Классификатор изображений фруктов и овощей

Годовой проект магистратуры Искусственный интеллект ВШЭ, первый курс.

Команда:

- Красюков Александр Сергеевич (https://github.com/MrASK2024) — (tg: @kas_dev)
- Мевший Илья Павлович (https://github.com/milia20) — (tg: @Ilya12)
- Климашевский Илья Алексеевич (https://github.com/Ilya-Klim) — (tg: @ferox_Y)
- Писаренко Сергей Сергеевич (https://github.com/SerejkaP) — (tg: @SerejP)

Куратор:

Тимур Ермешев (https://github.com/ermetim) - (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
(https://drive.google.com/file/d/1EbvcZbzVXSmB2N1SZYNeUfUuXb8wp3-k/view)
В датасете представлены изображения фруктов и овощей.
Датасет имеет **33 класса по 1400 изображений в каждом**
Подробнее про данные... (dataset.md)

Разведочный анализ (EDA) датасета изображений овощей и фруктов

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

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... (/Baseline/baseline.md)

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.

Демонстрация функционала веб-интерфейса Streamlit и FastAPI находится в файле readme.md.

Веб-интерфейс FastAPI

- /api/v1/dataset/load
    - Метод: POST
    - Описание: Загружает датасет в виде ZIP-архива, содержащего изображения, и обрабатывает его.
    - 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
    - Метод: GET
    - Описание: Возвращает информацию о текущем загруженном датасете.
    - Ответ: Информация о датасете (DatasetInfo).
    - Код ответа:
      200 OK — успешный запрос.
      400 Bad Request — датасет не загружен.
- /api/v1/dataset/samples
    - Метод: GET
    - Описание: Возвращает примеры изображений из классов.
    - Ответ: Стрим изображений в формате PNG.
    - Код ответа:
      200 OK — успешный запрос.
      400 Bad Request — датасет не загружен.

- /api/v1/models/fit
    - Метод: POST
    - Описание: Обучает модель с заданными параметрами.
    - 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
    - Метод: GET
    - Описание: Возвращает список всех моделей, доступных на сервере.
    - Ответ: Словарь с информацией о моделях (dict[str, ModelInfo]).
    - Код ответа: 200 OK
- /api/v1/models/info/{model_id}
    - Метод: GET
    - Описание: Возвращает информацию о модели по её id.
    - Ответ: Информация о модели (ModelInfo).
    - Код ответа:
      200 OK — успешный запрос.
      400 Bad Request — модель не найдена.
- /api/v1/models/load
    - Метод: POST
    - Описание: Загружает указанную модель по id.
- /api/v1/models/predict
    - Метод: POST
    - Описание: Возвращает предсказанный класс для изображения.
    - 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
    - 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/load_baseline
    - Метод: POST
    - Описание: Загружает baseline-модель в активное состояние.(В чекпоинте Baseline мы выбрали SVC модель. С ее параметрами можно ознакомиться в baseline.md(/Baseline/baseline.md))
    - Ответ: Информация о baseline-модели (ModelInfo).
    - Код ответа: 200 OK
- /api/v1/models/unload
    - Метод: POST
    - Описание: Выгружает текущую активную модель из памяти.
    - Ответ: Успешное сообщение (ApiResponse).
    - Код ответа: 200 OK
- /api/v1/models/remove/{model_id}
    - Метод: DELETE
    - Описание: Удаляет модель с указанным id.
    - Ответ: Список оставшихся моделей (dict[str, ModelInfo]).
    - Код ответа:
      200 OK — успешное удаление.
      404 Not Found — модель не найдена.
- /api/v1/models/remove_all
    - Метод: DELETE
    - Описание: Удаляет все модели на сервере.
    - Ответ: Успешное сообщение (ApiResponse).
    - Код ответа: 200 OK

Приложение Streamlit

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
```

- Для развертывания сервиса выполните команду:

```
docker-compose up --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 (https://docs.github.com/en/get-started/using-github/github-flow)]

Будет дополняться по мере развития проекта...