различных приложениях и сервисах требуется классифицировать объекты, изображенные на фотографии. Проект представляет собой сервис, решающий эту задачу.

Описание данных:

Используется [данный датасет](#) В датасете представлены изображения фруктов и овощей. Датасет имеет **33 класса по 1400 изображений в каждом** [Подробнее про данные...](#)

Разведочный анализ (EDA) датасета изображений овощей и фруктов

Был проведен разведочный анализ. В результате был выявлен дублирующий класс: *Strawberry/Strawberries*. Стоит провести эксперименты с разными размерами изображений, чтобы понять, какой размер стоит использовать для обучения модели. [Подробнее про разведочный анализ...](#)

Baseline

В качестве baseline-модели был выбран **метод опорных векторов SVC** с использованием PCA для уменьшения размерности данных. Для выделения признаков были испробованы HOG, SIFT, ResNet. Для baseline был выбран **HOG**, т.к. он выиграл в скорости по сравнению с ResNet и в качестве метрик по сравнению с SIFT. Для обучения использовались цветные изображения размером 64*64px. Т.к. в каждом классе содержится 1400 изображений, то для сбалансированного датасета используется метрика **accuracy**. Также вспомогательной метрикой используется **f1 macro**. [Подробнее про baseline...](#)

Разработка микросервиса

Взаимодействие с полученной моделью реализовано с помощью веб-интерфейса FastAPI и streamlit-приложения.

На данный момент оба модуля упакованы в докер контейнер. Streamlit-приложение отвечает на запросы пользователя, используя функционал сервиса FastAPI. Кроме того, проект развернут на арендованном VPS. Подробности в README.md.

Ниже представлено описание функционала веб-интерфейса Streamlit и FastAPI (демонстрация работы приложений находится в файле README.md).

Веб-интерфейс FastAPI

- **/api/v1/dataset/load (Load Dataset)**
 - Метод: POST
 - Описание: Позволяет загрузить датасет для дальнейшего использования. На вход подается архив, содержащий папки (с названиями классов) с изображениями для каждого класса.. После загрузки архива происходит сбор информации о датасете. Данные о датасете хранятся в модели DatasetInfo: количество изображений в каждом классе количество дубликатов в каждом классе размеры изображений информация о цветах изображений DatasetInfo хранится на сервере и передается в ответе на запрос клиента.
 - Ответ: 200 OK: Возвращает информацию о загруженном датасете, включая количество изображений в каждом классе, информацию о дубликатах, размеры изображений и цветовые характеристики.
 - Ошибки: 400 Bad Request: Если загруженный файл не является ZIP-архивом или произошла ошибка при обработке.
- **/api/v1/dataset/info (Get Dataset Info)**
 - Метод: GET
 - Описание: возвращает информацию о датасете, полученную при загрузке архива в методе Load Dataset. Если датасет загружен на сервер, но не проинициализирован (к примеру, после перезапуска сервера), то данные о датасете собираются в методе Get Dataset Info, иначе - отдается хранящаяся на сервере модель. Если датасет не загружен, то метод выдаст исключение.
 - Ответ: Информация о датасете (DatasetInfo).
 - Код ответа: 200 OK — успешный запрос. 400 Bad Request — датасет не загружен.
- **/api/v1/dataset/samples (Dataset Samples)**

- Метод: GET
 - Описание: Dataset Samples создает изображение с примерами картинок в каждом из классов. В первый раз изображение создается с помощью rpyplot и сохраняется на сервере, потом используется сохраненное изображение для отправки клиенту. Если датасет не загружен, то метод выдаст исключение.
 - Ответ: Стрим изображений в формате PNG.
 - Код ответа: 200 OK — успешный запрос. 400 Bad Request — датасет не загружен.
- /api/v1/models/fit (**fit**)
 - Метод: POST
 - Описание: fit служит для создания новой модели на сервере. В данном методе создается модель с использованием гиперпараметров для PCA и SVC. После создания новой модели есть возможность получить кривые обучения (если указать with_learning_curve = True). Модель обучается 10 секунд, по истечении времени процесс обучения модели прерывается с исключением, чтобы не нагружать сервер тяжелыми моделями. Если модели удалось обучиться, то данные о ней сохраняются в models. Если датасета для обучения нет на сервере, то метод выдаст исключение.
 - Параметры:
 - config (опционально): гиперпараметры модели.
 - with_learning_curve: сохранять ли кривую обучения.
 - name: название модели.
 - Ответ: Информация о созданной модели (ModelInfo).
 - Код ответа: 201 Created — успешное обучение. 400 Bad Request — ошибка обучения. 408 Request Timeout — превышение времени обучения.
- /api/v1/models/list_models (**List Models**)
 - Метод: GET
 - Описание: возвращает все хранящиеся на сервере ранее обученные модели. Также на сервере присутствует заранее загруженная baseline-модель. Информация о моделях возвращается в виде словаря, где ключом является идентификатор модели, а значением информация о модели ModelInfo.
 - Ответ: Словарь с информацией о моделях (dict[str, ModelInfo]).
 - Код ответа: 200 OK
- /api/v1/models/info/{model_id} (**Model Info**)
 - Метод: GET
 - Описание: возвращает пользователю информацию о конкретной модели по указанному идентификатору. Если модели с указанным идентификатором нет на сервере, то выдастся исключение.
 - Ответ: Информация о модели (ModelInfo).
 - Код ответа: 200 OK — успешный запрос. 400 Bad Request — модель не найдена.
- /api/v1/models/load (**Load**)
 - Метод: POST
 - Описание: позволяет сделать активной одну из хранящихся на сервере моделей. У модели существует уникальный идентификатор, по которому модели хранятся в models. Если модели с указанным идентификатором нет на сервере, то выдастся исключение.
- /api/v1/models/predict (**Predict**)
 - Метод: POST
 - Описание: предсказывает с помощью активной модели класс по переданному в него изображению. Если активной модели нет, то метод выдаст исключение.
 - Параметры: file: файл изображения для предсказания.
 - Ответ: Предсказанный класс (PredictionResponse).
 - Код ответа: 200 OK — успешное предсказание. 400 Bad Request — ошибка (например, модель не выбрана).
- /api/v1/models/predict_proba (**Predict Proba**)
 - Метод: POST
 - Описание: Возвращает предсказанный класс с вероятностью, при условии, что в загруженной модели был задан параметр svc__probability = true.
 - Параметры: file: файл изображения для предсказания.
 - Ответ: Предсказание с вероятностью (ProbabilityResponse).
 - Код ответа: 200 OK — успешное предсказание. 400 Bad Request — ошибка.
- /api/v1/models/unload (**Unload**)
 - Метод: POST
 - Описание: Выгружает текущую активную модель из памяти. То есть после выполнения данного метода нет активной модели и выполнить методы predict и predict_proba не получится.
 - Ответ: Успешное сообщение (ApiResponse).
 - Код ответа: 200 OK
- /api/v1/models/remove/{model_id} (**Remove**)
 - Метод: DELETE
 - Описание: удаляет модель по идентификатору из тех, которые до этого создавались методом fit. Если модель имеет тип custom (пользовательская модель) и есть на сервере, то она будет удалена и будет возвращен словарь с ModelInfo, оставшимися на сервере. Если модели нет на сервере или модель типа baseline (baseline-модель всегда хранится на сервере), то будет возвращено исключение.
 - Ответ: Список оставшихся моделей (dict[str, ModelInfo]).
 - Код ответа: 200 OK — успешное удаление. 404 Not Found — модель не найдена.
- /api/v1/models/remove_all (**Remove All**)
 - Метод: DELETE
 - Описание: Удаляет все пользовательские модели (custom) на сервере.
 - Ответ: возвращается словарь с имеющимися baseline моделями на сервере.
 - Код ответа: 200 OK

Приложение на Streamlit представляет собой сервис для обучения, анализа и применения моделей машинного обучения для классификации изображений фруктов и овощей. Оно организовано в виде трёх разделов, представленных в боковом меню:

- **EDA:**

Позволяет загружать датасет (в формате .zip, содержащий изображения и аннотации) на сервер. Отображает основную статистику датасета, включая:

- Средний размер изображений.
- Распределение изображений по классам.
- Распределение дубликатов (если они имеются).

Визуализирует средние значения и стандартные отклонения по цветовым каналам (R, G, B) для каждого класса. Показывает примеры изображений из загруженного датасета. Использует серверные API для загрузки данных, получения метрик и изображений.

- **Обучение модели:**

Содержит два основных блока:

- Работа с уже обученными моделями:

Список доступных моделей, обученных ранее. Отображение параметров модели и её кривой обучения. Возможность удаления одной или всех моделей. - Создание новой модели: Выбор гиперпараметров для алгоритма SVC: Параметр регуляризации C. Тип ядра (например, linear, poly, rbf). Включение оценки вероятности. Построение кривой обучения. Обучение новой модели с заданными параметрами и сохранение её на сервере. Использует серверные API для обучения моделей и управления ими.

- **Инференс:**

Позволяет загрузить изображение (форматы .jpeg, .png, .jpg) для классификации с использованием выбранной модели.

Загружает выбранную модель с сервера и выполняет предсказание. Возвращает результат: - Предсказанный класс изображения. - Вероятность принадлежности к классу (если включена опция probability при обучении модели).

Использует серверные API для загрузки модели и выполнения предсказания.

Инструкция по запуску

- Склонируйте репозиторий

```
git clone https://github.com/AI-YP-24-6/img_classifier.git
```

FastApi сервер отдельно можно запустить по в файле Backend/app/main.py

Streamlit приложение отдельно можно запускается по в файле Frontend/run.py

Для запуска FastApi и Streamlit одновременно выполните команды в 2 консолях:

```
$env:PYTHONPATH="C:<path>\img_classifier"
streamlit run .\Client\app_client.py --server.port=8081 --server.address=127.0.0.1
uvicorn Backend.app.main:app --host=0.0.0.0 --port=54545
```

- Для развертывания двух докер образов выполните команду:

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

Описание docker-compose: В данном сервисе поднимаются 2 объединенных докер контейнера с веб-приложением FastApi и веб-приложением Streamlit.

Этапы проекта

1. Сбор данных
2. Предобработка изображений
3. Подготовка к машинному обучению
4. Машинное обучение (ML)
5. Подбор гиперпараметров
6. Глубокое обучение (DL)
7. Реализация микросервиса

- ✓ Чекпоинт 1. Знакомство
- ✓ Чекпоинт 2. Данные и EDA
- ✓ Чекпоинт 3. Линейные модели | Простые DL модели - 1
- ✓ Чекпоинт 4. MVP Далее на данный момент порядок и даты не фиксированы:
- 📅 Нелинейные ML-модели | Простые DL модели - 2
- 📅 Модели глубинного обучения
- 📅 Финал

WorkFlow

В проекте придерживаемся [Github Flow](#)

Будет дополняться по мере развития проекта...

Структура проекта

img_classifier/	
├── .github/	
│ ├── workflows/	# Настройка автоматизации процессов CI/CD
│ │ ├── docker-linter.yml	# Линтинг Docker файлов
│ │ ├── gh-pages.yml	# Развертывание документации на GitHub Pages
│ │ ├── linters.yml	# Линтинг кода Python и других файлов
│ │ └── pre-commit.yml	# Проверки перед коммитом
├── Backend/	# Серверная часть приложения
│ ├── app/	# Основной код сервиса
│ │ ├── api/	# Реализация API
│ │ │ ├── v1/	# Версия 1 API
│ │ │ │ ├── __init__.py	# Инициализация модуля
│ │ │ │ └── api_route.py	# Маршруты API
│ │ │ ├── __init__.py	# Инициализация модуля API
│ │ │ └── models.py	# Модели данных API
│ │ ├── services/	# Логика сервиса
│ │ │ ├── __init__.py	# Инициализация модуля
│ │ │ ├── analysis.py	# Анализ данных
│ │ │ ├── model_loader.py	# Загрузка модели
│ │ │ ├── model_trainer.py	# Обучение модели
│ │ │ ├── pipeline.py	# Обработка данных в конвейере
│ │ │ ├── preprocessing.py	# Предобработка данных
│ │ │ └── preview.py	# Визуализация результатов
│ │ ├── __init__.py	# Инициализация модуля приложения
│ │ └── main.py	# Точка входа в Backend
│ └── data/	# Данные, используемые Backend
│ └── baseline.pkl	# Сохранённая модель
├── Baseline/	# Базовые эксперименты
│ ├── baseline.ipynb	# Jupyter ноутбук с базовой моделью
│ ├── baseline.md	# Документация по baseline
│ ├── baseline_HOG.ipynb	# Эксперимент с HOG
│ ├── baseline_ResNet18.ipynb	# Эксперимент с ResNet18
│ ├── baseline_SIFT.ipynb	# Эксперимент с SIFT
│ └── baseline_Vgg16.ipynb	# Эксперимент с VGG16
├── Client/	# Клиентская часть приложения
│ ├── app_client.py	# Главный клиентский модуль
│ ├── eda_page.py	# Страница анализа данных
│ ├── model_inference.py	# Предсказания модели
│ ├── model_training_page.py	# Обучение модели в клиенте
│ └── run.py	# Запуск клиентского приложения
├── Docs/	# Документация
│ ├── eda/	# Документация по анализу данных
│ │ └── EDA.md	# Основной файл анализа данных объединенного
датасета	
│ ├── EDA_Fruits360.md	# Анализ датасета Fruits360
│ ├── EDA_Vegetables.md	# Анализ Vegetables
│ ├── EDA_tasty_fruit.md	# Анализ tasty_fruit
│ ├── analysis.md	# Общий анализ данных
│ ├── dataset.md	# Подробности о наборе данных
│ └── goskatalog.ipynb	# Ноутбук для работы с Госкаталогом
│ ├── CI_CD.md	# Документация по CI/CD
│ ├── about.md	# О проекте
│ ├── index.md	# Индекс документации
│ └── tools.md	# Описание инструментов
├── EDA/	# Анализ данных
│ ├── Notebooks/	# Ноутбуки для анализа данных
│ │ ├── EDA.ipynb	# Общий анализ данных
│ │ ├── EDA_Vegetables.ipynb	# Анализ Vegetables
│ │ ├── EDA_fruits360.ipynb	# Анализ Fruits360
│ │ ├── EDA_tasty_fruit.ipynb	# Анализ tasty_fruit
│ │ └── dataset_merging.ipynb	# Объединение наборов данных
│ └── EDA.md	# Анализ объединенного датасета

```
└─ EDA_Fruits360.md # Анализ датасета Fruits360
└─ EDA_Vegetables.md # Анализ датасета Vegetables
└─ EDA_tasty_fruit.md # Анализ датасета tasty_fruit
Media/ # Медиа-материалы
└─ dataset_info.gif # Гифка dataset_info
└─ fit.gif # Гифка fit
└─ load_dataset.gif # Гифка load_dataset
└─ predict.gif # Гифка predict
└─ predict_proba.gif # Гифка predict_proba
└─ streamlit.gif # Гифка streamlit
└─ unload_remove.gif # Гифка unload_remove
Notebooks/ # Дополнительные ноутбуки
└─ download_datasets.ipynb # Скачивание данных
└─ goskatalog.ipynb # Анализ goskatalog.ipynb
└─ kaggle.json.example # Пример файла kaggle
└─ parser.ipynb # Парсинг данных
Tools/ # Утилиты
└─ __init__.py # Инициализация модуля
└─ analysis.py # Анализ данных
└─ download.py # Скачивание данных
└─ logger_config.py # Настройка логирования
└─ notebook.py # Работа с ноутбуками
└─ parser.py # Парсинг данных
.dockerignore # Исключения для Docker
.editorconfig # Общие настройки редактора
.env.example # Пример файла окружения
.flake8 # Настройка линтера Flake8
.gitignore # Исключения для Git
.pre-commit-config.yaml # Настройка pre-commit хуков
LICENSE # Лицензия проекта
README.md # Описание проекта
backend.Dockerfile # Dockerfile для Backend
checkpoint.md # Контрольный файл для заметок
client.Dockerfile # Dockerfile для Client
compose.yaml # Docker Compose файл
dataset.md # Описание набора данных
mkdocs.yml # Конфигурация MkDocs для документации
poetry.lock # Зависимости Poetry
pyproject.toml # Настройка Poetry
report.pdf # Итоговый отчёт
```