НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«РАЗРАБОТКА СЕРВИСА ДЛЯ КЛАССИФИКАЦИИ ИЗОБРАЖЕНИЙ ОВОЩЕЙ»

Выполнил студент

Павлов Сергей Владимирович

Научный руководитель:

Блуменау Марк

Москва, 2023 г.

Оглавление.

Глава 1. Постановка и описание задачи. Ожидаемые результаты.

1.1. Постановка задачи.

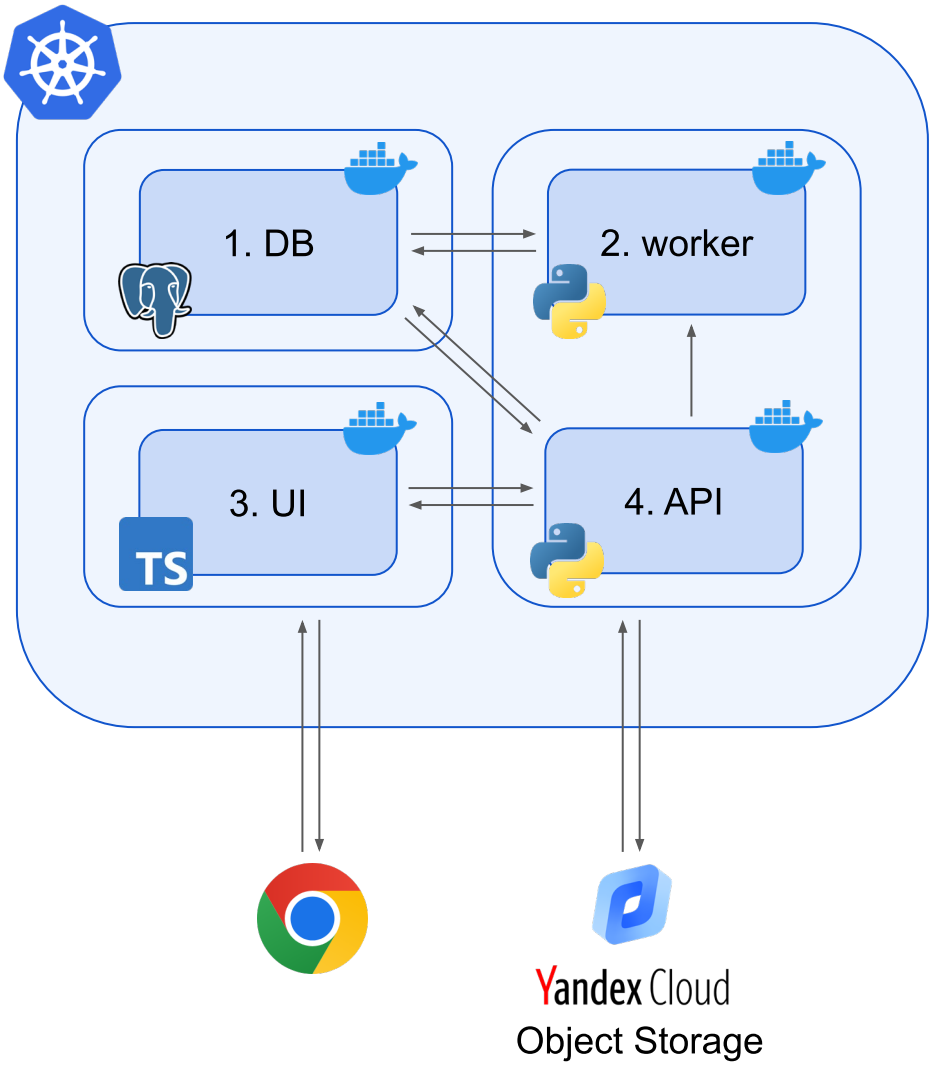
В рамках данной работы планируется использовать открытый набор данных с фотографиями овощей и выполнить следующие подзадачи:

1. Решить задачу классификации овощей по виду овоща (огурец, помидор и т.д.):
   1.1. Для выбора архитектуры воспользоваться примерами успешных решений участников сообщества Kaggle.
   1.2. Адаптировать имплементацию архитектуры выбранной модели под стек, используемый в данной работе (Python, PyTorch), обучить модель, оценить качество.
   1.3. Подготовить сериализованный объект с моделью для инференса.
2. Найти или разработать алгоритм, позволяющий определить доминантный «цвет» изображения (красный, зелёный и т.д.) и разметить при помощи него исходный набор данных. Также включить в процесс инференса разметку новых изображений с использованием этого алгоритма.
3. Разработать API-сервис, позволяющий:
   3.1. Отправлять запрос с комбинацией параметров «количество + вид + цвет» и получать вывод изображений овощей, соответствующих предоставленной комбинации параметров (или визуализацию отсутствия подходящих изображений в базе).
   3.2. Загружать пользовательское изображение овоща и получать вывод его классификации по виду и цвету, предсказанной обученными моделями.
4. Разработать web-интерфейс сервиса для взаимодействия с API через браузер.
5. Выполнить развертывание разработанной связки API + UI на демонстрационном стенде с публичным сетевым адресом и провести живую презентацию работы сервиса.

1.2. Ожидаемые результаты.

В качестве результата классификации по виду овоща ожидается PyTorch модель с качеством по accuracy score не менее 0.97 на данных для валидации. Данная задача уже хорошо отработана коллегами из сообщества Kaggle, поэтому целесообразно воспользоваться лучшими архитектурами решений для получения хорошего качества.

В качестве результата разработки веб-сервиса ожидается приложение со следующей архитектурой:



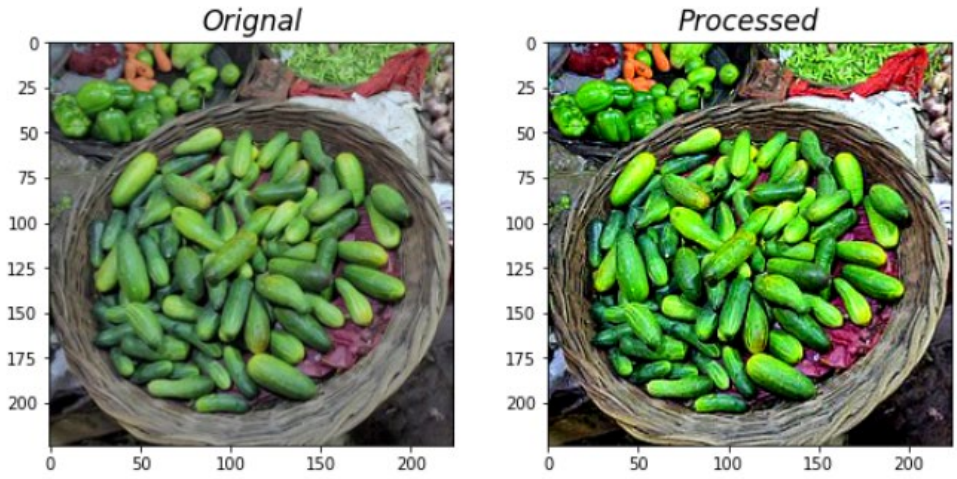
1. На схеме: «DB». База данных PostgreSQL для хранения метаинформации об изображениях из набора данных (имя / путь файла и классификация изображения).
2. На схеме: «worker». Вспомогательный Python-сервис для выполнения задач в фоновом режиме на базе Celery или аналогичного инструмента. В частности, для получения предсказаний от обученных моделей без задержки ответа основного API.
3. На схеме: «UI». Веб-сервер упакованных статических файлов для отрисовки пользовательского интерфейса в браузере. Исполнение кодовой базы UI на TypeScript с использованием одного из актуальных SPA-фрэймворков.
4. На схеме: «API». API-сервер на базе одного из актуальных веб-фрэймворков на Python, обеспечивающий ядро основного функционала.
5. Сборка образов при помощи GitHub CI/CD с последующим хранением артефактов в GitHub Container Registry (ghcr.io).
6. Развертывание в кластере Kubernetes с возможностью доступа к приложению через публичный сетевой адрес.

Глава 2. Краткий обзор существующих решений задачи классификации.

2.1. Пример 1. InceptionV3 transfer learning.

Решение с использованием модели InceptionV3, предобученной на ImageNet. В качестве предобработки исходного набора изображений автор применил следующие преобразования:

- повышение насыщенности цвета
- повышение контрастности
- повышение резкости



Поверх выходного слоя InceptionV3 были добавлены следующие слои:

- слой 2D average pooling
- полносвязный слой с функцией активации ReLU
- слой Dropout с коэффициентом 0.2
- выходной полносвязный слой для классификации с Softmax

Обучение производилось в 5 эпох. В качестве функции потерь была использована категориальная кросс-энтропия. В качестве оптимизационного алгоритма - Adam.

Полученная модель на наборе данных для валидации показала средний accuracy score в размере 0.992. Худшее значение среди всех классов - 0.97. Лучшее значение среди всех классов - 1.00.

2.2. Пример 2. EfficientNet-b0 transfer learning.

Решение с использованием модели EfficientNet-b0, предобученной на ImageNet. В качестве случайных аугментаций для исходного набора данных были использованы следующие преобразования:

- горизонтальное отражение изображения
- изменение высоты изображения с коэффициентом 0.2
- изменение ширины изображения с коэффициентом 0.2
- поворот изображения с коэффициентом 0.2 и заполнением пустоты ближайшим пикселем
- приближение / отдаление изображения с коэффициентом 0.2

Поверх выходного слоя EfficientNet-b0 были добавлены следующие слои:

- слой 2D average pooling
- выходной полносвязный слой для классификации с Softmax

В качестве функции потерь была использована категориальная кросс-энтропия. В качестве оптимизационного алгоритма - Adam.

Обучение производилось в 2 этапа по 5 эпох.

Сначала были использованы 20% от исходного набора данных для обучения. После этого шага модель на наборе данных для валидации показала средний accuracy score в размере 0.992.

На втором этапе веса из предыдущего обучения использовались в качестве начальных весов модели, а набор данных для обучения был уже полным (100% от исходного набора данных для обучения). Архитектура самой модели относительно предыдущего этапа осталась без изменений. Результатом второго этапа обучения стал средний accuracy score в размере 0.998 на наборе данных для валидации.

2.3. Пример 3. CNN.

Решение с использованием стандартного подхода - построения пользовательской архитектуры сверточной нейронной сети.

Преобразования исходного набора данных и случайные аугментации не использовались. Выбранная архитектура слоев нейронной сети выглядела следующим образом:

Тип слоя	Размерность выхода	Кол-во параметров
Conv2D	(None, 150, 150, 32)	896
MaxPooling2D	(None, 75, 75, 32)	0
Conv2D	(None, 75, 75, 64)	18496
MaxPooling2D	(None, 37, 37, 64)	0
Flatten	(None, 87616)	0
Dense / Linear	(None, 128)	11214976
Dropout	(None, 128)	0
Dense / Linear	(None, 128)	16512
Dense / Linear	(None, 15)	1935

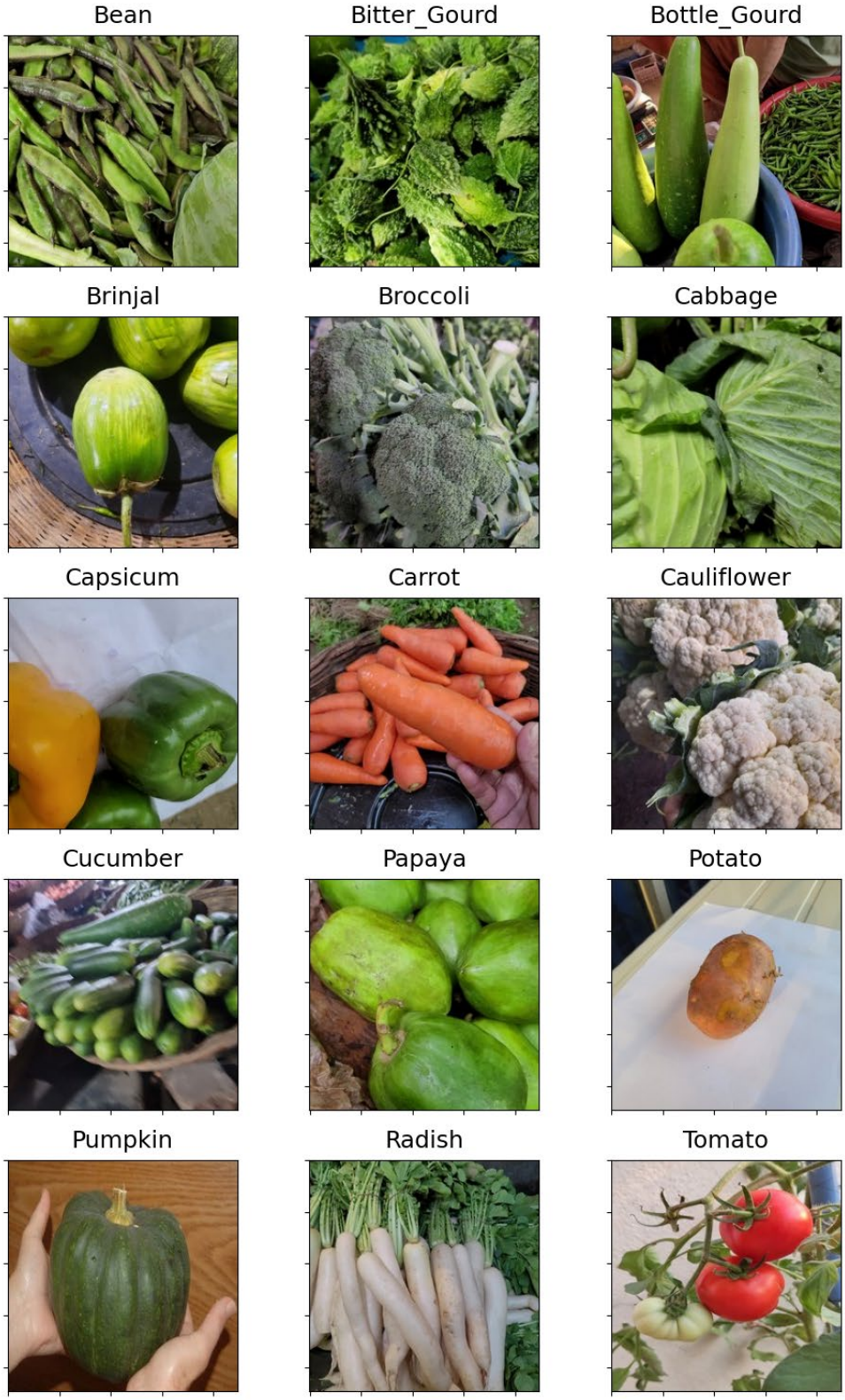
Обучение производилось в 100 эпох с условием остановки при отсутствии улучшения качества модели на протяжении 5 эпох. В качестве функции потерь была использована категориальная кросс-энтропия. В качестве оптимизационного алгоритма - Adam.

Обучение длилось 15 эпох. Полученная модель на наборе данных для тестирования показала средний accuracy score в размере 0.955.

Глава 3. Разведочный анализ данных.

3.1. Примеры классов.

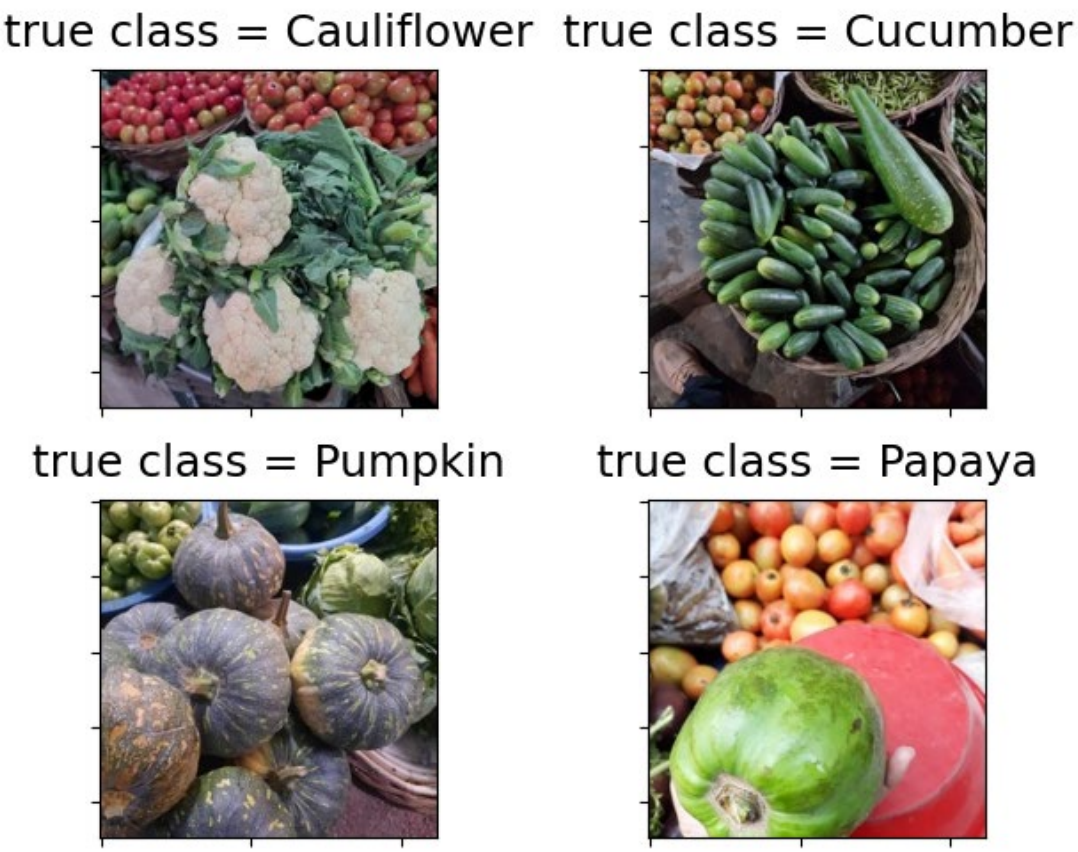
Примеры изображений каждого класса из набора данных для обучения:



3.2. Обзор данных.

Исходный набор данных состоит из изображений 21000 изображений 15 видов овощей. Классы сбалансированы ровно, то есть на каждый вид овоща приходится всего 1400 изображений. Исходный набор данных изначально разделен на группы для обучения, тестирования и валидации (train, test, validation) в отношении 70% / 15% / 15%. Это 15000 / 3000 / 3000 изображений соответственно (1000 / 200 / 200 на каждый класс).

Ручной просмотр изображений не позволил выявить откровенные выбросы, требующие исключения из обучения. Но точно имеют место угловые кейсы, когда несколько разных видов овощей попадаются вместе на одной фотографии, что может привести к ошибкам классификаторов. Примеры:



Глава 4. Описание baseline модели.

4.1. Архитектура модели.

В качестве основания baseline модели использовалась модель INCEPTION_V3 из хаба pytorch/vision версии 0.10.0, предобученная на наборе данных ImageNet. В качестве весов использовался объект torchvision.models.Inception_V3_Weights.DEFAULT.

Последний слой модели (классификатор) был заменён на полносвязный слой с размерностью выхода, соответствующей количеству классов в используемом в данной работе наборе данных – 15.

Для всех изображений из набора данных было применено преобразование увеличения размера картинки до (299, 299), поскольку именно такую размерность модель INCEPTION_V3 ожидает на вход.

В качестве функции потерь была использована категориальная кросс-энтропия. В качестве оптимизационного алгоритма - Adam. Также использовался планировщик StepLR с понижением коэффициента скорости обучения на 0.1 каждые 5 эпох. Размер батча был равен 64.

4.2. Результаты обучения модели.

После обучения в течение 5 эпох модель показала следующие результаты:

Номер эпохи	Accuracy score на тестовых данных	Accuracy score на валидационных данных
1	89.13%	98.60%
2	97.38%	99.07%
3	98.29%	99.30%
4	98.61%	99.50%
5	98.94%	99.60%

Глава 5. Алгоритм определения доминантного цвета.

5.1. Палитры цветов.

Доминантный цвет изображений определяется по двум разным популярным цветовым палитрам отдельно (по каждой палитре свой доминантный цвет):

1. RGB

hex	red	green	blue
#FF0000	255	0	0
#FF8000	255	128	0
#FFFF00	255	255	0
#80FF00	128	255	0
#00FF00	0	255	0
#00FF80	0	255	128
#00FFFF	0	255	255
#0080FF	0	128	255
#0000FF	0	0	255
#8000FF	128	0	255
#FF00FF	255	0	255
#FF0080	255	0	128

2. RYB

hex	red	green	blue
#FE2712	254	39	18
#FC600A	252	96	10
#FB9902	251	153	2
#FCCC1A	252	204	26
#FEFE33	254	254	51
#B2D732	178	215	50
#66B032	102	176	50
#347C98	52	124	152
#0247FE	2	71	254
#4424D6	68	36	214
#8601AF	134	1	175
#C21460	194	20	96

5.2. Описание алгоритма.

Алгоритм определения доминантного цвета основан на вычислении евклидового расстояния в трёхмерном пространстве (reg, green, blue) между пикселями изображений и точками, соответствующими цветам в выбранной палитре.

Для каждого цвета считается среднее по всем пикселям изображения расстояние:

```
import numpy as np

distance = np.mean(
    np.sqrt(
        np.sum(
            np.square(np.subtract(image_array, color_array)),
            axis=2,
        )
    )
)
```

Цвет, для которого его среднее расстояние является наименьшим среди всех цветов палитры, выбирается в качестве доминантного, и его hex код записывается в базу данных.

Глава 6. Описание архитектуры проекта.

6.1. Подготовка данных.

Исходный набор данных скачан с публичного сегмента Kaggle и размещён в приватном хранилище Object Storage на базе сервиса «Яндекс.Облако». Данный тип хранилища имеет интерфейс, совместимый с AWS S3, поэтому для доступа к изображениям из среды приложения используется Python-библиотека boto3.

В рамках приложения реализован API для чтения и записи изображений.

6.2. Обучение модели.

Модель строится на базе Python-библиотеки PyTorch. Обучение модели производится в бесплатной среде Kaggle с использованием GPU. Логика обучения описывается в формате Jupyter Notebook и сохраняется в контроле версий проекта.

6.3. Сохранение обученной модели.

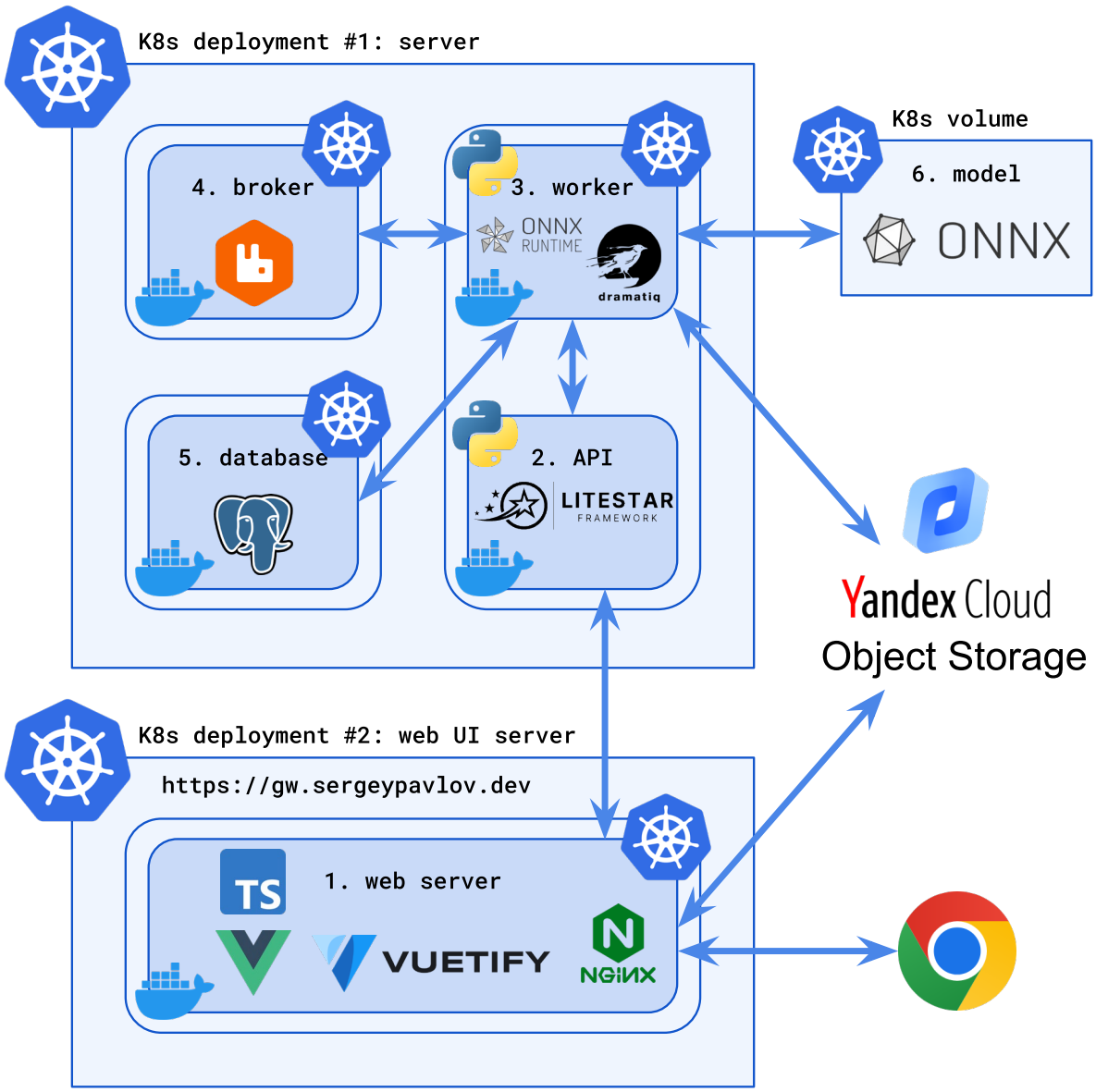
После обучения модели в целях дальнейшего использования для инференеса её объект сериализуется в формат, совместимый с ONNX Runtime, с помощью метода torch.onnx.export. Сериализованный экземпляр модели хранится в том же Object Storage, что используется и для хранения изображений.

6.4. Развертывание модели.

Развертывание выполняется при помощи применения установки Helm chart-ов в кластере Kubernetes на личной машине автора проекта. Серверная часть и веб-интерфейс являются отдельными приложениями и, соответственно, отдельными релизами в контексте Helm.

Автоматизация сборки образов и развёртывания / обновления релизов выполнена на базе GitHub Actions в репозиториях серверного приложения и веб-интерфейса. Образы приложений хранятся в GitHub Container Registry (ghcr.io).

Общая схема архитектуры выглядит следующим образом:



1. На схеме: «web server». Веб-сервер. Для пользователя точкой входа для взаимодействия с приложением является веб-интерфейс, доступный на одном из поддоменов личного сайта автора. Запросы статических файлов браузером для рендера веб-интерфейса обсуживаются веб-сервером nginx.

   Логика веб-интерфейса выполнена на языке TypeScript в формате single page application (SPA) на базе фрэймворка Vue.js и с использованием библиотеки UI-компонентов Vuetify.

2. На схеме: «API». Сервер API. Действия, выполняемые пользователем в веб-интерфейсе, отправляют запросы из браузера пользователя к серверу API

приложения. Логика сервера API выполнена на языке Python с использованием веб-фрэймворка Litestar. Рантайм сервера API обеспечивается процессами WSGI HTTP сервера Gunicorn с использованием класса воркеров UvicornWorker. Это необходимо для совместимости с интерфейсом ASGI, который является расширением WSGI для приложений, использующих асинхронный функционал Python.

3. На схеме: «worker». Процессы для исполнения фоновых задач запускаются как sidecar-контейнер относительно контейнера с сервером API. В рамках рантайма API сервера предусмотрена отправка сигналов на исполнение двух типов фоновых задач:

    1) Загрузка пользовательских изображений в Yandex Cloud Object Storage
    2) разметка доминантных цветов загруженного изображения при помощи алгоритма и предсказание обученной моделью вида овоща на загруженном изображении, а также запись этих результатов в базу данных

    Исполнение фоновых задач осуществляется процессами, вызываемыми с помощью Python-библиотеки Dramatiq. Для инференса внутри процессов используются сессии ONNX Runtime. Сам файл с моделью помещается в Persistent Volume в процессе инициализации основного приложения и считывается процессом фоновой задачи для каждого запуска сессии инференса.

4. На схеме: «broker». Менеджер очередей сообщений RabbitMQ, обрабатывающий сигналы на запуск фоновых задач из рантайма сервера API.
5. На схеме: «database». База данных PostgreSQL. Хранит информацию об изображениях из исходного набора данных и о загруженных пользователем в таблице IMAGE. Есть справочная таблица CATEGORY со списком всех классов овощей, на одну строку которой ссылается каждая запись из таблицы IMAGE. Также таблица IMAGE содержит hex коды доминантных цветов изображения по палитрам RGB и RYB.

    Данные базы хранятся в локальном Persistent Volume сервера. При удаление релиза серверного приложения данные сохраняются и могут быть

использованы при повторной установке. Lifecycle данного PV управляется Helm chart-ом PostgreSQL.

6. На схеме: «model». Persistent Volume для хранения сериализованного экземпляра модели, упакованного в совместимый с ONNX Runtime формат.

Глава 7. Описание плана экспериментов.

7.1. Оптимизация преобразований исходного набора данных.

В рамках baseline модели использовалось только одно преобразование – увеличение размера изображений с 224^2 до 299^2. Даже такой простейший подход показал отличный результат – 99.60% accuracy score на валидационных данных.

Тем не менее, в библиотеке PyTorch имеется объект, доступный по пути torchvision.models.Inception_V3_Weights.IMAGENET1K_V1.transforms(), содержащий набор готовых преобразований для инференса на архитектуре InceptionV3.

Попробовать включить в пайплайн применение этих преобразований и оценить изменения качества предсказаний.

Глава 8. Результаты экспериментов.

8.1. Оптимизация преобразований исходного набора данных.

В пайплайн были добавлены рекомендованные для InceptionV3 преобразования, хранящиеся в torchvision.models.Inception_V3_Weights.IMAGENET1K_V1.transforms().

Список преобразований:

1. Увеличение размерности изображения до (342, 342).
2. Central crop размера (299, 299).
3. Масштабирование числовых значений пикселей на шкалу [0.0, 1.0].
4. Нормализация со средним [0.485, 0.456, 0.406] и стандартным отклонением [0.229, 0.224, 0.225].

С данными преобразованиями обучение модели показало следующий результат:

Номер эпохи	Accuracy score на тестовых данных	Accuracy score на валидационных данных
1	89.57%	98.43%
2	97.57%	99.17%
3	98.17%	99.37%
4	98.56%	99.63%
5	98.71%	99.57%

Можно резюмировать, что на качество модели применение к входным данным преобразований значимо не повлияло.