СОДЕРЖАНИЕ

[Введение 3](#_Toc164140415)

[1. Постановка задачи 8](#_Toc164140416)

[1.1. Введение 8](#_Toc164140417)

[1.2. Цель и задачи выпускной квалификационной работы 8](#_Toc164140418)

[1.3. Предметная область 9](#_Toc164140419)

[1.4. Актуальность 9](#_Toc164140420)

[1.5. Заключение 10](#_Toc164140421)

[2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 12](#_Toc164140422)

[2.1. Введение 12](#_Toc164140423)

[2.2. Описание архитектуры проекта 12](#_Toc164140424)

[2.3. Модель искусственного интеллекта 16](#_Toc164140425)

[2.3.1. Выбор способа обучения 16](#_Toc164140426)

[2.3.2. Выбор архитектуры нейронной сети 20](#_Toc164140427)

[2.3.3. Подбор датасета 21](#_Toc164140428)

[2.3.4. Выбор языка 22](#_Toc164140429)

[2.3.5. Выбор фреймворка 26](#_Toc164140430)

[2.3.6. Выбор средства интеграции 32](#_Toc164140431)

[2.4. Архитектура интеллектуального сервиса 36](#_Toc164140432)

[2.5. Резюме 36](#_Toc164140433)

[3. ПРАКТИЧЕСКАЯ ЧАСТЬ 38](#_Toc164140434)

[3.1. Введение 38](#_Toc164140435)

[3.2. Подготовка 39](#_Toc164140436)

[3.2.1. Анализ датасета и его возможностей 39](#_Toc164140437)

[3.2.2. Анализ и выбор моделей 45](#_Toc164140438)

[3.2.3. Повышение уровня абстракции 47](#_Toc164140439)

[3.2.4. Подготовка инфраструктуры 48](#_Toc164140440)

[3.2.5. Установка зависимостей 49](#_Toc164140441)

[3.2.6. Подготовка данных 50](#_Toc164140442)

[3.2.7. Общая конфигурация для всех моделей 59](#_Toc164140443)

[3.3. Обучение моделей и анализ метрик 63](#_Toc164140444)

[3.3.1. RetinaNet ResNet-50 FPN 63](#_Toc164140445)

[3.3.2. RetinaNet ResNet-101 FPN 67](#_Toc164140446)

[3.3.3. Faster R-CNN ResNet-50 FPN 69](#_Toc164140447)

[3.3.4. Faster R-CNN ResNet-101 FPN 69](#_Toc164140448)

[3.3.5. Faster R-CNN ResNeXt-101 32x8d FPN 69](#_Toc164140449)

[3.3.6. Mask R-CNN ResNet-101 FPN 69](#_Toc164140450)

[3.3.7. Cascade R-CNN ResNet-50 FPN 69](#_Toc164140451)

[3.3.8. Cascade R-CNN ResNeXt-152 32x8d FPN 69](#_Toc164140452)

[3.3.9. Итоговый анализ и выбор лучшей модели 69](#_Toc164140453)

[3.4. Разработка интеллектуального сервиса 69](#_Toc164140454)

[3.5. Интеграция в микробиологию 69](#_Toc164140455)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 70](#_Toc164140456)

Введение

Медицина – одна из важнейших сфер деятельности человека. На сегодняшний день в России существует набирающая обороты тенденция по цифровизации многих медицинских процессов, которые раньше делались вручную и требовали большой бумажной волокиты, а также приводили к ошибкам и неточностям, что в итоге прямым образом сказывалось на качестве лечения и постановке диагнозов.

Наметившиеся тренды по развитию медицины в нашем государстве появились сравнительно недавно, а обрели по-настоящему значительную силу буквально несколько лет назад. Так, в распоряжении правительства РФ от 29 декабря 2021 года № 3980-р утверждено стратегическое направление в области цифровой трансформации здравоохранения. [1]. Основными тезисами принятого распоряжения являются:

* достижение "цифровой зрелости" в здравоохранении
* увеличение доли массовых социально значимых услуг, доступных в электронном виде
* увеличение доли отечественные решения в сфере информационных технологий
* формирование электронных баз знаний
* переход к электронному документообороту

Данному указу предшествовали другие, более общие указы и постановления правительства, направленные на развитие и цифровизации сферы здравоохранения в России. Такие, как:

* указ Президента Российской Федерации от 21 июля 2020 г. N 474 "О национальных целях развития Российской Федерации на период до 2030 года"; [1]
* указ Президента Российской Федерации от 06 июня 2019 г. N 254 "О Стратегии развития здравоохранения в Российской Федерации на период до 2025 года"; [1]
* указ Президента Российской Федерации от 7 мая 2018 г. N 204 "О национальных целях и стратегических задачах развития Российской Федерации на период до 2024 года"; [1]
* постановление Правительства Российской Федерации от 26 декабря 2017 г. N 1640 "Об утверждении государственной программы Российской Федерации "Развитие здравоохранения". [1]

Это доказывает заинтересованность государства в развитии и улучшении качества услуг в области медицины, особенно в последнее десятилетие.

Внедрение информационных систем взамен существующих бумажных и повсеместная цифровизация в медицине даёт неоспоримые преимущества по сравнению с традиционными и двигает отрасль вперёд.

Под цифровизацией в широком смысле понимается процесс социально-экономической трансформации, привносящий массовое внедрение и освоение цифровых технологий. Цифровизация – очень мощный инструмент повышения эффективности и автоматизации процессов в совершенно разных областях деятельности. Она способна качественно и многогранно улучшить многие сферы жизни общества.

Однако недостаточно просто цифровизировать привычный бумажный процесс, переведя его в компьютер. Вызовы времени и инициативы государства ставят требование привнести новые возможности и внедрить технологии, недоступные прежде.

В развитии самых актуальных новшеств в медицине заинтересованы все: пациенты (как платных, так и государственных клиник), получающих более качественные услуги за меньшее время; работники клиник и больниц, результативность деятельности которых возрастает; государство, в интересы которого входит улучшение благосостояния людей и обретение технологического суверенитета.

В медицине для цифровизации процессов внедряются медицинские информационные системы (МИС), а также их вариации, предназначенные для конкретных направлений медицины, например лабораторные информационные системы (ЛИС).

МИС – это комплекс программных и аппаратных средств, предназначенных для организации и управления медицинскими данными и процессами. Они в цифровом виде объединяют электронные карты пациентов, хранят и обрабатывают данные медицинских исследований и мониторинга состояния пациентов, работают с оборудованием, автоматизируют документооборот и так далее.

Первые МИС были достаточно простыми. Они решали только рутинные и трудоемкие операции, такие как:

* регистрация и учет пациентов
* финансовые расчеты
* записи на прием

Более современные системы автоматизируют процесс управления клиникой целиком, берут на себя основную часть нагрузки по ведению записей, карт пациентов, составлению отчётов и взаимодействию с другими подобными информационными системами.

На данный момент разработкой МИС занимаются многие компании как за рубежом, так и в России. Вследствие обширности направлений в сфере здравоохранения таких систем должно быть много. МИС должны учитывать особенности того или иного медицинского процесса, учитывать всевозможные факторы и ситуации, когда что-то идет не так. Они должны реально ускорять работу по сравнению с традиционными методами и помогать людям, которые пользуются ей.

Современным МИС недостаточно просто автоматизировать бумажные процессы. Потеряли актуальность даже те медицинские информационные системы, которые берут под свой контроль все системы и процессы медицинской деятельности. На сегодняшний день актуальны решения, предлагающие инновационные технологии, способные кратно увеличить качество и эффективность деятельности, которую они автоматизируют, такие как:

* нейронные сети и машинное обучение
* облачные вычисления
* большие данные
* телемедицина

Все эти технологии, безусловно, являются передовыми. Отправной точкой для развития было принято решение выбрать нейронные сети и машинное обучение. Поставлена задача разработать интеллектуального компонента для одной из МИС, разрабатываемой на текущий момент по месту работы автора, задачей которого является помощь лаборантам в идентификация микроорганизмов, посеянных и выросших в чашке Петри.

На момент написания автор трудоустроен в организации ООО «НОВОЛАБСИСТЕМ» на должности java-разработчика. В обязанности автора входит написание и сопровождение серверной части медицинской информационной системы для микробиологии.

Чашка Петри – это пластиковая ёмкость, в которую нанесён слой питательной среды. В этой среде могут развиваться разнообразные микроорганизмы, которых туда посеют лаборанты. Нужна она для того, чтобы увеличить количество колоний, взятых из некоторой биологической среды, поскольку на самом образце-источнике биоматериала с микроорганизмами недостаточно для их точной идентификации. После созревания колонии анализируются и изучаются.

Для распознавания колоний в чашке Петри выпускается дорогое медицинское оборудование, в основном импортное, с некоторого момента переставшее ввозиться на территорию нашей страны. Для приобретения и работы с ним требуются внушительные средства, составляющие не только стоимость аппарата, но и труд специалиста, занимающегося обслуживанием и поддержкой этого устройства. Далеко не все лаборатории, особенно в провинциальных городах, могут себе это позволить.

В качестве альтернативы «аппаратному» подходу предлагается программный механизм, идентифицирующей микроорганизмы по одной лишь фотографии, не требующий ни больших вложений, ни сил на поддержку и обслуживание.

Актуальность данного подхода подтверждается обязательным пунктом в техническом задании от министерства цифрового развития для разрабатываемой системы МИС по месту работу автора. Техническое задание звучит так: «Разработка микробиологической информационной системы для цифровизации деятельности микробиологических лабораторий в условиях импортозамещения, функционирующей на системном программном обеспечении российских производителей и свободном ПО (системном ПО с открытым кодом), с функцией формирования и хранения изображений для последующего распознавания образов колоний микроорганизмов с помощью ИИ» [2].

Таким образом, будет разработан интеллектуальный компонент (интеллектуальная подсистема) для распознавания колоний микроорганизмов в чашке Петри и внедрена в разрабатываемую на данный момент МИС (медицинскую информационную систему) для микробиологии.

# Постановка задачи

* 1. Введение

Целью данной главы является обобщения и описание задач данной квалификационной работы, шагов по достижению цели и доказательство её актуальности. В части 1.2 «Цель и задачи выпускной квалификационной работы» описана основная цель квалификационной работы, результатом которой является работоспособный программный продукт, и задачи, которые ставятся на пути к реализации поставленной цели. В части 1.3 «Предметная область» описана предметная область, то есть сфера знаний, в которой будет работать программный продукт. В части 1.4 «Актуальность» подробно описана проблема, существующая в предметной области, которую призвано решить разрабатываемое в данной квалификационной работе программное обеспечение.

* 1. Цель и задачи выпускной квалификационной работы

Главная цель выпускной квалификационной работы – разработать интеллектуальную подсистему для идентификации микроорганизмов в чашке Петри, и встроить её в медицинскую информационную систему, разрабатываемую на данный момент, для последующего внедрения программного обеспечения в реальные лаборатории.

Для достижения поставленной цели выделены следующий этапы:

1. определить задачи, которые продукт будет способен решить.
2. спроектировать концепцию и архитектуру, в какой форме будет разработан и внедрён продукт.
3. описать каждый из компонентов сформированной архитектуры
4. определить основную и второстепенную проблемы, которые может решить продукт.
5. разработать программный продукт в соответствии с намеченными задачами.
6. протестировать продукт и убедиться в его полной работоспособности.
   1. Предметная область

Предметной областью, или, другими словами, сферой деятельности, для которой разрабатывается программный продукт, является медицина. Конкретное направление – микробиология.

Микробиология, как наука, занимается изучением мельчайших организмов – микроорганизмов (микробов). Она изучает закономерности их жизни и развития, а также изменения, которые они вызывает в организме животных и людей.

Уклон сделан не в научную, а в практическую сторону. Это является важным уточнением, поскольку характер работы с микроорганизмами в целях научных изысканий сильно отличается от работы с ними в клинической лаборатории. Так, например, научное изучение неизведанных видов требует крайней осторожности. Если что-то пойдет не так, это может иметь очень серьезные последствия, вплоть до вспышки новых заболеваний или даже глобальной эпидемии. Также, работа в научных целях ведется с несколькими чашками Петри в день (количество может варьироваться, но в целом чашек немного, и их можно контролировать). Клиническая лаборатория, напротив, может работать с сотнями и тысячами чашками в день, и особую биологическую угрозу микроорганизмы в этих чашках, как правило, не представляют.

Приведённые мною в пример факторы дают понимание, почему процессы сильно разнятся несмотря на то, что наука и деятельность одни и те же.

Таким образом, все процессы предметной области, описанные здесь, применимы к деятельности клинических лабораторий.

* 1. Актуальность

Традиционно, для идентификации микроорганизмов в чашке Петри используются масс-спектрометры. Это довольно большие и дорогие устройства, требующие немалых средств на закупку и обслуживание. Не все медицинские организации, особенно государственные и провинциальные, могут себе их позволить. Взамен крупных и затратных аппаратов предлагается программный подход, который будет работать на обычном простом компьютере, в браузере. Лаборатории, которые приобретут МИС (медицинскую информационную систему), получат в её составе механизм, который избавляет от необходимости закупать реальное оборудование, кратно уменьшает время процесса и, если смотреть глобальней, делает медицину доступней для всех людей. Расплатой за программную альтернативу является разве что несколько меньшая точность распознавания микроорганизмов (зависит от многих факторов, колеблется в больших пределах). При правильном использовании, в благоприятных условиях программный подход даёт не менее высокую точность распознавание, как и реальное оборудование.

Ещё одним фактором актуальности данной темы является техническое задание, обязательным пунктом в котором прописана реализация данной подсистемы. Техническое задание подготовлено министерством цифрового развития России для организации, которая является исполнителем проекта, сотрудников которой является автор данной выпускной квалификационной работы.

Актуальность разработки подобных систем обусловлена так же необходимостью обретения страной технологической независимости. В нынешних условиях нет возможности использовать существующие импортные аналоги. Более того, подобных информационных систем может и вовсе не существовать, поскольку Россия является одним из лидеров в IT-сфере в мире, и по многих областям находится на переднем крае, в авангарде.

Данную подсистему действительно ждут медицинские организации, являющиеся пользователями МИС (или планирующие её к приобретению), компонентом которой она является. В этом автор убедился из личных контактов с сотрудником одной из лабораторий.

* 1. Заключение

В части 1.2 «Цель и задачи выпускной квалификационной работы» были подробно разобраны цели, для которых создаётся данное решение, и задачи, которые будет решать этот продукт. Проблемы, ради которых он разрабатывается, действительно остро стоят в нашей стране, и их решение внесёт значительный эффект в развитие медицинской отрасли, а также в общее состояние дел экономики, и увеличит шансы на обретение технологического суверенитета государством.

В части 1.3 «Предметная область» была описана сфера деятельности, в рамках которой будет существовать программный продукт. Медицина является важнейшим институтом в стране, и её развитие принесёт благо для всех граждан.

В части 1.4 «Актуальность» подробно расписаны причины, по которым продукт является актуальным и необходимым для отрасли именно сейчас.

Подводя итог, можно сделать вывод, что тема разработки интеллектуальной подсистемы для распознавания колоний микроорганизмов в чашке Петри актуальна на сегодняшний день и не потеряет своей актуальности в будущем.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

* 1. Введение

В данной главе будут рассмотрены инструменты, технологии и решения, которые возможно применить для решения задачи данной выпускной квалификационной работы. В процессе рассмотрения инструментов будут выявлены их преимущества и недостатки в контексте решаемой задачи, а также рассмотрены их аналоги. По итогу анализа будет определен стек технологий, на котором в дальнейшем будет реализовано приложение.

Обоснованное использование подходящих технологий, а также их актуальность и доступность важны для успешной реализации продукта.

* 1. Описание архитектуры проекта

Архитектура программного проекта – совокупность решений об организации программной системы. Она включает в себя элементы, из которых состоит система, их коммуникации, типы связей, поведение.

Правильная, хорошо продуманная архитектура программного обеспечения целиком и полностью определяет качество работы продукта, его скорость, простоту поддержки. Поэтому так важно заранее тщательно продумать архитектуру программного решения.

Фактически, система будет состоять из трёх взаимосвязанных компонентов:

1. Интеллектуального сервиса для идентификации колоний микроорганизмов в чашке Петри по фотографиям
2. Файлового сервиса для хранения фотографий чашек
3. Пользовательского интерфейса

Интеллектуальный сервис и будет тем самым ядром интеллектуальной подсистемы. Он будет содержать средства, выполняющие процесс распознавания колоний микроорганизмов в чашке Петри на фотографии, а также предоставлять наружу интерфейс для работы с ними.

Задача файлового сервиса состоит в том, чтобы хранить исходные фотографий чашек, а также отдавать их на идентификацию по запросу от клиентской части.

Клиентская часть будет предоставлять удобный интерфейс для работы с фотографиями колоний микроорганизмов и результатами их идентификации.

Такая архитектура обеспечит масштабируемость и гибкость системы, позволяя легко добавлять новые функциональные возможности и модифицировать существующие. Введение микросервисной архитектуры также обеспечит высокую отказоустойчивость системы, так как каждый компонент может быть независимо масштабирован и обновлен без прерывания работы всей системы. Она позволит лучше распределять нагрузку между компонентами и обеспечит легкость в мониторинге и управлении каждым отдельным сервисом. Такая архитектура соответствует принятой в команде концепции разработки системы.

Поскольку программный продукт представляет из себя «подсистему» в пределах единой микробиологической информационной системы, в которую она будет встроена, часть инфраструктуры для её реализации уже в каком-то виде существует. Задача состоит в том, чтобы интегрировать новую интеллектуальную среду в состоявшуюся систему посредством доработки и/или дополнения её прежних компонентов, а также внедрением новых, если потребуется.

Далее будет рассмотрены три компонента, описанные выше. Будут сформулированы полезные функции и предполагаемый контракт для каждого из них. На основании проведенного анализа будет сделан вывод о необходимых действиях для каждого из компонентов, целью которых будет являться разработка завершенную версии всей интеллектуальной подсистемы.

1. Интеллектуальный сервис

На данный момент в микробиологической системе такого сервиса нет. Его надо будет разработать с нуля. Он должен предоставлять такой контракт, работая с которым можно будет отправлять на анализ фотографии с чашкой Петри, а в ответ получать некоторые полезные данные, являющиеся результатами работы искусственного интеллекта или машинного зрения. Эти данные должно быть легко интерпретировать. Их структура должна объединять два качества – простоту и достаточную гибкость. Результаты также должны быть наглядными настолько, чтобы их мог интерпретировать человек, чья профессия не связано с информационными технологиями.

Из вышесказанного определено достаточным сделать две HTTP Rest конечных точки: обе будут принимать на вход изображение в виде файла с типом контента multipart/form-data.

* Первая конечная точка в ответ будет возвращать исходную фотографию, на которую будут наложены обнаруженные участки колоний с их классами. Таким образом, вся работа по визуализации результатов идентификаций будет выполнять на стороне интеллектуального сервиса, а доступна она будет в виде простого изображения.
* Вторая конечная точка в ответ будет возвращать JSON (англ. JavaScript Object Notation). В его структуре будет определён массив, объекты которого состоят из:
  + Класса (категории) микроорганизма
  + Процента уверенности в её верной идентификации
  + Координат местоположения на фотографии

Такой контракт удовлетворяет обоим условиям: первая конечная точка обеспечивает максимальную простоту и наглядность, а вторая предлагает гибкость интерпретации результатов и широкие возможности для кастомизации. В будущем, опираясь на отзывы заказчиков, возможно, можно будет отказаться от одной из них.

1. Файловый сервис

Сервис для хранения файлов, в частности фотографий, уже существует в микробиологической информационной системе. Он предоставляет широкое функциональное API для работы с медиа контентом. На данный момент все фотографии чашек Петри хранятся в нём. Суть работы интеллектуальной подсистемы с файловым сервисом будет сводиться к следующему:

* Получить фотографию из файлового сервиса, нужную для распознавания в ней колоний микроорганизмов
* Отправить фотографию с результатами распознавания обратно файловому сервису для сохранения

Вторая функция, во-первых, актуальна лишь для первой конечной точки интеллектуального сервиса, возвращающей фотографию, а, во-вторых, опциональна, поскольку на данным этапе неизвестно, требуется ли вообще сохранять результаты идентификаций.

Таким образом, в минимальном виде всё взаимодействие с файловым сервисом сводится лишь к получению из него файла. Такой функциональностью он, естественно, уже обладает. Следственно, доработки файлового сервиса для обеспечения работы всей интеллектуальной системы не нужны.

1. Пользовательский интерфейс

Уже существует интерфейс для просмотра фотографий чашек. Он обладает минимальной функциональностью, и разработан лишь для просмотра изображений. Его доработки будут заключаться в том, чтобы обеспечить выполнение контракта интеллектуального сервиса. Основная цель пользовательского интерфейса – выводить результаты распознавания колоний для их последующего анализа сотрудниками лабораторий. Опционально, он может содержать удобные инструменты для работы с фигурами распознанных сегментов, чтобы лаборанты могли корректировать сформированные компьютером результаты, а также выполнять с ними некоторые бизнес-процессы.

Стоит сказать, что пользовательский интерфейс будет разработан другим человеком, являющимся коллегой автору данной выпускной квалификационной работы. Задача автора касательно клиентской части будет заключаться в том, чтобы договориться о контракте и предоставить его наружу через HTTP Rest интерфейс. В связи с этим детали реализации сервиса пользовательского интерфейса будут опущены.

Таким образом, в результате анализа был сформулирован перечень необходимой инфраструктуры, необходимой для интеграции интеллектуальной подсистемы в существующую микробиологическую информационную систему. Была конкретизирована вся необходимая для этого функциональность, а также разработан контракт, который должен её поддержать.

В следующих частях данной главы будут рассмотрены технологии и решения, наиболее подходящие для реализации модели искусственного интеллекта, а также интеллектуального сервиса, который будет её «хостить».

* 1. Модель искусственного интеллекта
     1. Выбор способа обучения

В данной главе стоит задача выбрать наиболее подходящий метод машинного обучения для распознавания классов колоний микроорганизмов и их локализации. Важность выбора правильного подхода определит судьбу всего проекта, поскольку является центральной задачей, решаемой в контексте данной выпускной квалификационной работы.

Машинное обучение – это область искусственного интеллекта, позволяющая компьютерам принимать автономные решения на основе данных без явного программного задания.

Методы машинного обучения на сегодняшний день принято подразделять на две обширные категории:

* Обучение без учителя
* Обучение с учителем

В первом случае обучения проходит без заранее известных правильных ответов. Система сама находит внутренние взаимосвязи, зависимости, закономерности, существующие между объектами без вмешательства внешнего учителя.

При обучении с учителем, наоборот, заранее известен набор правильных ответов, а задача состоит в том, чтобы восстановить зависимости между множеством примеров и множеством правильных ответов.

На сегодняшний день принято выделять две основные сферы машинного обучения:

* Традиционные методы обработки изображений
* Современные методы глубокого обучения

Традиционные методы основываются на алгоритмах извлечения признаков, таких как края, углы, текстуры и формы. Затем эти признаки используются для классификации объектов на изображениях. Важно отметить, что некоторые традиционные методы, например, метод опорных векторов (SVM) или метод ближайших соседей (kNN), могут требовать обучения на исторических данных с аннотированными изображениями. Однако, существуют и методы, не требующие обучения, такие как сопоставление с шаблоном или обнаружение краев.

Традиционные методы могут быть эффективны для простых задач и сценариев с контролируемыми условиями. Однако, они часто испытывают трудности при работе со сложными сценариями, включающими вариации освещения, окклюзии, тени, эффект беспорядка и частично скрытые объекты.

Современные методы глубокого обучения, с другой стороны, зависят от контролируемого или неконтролируемого обучения. Эти методы используют искусственные нейронные сети с большим количеством слоёв для автоматического извлечения признаков и классификации объектов на изображениях. Для высокого качества они требуют большого количества аннотированных данных для обучения, но способны достигать высокой точности распознавания даже в сложных сценариях.

Таким образом, существенная сложность заключается в том, что для глубокого обучения требуется большое количество обучающих данных, разметить которые, как правило, должен эксперт. Процесс аннотации изображений является сложным, долгим и дорогостоящим. Для качественного обучения нужен набор данных с десятками или даже сотнями тысяч размеченных изображений.

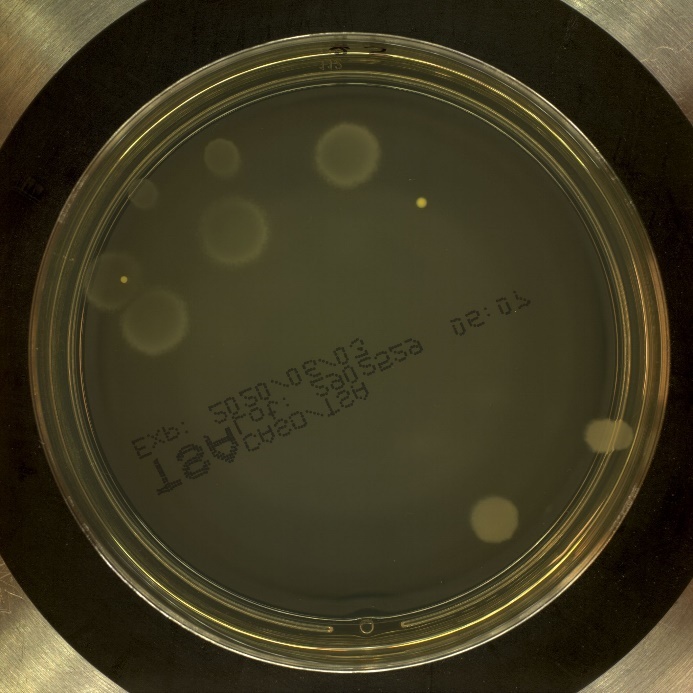
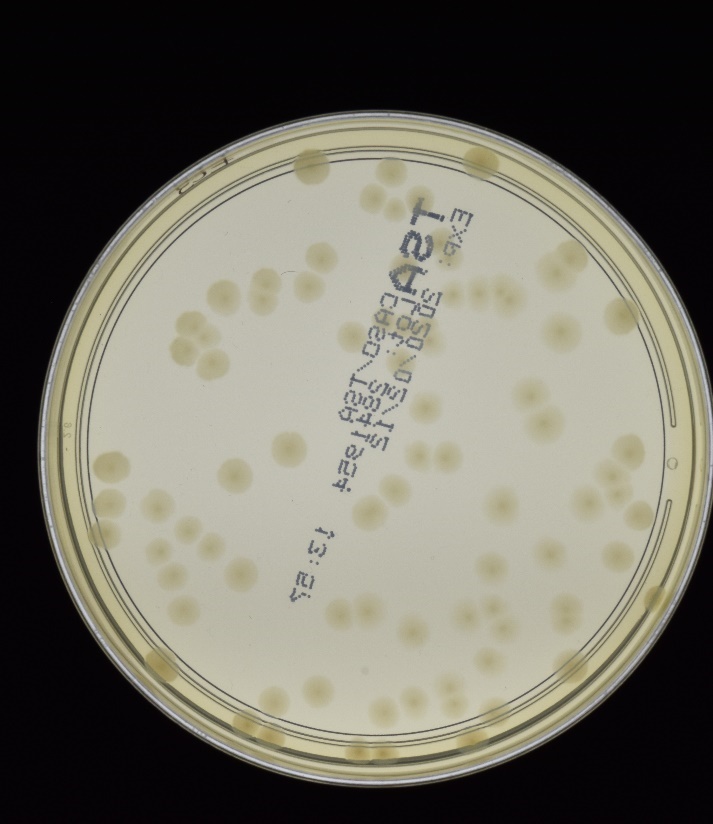
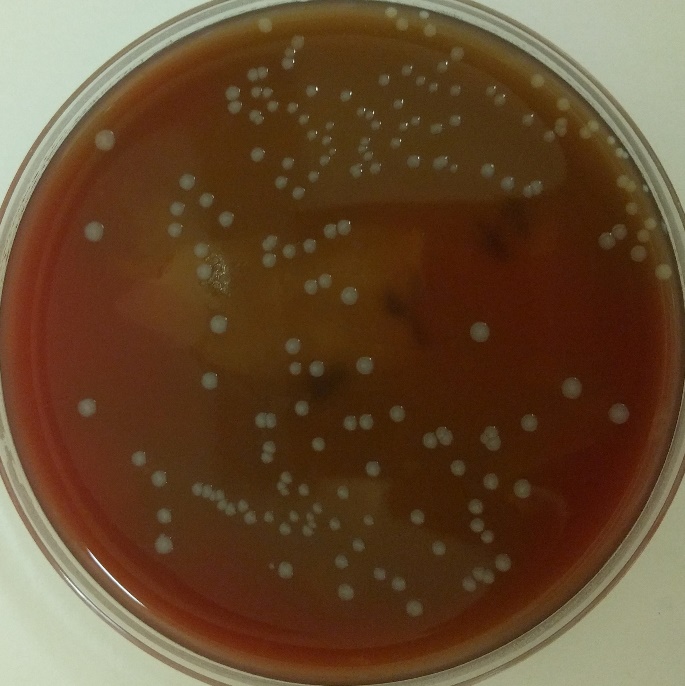
Несмотря на все сложности второго подхода, на данный момент он признан стандартом в области качественного распознавания образов, широко признан исследователями и активно используется крупными компаниями. [7]

Предполагается, что лаборанты будут делать фотографии чашек Петри либо с камер своих смартфонов, либо с веб-камер рабочих компьютеров. В любом случае никогда не будут созданы идеальные условия, в которых, грубо говоря, посреди питательной среды контрастно будут обозначать своё присутствие колонии микроорганизмов. Каждая фотография будет иметь свои уникальные искажение, среди которых:

* Низкий контраст, вызванный несовершенством оборудования
* Вуаль или замыленность из-за грязной оптики
* Пластиковая крышка, закрывающая содержимое чашки
* Непрямой угол сделанной фотографии
* Посторонний мусор
* Брак (неправильный посев биоматериала)

И множество других факторов, из-за которых страдает качество. С такими “боевыми” условиями традиционные методы неглубокого обучения не справятся, или, по крайней мере, не смогут показать достаточную точность.

Примеры фотографий:

Рис. 1, 2, 3, 4. Изображения чашек Петри с созревшими в них микроорганизмами.

В случае выбора методов глубокого обучения для распознавания колоний микроорганизмов в чашке Петри встаёт проблема размеченных данных – нужны тысячи аннотированных специалистами изображений, на которых чётко локализованы и классифицированы колонии.

Таким образом, взвесив все за и против, вследствие сложности распознавания колоний микроорганизмов на фотографиях неидеального качества, вызванного практическими условиями работы, было принято решение использовать современные методы распознаваний образов на изображениях с использованием глубокого обучения. Обучение будет производиться с учителем, так как заранее нужно чётко понимать, где локализованы и как классифицированы колонии микроорганизмов в чашке Петри.

* + 1. Выбор архитектуры нейронной сети

На сегодняшний день есть несколько архитектур глубоких нейронных сетей, предназначенных для классификации изображений. Далее будет приведён краткий анализ наиболее популярных из них. Результат анализа позволит выбрать подходящую архитектуру нейронной сети для эффективного решения задачи в контексте данной выпускной квалификационной работы.

1. Свёрточные нейронные сети (CNN): CNN являются наиболее распространенным типом DNN для классификации изображений. Их архитектура, вдохновленная зрительной корой животных, использует свёрточные слои для извлечения локальных признаков и слои подвыборки для уменьшения размерности данных. CNN обладают высокой способностью к обучению и могут эффективно извлекать иерархические признаки из изображений, начиная от простых краев и текстур до сложных объектов и сцен.
2. Рекуррентные нейронные сети (RNN): RNN специализируются на обработке последовательных данных и могут быть использованы для классификации изображений, представленных в виде последовательности патчей или областей. Они способны улавливать временные зависимости между элементами последовательности, что может быть полезно для анализа изображений с динамическим содержанием, таких как видео.
3. Глубокие сети доверия (DBN): DBN состоят из нескольких слоев ограниченных машин Больцмана (RBM), которые обучаются послойно с использованием жадного алгоритма. DBN могут эффективно обучаться на немаркированных данных и используются для предварительного обучения CNN, что может улучшить их производительность.

Свёрточные нейронные сети (CNN) являются предпочтительным выбором по следующим факторам:

* Эффективное извлечение пространственных признаков. CNN превосходно справляются с извлечением локальных и глобальных пространственных признаков из изображений. Это критически важно для идентификации колоний, так как их форма, размер, текстура и расположение относительно друг друга несут важную информацию для классификации
* Инвариантность к трансляции и повороту. Колонии могут находиться в разных местах чашки Петри и иметь разную ориентацию. CNN обладают свойством инвариантности к трансляции и повороту, что означает, что они могут распознавать колонии независимо от их местоположения и угла поворота изображения.
* Иерархическое извлечение признаков: CNN способны извлекать иерархические признаки, начиная от простых (края, текстуры) до сложных (формы колоний). Это позволяет моделировать сложные визуальные паттерны, характерные для разных видов микроорганизмов.

Таким образом, сверточные нейронные сети (CNN) представляют собой наилучший выбор для задачи распознавания колоний микроорганизмов в чашке Петри. Их способность эффективно извлекать пространственные признаки, игнорируя при этом различия в положении и ориентации колоний, а также возможность моделирования сложных визуальных паттернов, делают CNN идеальным инструментом для анализа микробиологических изображений.

* + 1. Подбор датасета

Подобрать датасет – значит найти такой набор данных, который подойдёт своими характеристиками под поставленную задачу. Он должен:

* Быть достаточно большим. Для выявления и обобщения целевых признаков требуется много обучающих примеров.
* Содержать фотографии, аналогичные реальным. То есть обучающая выборка должна быть схожа с рабочей.
* Быть достаточно разнообразным. Фотографии должны быть сделаны в разных условиях, под разным углом, с разным фоном, освещением, посторонними деталями и так далее. Разнообразить выборку может помочь аугментация.
* Содержать как минимум основные классы рабочих объектов. Классы микроорганизмов в обучающей выборке должны соответствовать обыкновенным штаммам, с которыми лаборанты чаще всего работают. Если выборка содержит только музейные штаммы, то обучение приведёт к результату, при котором обычные колонии микроорганизмов выявлены не будут, что сделает механизм их распознавания бесполезным.

В свободном доступе не так уж и много подобных данных. Поскольку задача – узкоспециализированная и трудоёмкая (размечать объекты должны только специалисты), найти подходящий датасет было крайне тяжело.

К счастью, удалось найти датасет, соответствующий всем необходимым для задачи критериям. Датасет был собран Вроцлавским научно-техническим университетом в 2021 году. Он содержит 18 тысяч изображений чашек Петри, на которых размечено в общей сложности чуть более 336 тысяч колоний микроорганизмов. Он распространяется бесплатно под свободной лицензией Creative Common Attribution-NonCommercial 2.0 Generic license для научных изысканий. [8]

Подробности его состава, особенностей, анализа и эффективного применения будут рассмотрены в 3 части данной выпускной квалификационной работы.

* + 1. Выбор языка

Разрабатывать и обучать нейронные сети можно на разных языках. Каждый содержит свои библиотеки и фреймворки для работы с машинным обучением. При выборе языка программирования для разработки нейронных сетей и их обучения, необходимо учитывать несколько ключевых факторов, таких как поддержка библиотек машинного обучения, удобство использования, производительность, сообщество разработчиков и доступность ресурсов для обучения.

Стоит выбор из трёх вариантов – три языка, предоставляющие необходимые библиотеки для работы с данными и нейросетями: java, c# и python. Первые два используются в организации, в которой автор выпускной квалификационной работы разрабатывает программный продукт. Использование java или c# является выгодным, так как другие разработчики внутри организации смогли бы поддерживать решение и тоже принимать участие в его разработке. Язык python является крайне популярным в области обучения нейронных сетей и ИИ. Он предоставляет мощнейшие инструменты для этого и имеет огромное сообщество, которое поддерживает эти инструменты. Если вкратце: java и c# выгодно использовать в компании из-за принятых стандартов, а python является самым популярным языком с самыми мощными для обучения ИИ инструментами. Кроме того, автор работы имеет опыт программирования на всех трёх.

Далее будут представлены преимущества и недостатки инструментов, выбранных претендентами для реализации интеллектуального сервиса с искусственным интеллектом.

Python является одним из наиболее популярных языков программирования в области разработки нейронных сетей и машинного обучения. Следующие факторы показывают, почему он является хорошим выбором для разработки и обучения нейронных сетей:

В первую очередь это, конечно же, широкий перечень библиотек и фреймворков, специально разработанных для работы с нейронными сетями. Наиболее известные из них - TensorFlow, Keras и PyTorch, которые предоставляют разработчикам инструменты для создания и обучения сложных нейронных сетей. Благодаря этому, разработчики могут легко создавать и исследовать различные архитектуры нейросетей, а также выявлять и исправлять ошибки в процессе обучения.

Во-вторых, Python предлагает простой и интуитивно понятный синтаксис программирования, что позволяет лучше сосредоточиться на разработке целевого решения, и не обращать внимание на особенности языка.

Кроме того, python является кроссплатформенным языком программирования. Это следствие того, что язык интерпретируемый, то есть для его работы нужен интерпретатор, который не зависит от платформы. Переносимость – важная особенность python, и она соответствует требованиям технического задания. Интеллектуальный сервис на python легко запустится на отечественном дистрибутиве Linux.

Наконец, python поддерживает большое и очень активное комьюнити людей. Это означает, что решение большого спектра задач, даже редких и уникальных, скорее всего, имеет решение. За помощью всегда можно обратиться к его разработчикам. А документация постоянно обновляется и находится в актуальном состоянии.

Таким образом, выбор интерпретируемого языка Python является хорошим по следующим причинам:

* Кроссплатформенность
* Простой и понятный синтаксис
* Множество библиотек для работы с данными и машинного обучения для решения любых задач
* Большое комьюнити разработчиков

Java является одним из наиболее популярных языков программирования, который широко используется для разработки приложений, включая приложения и системы искусственного интеллекта и нейронных сетей. Существует несколько причин, почему стоит выбрать Java для разработки и обучения нейросети.

Java предлагает широкий спектр библиотек и инструментов для работы с нейронными сетями. Библиотеки, такие как DeepLearning4j, Neuroph и Encog, предоставляют удобные средства для создания и обучения нейронных сетей в среде Java. Они обеспечивают простоту в использовании и гибкость при разработке различных типов нейронных сетей, включая сверточные, рекуррентные и глубокие нейронные сети.

Java поддерживает распределенные вычисления через фреймворки, такие как Apache Spark и Hadoop, что позволяет использовать несколько компьютеров для обучения больших нейронных сетей. Также java поддерживает использование графических процессоров (GPU) для ускорения обучения нейронных сетей, что особенно полезно для больших и сложных моделей.

Java предоставляет богатую библиотеку классов, таких как Java Neural Network Framework (JNNF), Deeplearning4j и Weka, которые упрощают разработку и обучение нейронных сетей. Эти библиотеки предлагают различные алгоритмы обучения, такие как обратное распространение, оптимизатор градиентного спуска и т. д.

Виртуальная машина Java (JVM) оптимизирует код во время выполнения, что приводит к высокой производительности и делает Java подходящим выбором для ресурсоемких задач обучения нейронных сетей.

Таким образом, java – быстрый, стабильный, статически типизированный объектно-ориентированный язык программирования с обширными инструментами для работы с нейронными сетями.

Что касается C# – он имеет обширную и активную экосистему с обширными библиотеками, инструментами и ресурсами, ориентированными на разработку и обучение нейронных сетей, такими как:

* ML.NET: кроссплатформенная библиотека машинного обучения, которая упрощает разработку моделей нейронных сетей.
* TensorFlow.NET: высокоуровневый API для создания и обучения моделей TensorFlow на C#.
* Keras.NET: библиотека для высокоуровневого создания и обучения нейронных сетей с использованием Keras.

В целом, язык c# имеет схожие с java инструменты и предоставляет равные возможности для разработки и обучения нейронных сетей.

При выборе языка также надо полагаться на возможность удобной и эффективной интеграции с другими сервисами (клиентская часть, сервера). При выборе инструмента доступны компромиссы – выбрать можно самый популярный язык с самыми мощными для обучения инструментами – python, а затем интегрировать его в сервер, например, на java, или вывести наружу конечные точки, сделав его самостоятельным сервисом.

Итак, после тщательного рассмотрения всех достоинств и недостатков языков программирования, используемых для создания и обучения нейронных сетей, было принято решение в пользу Python. Этот язык программирования является лидером в области нейронных сетей и машинного обучения, поэтому использование другого языка только усложнило бы процесс и ограничило бы доступ к новейшим технологиям и инструментам.

* + 1. Выбор фреймворка

Для разработки и обучения нейронных сетей на python предлагается множество разнообразных фреймворков. Неполный список доступных для глубокого обучения нейронных сетей (начиная с наиболее популярных): TensorFlow, PyTorch, Keras, MXNet, CNTK, Caffe2, Deeplearning4j, Chainer. Каждый из них имеет свои преимущества и недостатки, а также зону своего эффективного применения.

Далее будут рассмотрены данные инструменты для разработки и обучения нейронных сетей. В итоге будет сделан выбор в пользу того или иного фреймворка, подходящего лучше всего для решения задачи в контексте данной выпускной квалификационной работы.

1. TensorFlow

Фреймворк создан Google и написан на python и c++. Один из лучших фреймворков с открытым исходным кодом для численных расчётов. Используется в крупных и продвинутых проектах и используется для создания многослойных нейронных сетей. Применяется для распознавания голоса, изображений и в текстовых приложениях, например в гугл-переводчике. Из преимуществ фреймворка можно отметить:

* Немалое количество руководств и документаций
* Поддержка большим сообществом разработчиков и крупными технологическими гигантами
* Поддержка распределенного обучения
* Поддержка мониторинга процессов обучения моделей и визуализации

Однако есть и существенное ограничение – поддержка лишь одного языка программирования python.

В настоящий момент фреймворк TensorFlow является устаревшим. Он уже несколько лет не развивается, поскольку его команда разработчиков была расформирована.

Таким образом, по состоянию на 2024 год фреймворк TensorFlow неактуален.

1. PyTorch

PyTorch – библиотека на python. Является наследником Torch, которая написана на Lua. Разработана в компании Facebook и используется Facebook, Tesla, Twitter, Nvidia, Salesforce, Оксфордским университетом и многими другими. Используется для быстрого и эффективного обучения моделей и является выбором большого числа исследователей. Из преимуществ PyTorch можно выделить:

* Процесс моделирования прост и прозрачен благодаря архитектуре фреймворка
* Имеет большую экосистему пакетов, которая развивается по инициативе сообщества
* Простота отладки
* Обладает декларативным параллелизмом данных
* Содержит большое количество предобученных моделей, которые легко совмещать и комбинировать
* Поддерживает распределённое обучение

Из недостатков можно выделить меньшее количество примеров и менее многочисленное сообщество.

На данный момент PyTorch является основным инструментом для обучения нейронных сетей, особенно в области обработки изображений. В научном сообществе по состоянию на 2024 год он вне конкуренции.

1. Keras

Keras – минималистичная python библиотека, которая может работать поверх TensorFlow, Theano или CNTK. Была разработана инженером Google Франсуа Шолле для ускорения экспериментов. Поддерживает большое количество слоёв нейронных сетей: свёрточные, рекуррентные, плотные. Keras эффективен в решении задач распознавания изображений, распознавания речи, звука и так далее. Главная особенность библиотеки – скорость разработки, что позволяет быстро перейти от замысла к реализации.

Из важных особенностей можно отметить:

* Очень быстрое и простое прототипирование
* Легковесный инструмент в контексте создания моделей глубокого обучения с большим количеством слоёв
* Имеет простой и интуитивно понятный интерфейс – справится даже новичок
* Просто расширяется – новые модули легко добавляются в виде классов и функций
* Поддерживает обучение на кластере графических процессоров
* Поддерживает тензорные процессоры разных производителей, таких как Nvidia, Google, AMD и другие

Среди недостатков библиотека можно выделить:

* Слишком высокоуровневая, что ограничивает некоторые возможности кастомизации
* Её возможности ограничиваются фреймворками, над которыми она построена: TensorFlow, CNTK, Theano

Таким образом, Keras – минималистичная и простая библиотека, способная работать поверх более мощных фреймворков и предоставляющая меньший контроль над нейронной сетью, чем, например, TensorFlow вследствие своей простоты.

1. MXNet

MXNet – платформа для глубокого обучения нейронных сетей с открытым исходным кодом. Берёт своё начало из научный среды и представляет из себя лёгкую, универсальный и высоко масштабируемую среду для глубокого обучения. Библиотека активно поддерживается крупными технологическими гигантами, такими как Amazon, и делает особый упор на машинное зрение, понимание человеческой речи и языка, порождающие модели, свёрточные и рекуррентные сети. Библиотека очень легковесная, и вследствие этого её достаточно просто масштабировать на нескольких процессорах и машинах, что отлично подходит для обучения на больших наборах данных. Она позволяет определять, обучать и развёртывать сети на самых разных устройствах – от высокопроизводительных серверов в облаках, до мобильных и даже носимых устройств. Важные особенности MXNet:

* Простота и удобство программирования
* Высокая степень переносимости между платформами
* Возможность масштабирования в самых широких диапазонах
* Совместима с большим количеством языков программирования

Среди недостатков MXNet можно отметить невысокую популярность в исследовательской среде и небольшое сообщество разработчиков по сравнению, например, с TensorFlow.

В итоге, MXNet – хороший фреймворк для больших промышленных проектов. Но, поскольку он еще относительно нов – есть вероятность, что не получится получить нужную поддержку от сообщества, когда она понадобится.

1. CNTK

CNTK – иначе называется как Microsoft Cognitive Toolkit – фреймворк для обучения глубоких нейронных сетей с открытым исходным кодом. Его особенность заключается в том, что он представляет нейронные сети как серию вычислительных действий через направленный граф, где конечный узлы – это значения или параметры сети, а другие узлы – операции матрицы в ответ на входные значения. Фреймворк позволяет легко разрабатывать и совмещать разные типы моделей, например, упреждающие глубокие нейронные сети, свёрточные сети (RNN), сети с краткосрочной памятью (LSTM). Обучение происходит по принципу стохастического градиентного спуска, при котором вычисления автоматически распараллеливаются между несколькими вычислительными машинами и серверами.

В фреймворке CNTK можно отметить следующие особенности:

* Высокие скорость и масштабируемость. Глубокое обучение происходит быстрее, чем в других инструментах глубокого обучения
* Обладает самой простой в использовании и выразительной архитектурой среди всех инструментов глубокого обучения
* Качество на уровне коммерческого продукта

Выделить можно лишь один минус – маленькое сообщество разработчиков.

1. Caffe2

Caffe2 – как можно понять из названия – прямой потомок Caffe. Это фреймворк, написанный на c++, имеющий интерфейс на python. Он поддерживает свёрточные нейронные сети и сети прямого распространения. Фреймворк хорош для обучения моделей (без написания лишних строк кода), обработки изображений, улучшения существующих сетей. Создан для мобильных, а также крупномасштабных развёртываний в промышленных средах.

Фреймворк хорош по следующим причинам:

* Фреймворк содержит предварительно обученные модели, которые можно использовать для демонстраций
* Фреймворк характеризуется легкостью, быстротой и хорошей масштабируемостью
* Адаптирован к работе с другими библиотеками и фреймворкам, например PyTorch

Из недостатков можно отметить плохую документацию кода и сложность в компиляции, а также исключительно маленькое сообщество разработчиков.

На данный момент ни одна крупная организация не использует Caffe2 в своих продуктах.

1. Deeplearning4j

Выбрать библиотеку Deeplearning4j стоит при использовании языка java для разработки и обучения нейронной сети. Библиотека написана на java, интегрирована с Hadoop и Apache Spark и предназначена для использования в корпоративных средах на распределённых вычислительных модулях. Представляет из себя новую библиотеку, цель которой – обеспечить быстрое создание моделей для специалистов, не являющихся профессиональными исследователями. Deeplearning4j позволяет импортировать модели нейронных сетей из большинства популярных сред, таких как: TensorFlow, PyTorch, Keras и других, тем самым связывая богатую экосистему python с таковой на java. В качестве прикладного программного интерфейса DL4J на Python используется Keras.

Библиотека Deeplearning4j обладает следующими особенностями:

* Быстрота, гибкость и эффективность
* Базируется на java и создана для java и производных jvm
* Возможность работы с Apache Spark и Apache Hadoop
* Открытые библиотеки и достаточно большое сообщество
* Хорошо задокументирована

В целом, библиотека Deeplearning4j будет полезна в том случае, если разработчик хочет разрабатывать и обучать нейронные сети на языке java.

1. Chainer

Chainer – ещё один python-фреймворк для обучения глубоких нейронных сетей. Поддерживается такими крупными компаниями, как IBM, Intel, Nvidia, Amazon. Как и остальные, может без проблем запускаться на нескольких GPU.

Особенность chainer заключается в том, что существующие нейросети могут быть модифицированы прямо во время исполнения. Главными же недостатками фреймворка являются сложность отладки и крайне маленькое сообщество людей, поддерживающих его. [4][5][6]

После тщательного анализа сильных и слабых сторон всех библиотек, принимая во внимание ранее принятые решения по поводу языка для разработки и обучения нейронной сети, а также сложившийся технологический стек в компании, наиболее подходящим и сбалансированным по совокупности всех факторов решением представляется фреймворк на python PyTorch.

* + 1. Выбор средства интеграции

Существует множество способов сохранять, экспортировать и импортировать модели и веса нейронных сетей. Среди такого большого количества вариантов важно выбрать подходящий способ, удовлетворяющий всем необходимым к интеллектуальной подсистеме требованиям.

Далее будут приведены способы, с помощью которых нейронную сеть на python можно “подружить” с другими языками, на которых будет написан сервер, а также описаны преимущества и недостатки каждого подхода. Результаты исследования позволят сделать выбор в пользу того или иного решения.

* ONNX (Open Neural Network Exchange) – это библиотека с открытым исходным кодом, которая предоставляет стандартный, совместимый формат для моделей машинного обучения. Она позволяет специалистам по обработке данных и разработчикам искусственного интеллекта использовать модели в различных рамках [глубокого обучения](https://saturncloud.io/glossary/deep-learning), инструментах, средах выполнения и компиляторах. Когда модель экспортируется в формат ONNX, она включает в себя как структуру графа вычислений, так и параметры модели. Это позволяет импортировать модель в другой фреймворк, где ее можно использовать для вывода или дальнейшего обучения. [3]

Стандарт ONNX совместим с большим количеством фреймворков, таких как:

* TensorFlow
* PyTorch
* scikit-learn
* Keras
* MXNet
* CNTK
* Core ML
* PaddlePaddle

И другими.

Как работает ONNX:

1. Экспорт модели:
   * Разработчик экспортирует свою модель из фреймворка машинного обучения в формат ONNX.
   * Фреймворк машинного обучения предоставляет инструменты для экспорта модели в ONNX.
2. Преобразование модели:
   * ONNX Runtime преобразует модель ONNX в формат, совместимый с целевым фреймворком или приложением.
   * ONNX Runtime может оптимизировать модель для обеспечения высокой производительности.
3. Загрузка и использование модели:
   * Разработчик загружает преобразованную модель в целевой фреймворк или приложение.
   * Целевой фреймворк или приложение использует модель для выполнения вычислений.

Из очевидных преимуществ отмечается:

* Кроссплатформенность. ONNX совместим с различными платформами и языками программирования, что позволяет легко переносить модели между фреймворками и приложениями.
* Простота использования. ONNX имеет простой и понятный формат, что упрощает его использование и интеграцию.
* Производительность. ONNX может быть оптимизирован для обеспечения высокой производительности.
* Сокращение времени разработки. ONNX позволяет разработчикам сосредоточиться на разработке своих приложений, а не на конвертации моделей.

Однако библиотека обладает своими ограничениями:

* Не все модели могут быть экспортированы в ONNX, так как они могут быть слишком сложны
* При экспорте модели может произойти потеря точности
* Не все фреймворки машинного обучения совместимы с ONNX.
* Fast-api – веб-фреймворк для создания быстрых и минималистичных HTTP API-серверов. Поддерживает сериализацию, валидацию и асинхронность из коробки. Благодаря ему можно минимальными трудозатратами предоставить интерфейс работы с нейронной сетью наружу. В этом случае интеллектуальный сервис будет представлять из себя самостоятельную единицу, запускаемую отдельно и независимо. Кроме того, fast api автоматически генерирует и отображает документацию согласно спецификации OpenAPI. Это означает, что клиенты легко смогут разобраться с предоставляемым интерфейсом интеллектуального сервиса.

Поскольку в команде, где в настоящий момент работает автор, параллельно развивается несколько проектов, использующих разных технологии и разрабатываемые разными командами разработки, наиболее предпочтительным методом представляется второй – создание отдельного и независимого интеллектуального сервиса, позволяющего работать с искусственным интеллектом через REST API интерфейс. Этот выбор имеет следующие преимущества:

* Простота реализации. Коллегам, не владеющим навыками работы с ONNX, и, вероятно, не разбирающимся в архитектуре нейронных сетей, гораздо проще будет работать с REST API, нежели с библиотекой, которая подключается определённым образом прямо в коде
* Выделенные ресурсы. Интеллектуальный сервис будет хоститься на производительной машине. Машина эта может находиться в специальном ЦОДе (центре обработки данных). Это позволит эффективно переиспользовать ресурсы
* Масштабирование. Интеллектуальный сервис можно будет легко масштабировать посредством запуска большего количества экземпляров.

Но у этого решения так же есть и недостатки:

* Разрыв соединения. Интеллектуальный сервис может упасть, сетевая связность может нарушиться – подобные форс-мажорные обстоятельства оставят микробиологическую и другие системы без интеллектуальной составляющей
* Сложность развёртывания. Добавляется ещё один сервис с новым для компании языком, который придётся собирать, развёртывать, хостить и обслуживать.

Взвесив все преимущества и недостатки обоих решений, принято остановиться на втором: Искусственный интеллект будет «упакован» в отдельный независимый сервис, называемый Интеллектуальным, посредством вывода конечных точек наружу с помощью REST API интерфейса. Данное решение обеспечит простоту реализации и гибкость в применении по отношении к разным системам в компании.

* 1. Архитектура интеллектуального сервиса

Интеллектуальный сервис в контексте данной выпускной квалификационной работы означает сервис, который будет «хостить» модель нейронной сети, предоставляя удобный контракт для работы с ней. Опираясь на результаты анализа из предыдущей части данной главы (выбор средства интеграции), было принято решение использовать фреймворк fast-api для создания HTTP Rest конечных точек, что позволит простым образом предоставить интерфейс взаимодействия с моделью через интернет. Fast-api – это фреймворк на языке python, поэтому выбор языка программирования для данного сервиса также не стоит.

Таким образом, выбранные технологии позволят реализовать интеллектуальный сервис максимально быстро и просто.

* 1. Резюме

Для решения задачи обнаружения и классификации колоний микроорганизмов в чашке Петри был принят ряд решений, касающихся метода машинного обучения, подбора датасета, выбора языка программирования, фреймворка и средств интеграции. На основании проведённого анализа решено следующее:

* Для решения задачи распознавания образов была выбрана методология обучения с учителем с использованием глубокой нейронной сети. В связи с этим, возникла необходимость в обширном датасете, содержащем размеченные изображения для эффективного обучения модели.
* Был подобран датасет, содержащий большое количество аннотированных данных, достаточного для качественного обучения нейронной сети.
* Языком программирования, на котором будет производиться обучение, выбран Python – несомненный лидер в области машинного обучения и нейронных сетей.
* Фреймворком, который будет помогать в разработке и обучении нейронной сети, был выбран PyTorch. На сегодняшний день он является самым популярным инструментом в области решаемой задачи, а также обладает самыми широкими для этого инструментами.
* Наконец, средством интеграции для практического использования обученной модели выбран фреймворк FastApi. Он позволит создать сервер на python и работать c обученной модель через протокол HTTP.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

* 1. Введение

В данной главе выпускной квалификационной работы будет представлен и описан процесс практической разработки интеллектуальной подсистемы для распознавания колоний микроорганизмов в чашке Петри.

Повествование начнётся с анализа датасета и его особенностей. Это даст представление о:

* формате хранения данных в нём
* содержимом фотографий
* категориях объектов
* заложенных разработчиками вспомогательной информации для более рациональной подготовки данных

И так далее. Результатом анализа датасета будет понимание, какой в принципе есть потенциал для обучения на нём модели глубокой нейронной сети.

Далее будут выбраны модели нейронных сетей, а также инструмент, который поможет их сконфигурировать и обучить. Причины выбора той или иной модели также будут подробно обоснованы.

После выбора моделей, подходящих для задачи, начнётся этап, непосредственно, их обучения. Ход процесса будет наглядно продемонстрирован. Он будет включать метрики, доступные во время обучения, предполагаемое качество итогового результата и его совпадение с ожиданиями, а также примеры фотографий, выбранные таким образом, чтобы наглядно показать характерные черты и нюансы обученной модели.

Далее, после того как все модели будут обучены, отдельным циклом начнётся сбор метрик по каждой из них. Метрики будут выведены в интерактивную доску TensorBoard, нужную для того, чтобы их удобно проанализировать и интерпретировать. Результатом этой части станут финальные показатели качества обученных моделей в виде таблиц и графиков. Этот этап позволит выделить модель, показавшую лучшие результаты, для последующего внедрения её в интеллектуальный сервис и использование в промышленной среде.

Следующим этапом будет произведена разработка интеллектуального сервиса в соответствии с определённым во второй части контрактом, с внедрением в него обученной модели нейронной сети и её весов.

Предпоследним разделом будет являться внедрение всей разработанной в предыдущих главах инфраструктуры в существующую микробиологическую информационную систему.

Завершит данную квалификационную работу краткая глава с демонстрацией её работы на реальных данных, в практических условиях, в составе микробиологической информационной системы.

Каждый из этапов будет сопровождать код, с помощью которого запускаются те или иные программные действия. Идея внедрения кода посреди процесса даст ощущения большей связи с происходящим и добавит некоторого интерактива. К тому же это отразит реальный научный процесс, который именно в такой последовательности и происходит (разработка ведётся в интерактивном блокноте Jupyter Notebook).

* 1. Подготовка
     1. Анализ датасета и его возможностей

Важность анализа датасета состоит в том, чтобы понять, как с ним работать, и как рационально применить всю полезную в нём информацию для качественного обучения с лучшим результатом.

Итак, датасет содержит 18 тысяч фотографий чашек Петри. Каждая из фотографий имеет различные условия освещения, разную питательную среду, разные оптические искажения и так далее. Изображения достаточно разнообразные.

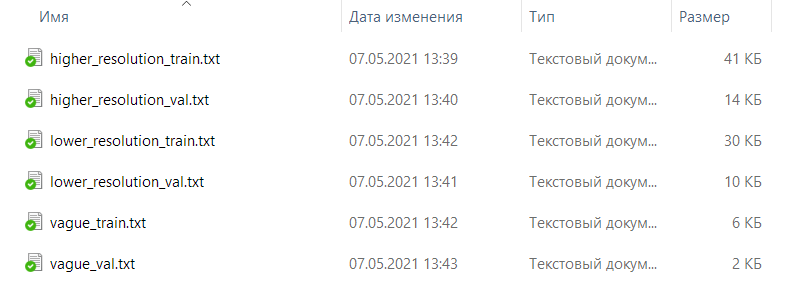
Касаемо вспомогательной информации, которую заложили разработчики датасета для обучения, о которой было упомянуто во введении третьей части, являются тренировочные листы – это обыкновенные файлы с расширением txt, в которых определён массив идентификаторов изображений в json формате. Таких тренировочных листов датасет содержит 6. Вид папки с файлами, содержащей тренировочные листы: 

Рис. 5. Папка с тренировочными листами.

Каждый файл содержит список идентификаторов таких изображений, для каких выборок они предназначены. Таким образом решается сразу несколько проблем:

* Данные изначально поделены рационально. Разработчики датасета в оптимальных пропорциях поделили изображения со схожими характеристиками. Дата-инженеру не надо будет самостоятельно разделять данные по разным выборкам – они сбалансированы изначально. Вследствие этого будет достигнуто более высокое качество обучения за счёт равномерности распределения фотографий с похожими условиями между обучающими выборками.
* В случае, если разработчик нейросети не захочет менять структуру датасета в файловой системе, а выборки он поделит случайным образом, в процессе работы он может столкнуться с тем, что любая внезапная ошибка и завершение сеанса блокнота приведёт к тому, что загруженные по случайно выбранным идентификаторам изображения потеряются. Повторный вызов функции деления выборок вновь перемешает изображения, и станет невозможно впоследствии объективно оценить качество обучения на независимых данных, не участвовавших в обучении, поскольку не понятно, на каких данных модель обучалась, а на каких - нет. С этой проблемой и столкнулся автор данной квалификационной работы поначалу, не имея достаточного анализа производимого процесса.

В общей сложности на фотографиях размечено более 336 тысяч колоний 5 видов и 2 дополнительные категории:

0: "S.aureus" – Золотистый стафилококк (Staphylococcus aureus)

1: "B.subtilis" – Сенная палочка (Bacillus subtilis)

2: "P.aeruginosa" – Синегнойная палочка (Pseudomonas aeruginosa)

3: "E.coli" – Кишечная палочка (Escherichia coli)

4: "C.albicans" – Грибок рода Candida (Candida albicans)

5: "Defect" – Дефект

6: "Contamination" – Загрязнение

Последние две категории решено исключить из обучения по следующим причинам:

* Не стоит задачи распознавать дефекты и загрязнения на чашке Петри.
* Экземпляров этих двух категорий крайне мало относительно экземпляров колоний. Это приводит к большому дисбалансу в данных. Мало того, что дефекты и загрязнения вряд ли будут обнаружены на изображении, так еще и общее качество обучения снизится.

Нумерация категорий начинается с нуля, как и положено делать, если датасет размечен в формате COCO.

COCO (Common Objects in COntext) – это стандартный и наиболее распространённый формат для хранения визуальной информации (фотографий или видео). Он применяется для задач обнаружений объектов (локализации на изображении и определении классов), сегментации экземпляров (разделения на сегменты, соответствующие отдельным объектам), семантической сегментации (классификации каждого пикселя в соответствии с определённым классом) и нахождении ключевых точек (например, суставов на человеческом теле).

Структура формата COCO состоит из следующих частей:

* Набора изображений в папке
* Файла аннотаций в формате json, содержащего:
  + Метаинформацию об изображениях (имя в текущей папке, высота, ширина и т.д.)
  + Сегментации (область, охватывающая объект на изображении, категория объекта, id, площадь и т.д.)
  + Категории (id и название)

В данной выпускной квалификационной работе стоит задача обнаружения объектов, то есть локализация их на изображении с назначением класса. Именно поэтому разработчики датасета разметили данные в формате COCO.

Далее, для большей наглядности будут прикреплены скриншоты директории с фотографиями и содержимым json файла с аннотациями.

Скриншот директории с изображениями:

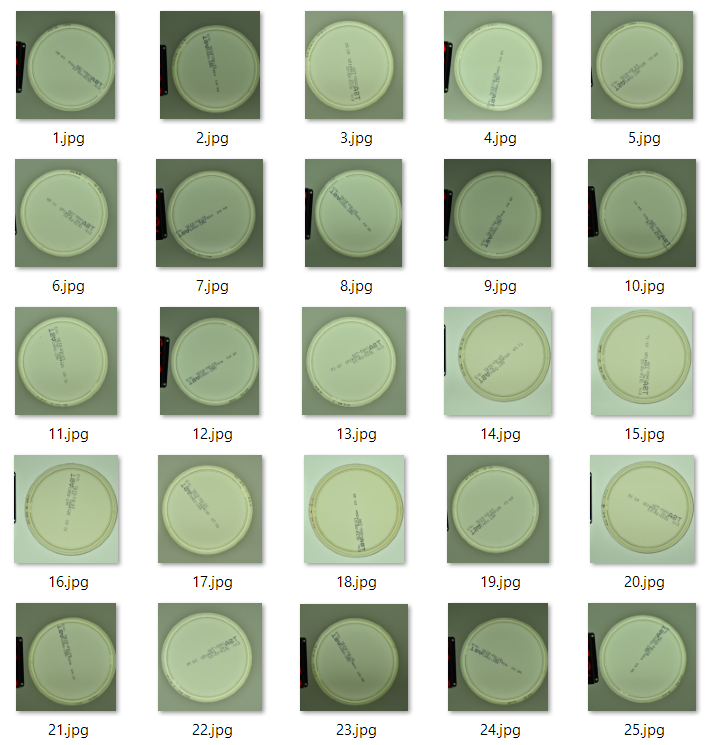


Рис. 7. Директория датасета с фотографиями чашек.

Как можно заметить, именование файлов является порядковым номером. Также, имя файла соответствует его идентификатору в формате {id}.jpg.

Сокращённый вид содержимого файла с аннотациями:

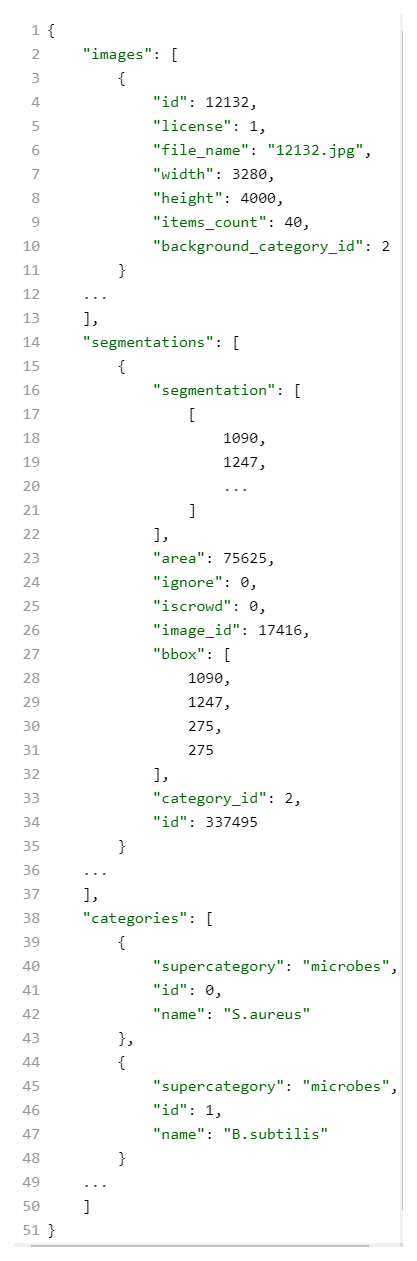


Рис. 6. Сокращённый вид содержимого файла с аннотациями.

Важное уточнение по поводу изображений, который будут участвовать в выборках: на большом количество фотографий в чашках нет созревших микроорганизмов колоний, соответственно, и аннотаций на них тоже нет. Подобные фотографии не несут никакой полезной информации, поэтому они не будут участвовать в обучении. Разработчиками датасета это также было предусмотрено – тренировочные листы содержат идентификаторы только аннотированных изображений.

Также стоит отметить, что все фотографии поделены на 3 раздела: низкого разрешения, высокого разрешения и расплывчатые. То, что датасет содержит фотографии, сделанные в различных условиях, положительно скажется на качестве обучения, поскольку чем большее разнообразие имеется в данных, тем больше модель нейронной сети вынуждена обобщать главные признаки, и не зацикливаться на вторичных.

Впоследствии, метрики будут составлены для трёх этих категорий по отдельности. Это позволит гибче оценить качество обученной модели для фотографий разного качества.

Подводя итоги касаемо анализа датасета: для сбалансированного обучения и достижения более высоких результатов, были приняты следующие решения:

* Исключить из обучения категории «дефект» и «загрязнение»
* Исключить из обучения фотографии без аннотаций
* Тренировочную и проверочную выборки составить на основе оставленных создателями датасета тренировочных листов
* Для каждой из выборок собирать метрики по отдельности
  + 1. Анализ и выбор моделей

В последующих практических частях будут рассмотрены 8 моделей с разной архитектурой. Они взяты из model zoo detectron2 за счёт импортирования конфигурационного yaml файла и скачивания весов. Среди них 2 модели с одностадийной архитектурой, 6 остальных – двухстадийные. Про одностадийные и двухстадийные детекторы будет рассказано далее.

Как правило, детекторы объектов, основанные на глубоком обучении, извлекают особенности из входного изображения или видеокадра. Детектор объектов решает две последующие задачи:

1. Нахождение произвольного количества объектов (возможно, 0)
2. Классификация каждого найденного объекта и определение его размера с помощью ограничивающей рамке (bounding box)

Чтобы упростить процесс, можно разделить эти 2 задачи на 2 стадии. В этом и есть суть работы двухстадийных детекторов. Но такое решение дорого стоит в плане производительности. Архитектура, составленная из двух отдельных стадий, не подходит для быстрого анализа кадров. Эту проблема решают одностадийные детекторы, которые выполняющие обе задачи за один проход. Но это приводит к значительному снижению точности распознавания, а также ухудшает обобщающую способность модели. Далее будет приведено сравнение двух названных архитектур:

Двухстадийные детекторы:

* Включают в себя 2 стадии: определение области объекта с использованием обычных методов компьютерного зрения или глубоких сетей, и классификацию объектов на основе признаков, извлечённых из предагаемой области с помощью регрессии в виде ограничивающей рамки (bounding box)
* Предлагают более высокую степень обнаружения, чем одностадийные, но, как правило, медленнее. Из-за большого количества шагов вывода для каждого изображения производительность (кадров в секунду) не так высока, как у одноступенчатых детекторов
* Включают в себя региональную свёрточную нейронную сеть (RCNN – Region-based Convolutional Neural Network) с разными архитектурами, такими как Faster R-CNN, Mask R-CNN, G-RCNN
* Сначала находят предполагаемую область с объектом, затем используют эту область для классификации.

Одностадийные детекторы:

* В главу угла ставят скорость работы, теряя при этом в точности. Они являются менее хорошими в распознавании объектов непривычных форм или групп маленьких объектов.
* Включают в себя самые популярные архитектуры, такие как: YOLO, SSD, RetinaNet. Самые актуальные из них: YOLOv7, YOLOv8, YOLOv9.
* Основные преимущества обнаружения объектов с помощью одностадийных алгоритмов включают в себя, как правило, более высокую скорость обнаружения и большую конструктивную простоту и эффективность по сравнению с многостадийными детекторами.

Задача данной выпускной квалификационной работы состоит в том, чтобы с максимальной точностью находить и классифицировать колонии микроорганизмов на фотографии. Программное обеспечение не будет использоваться для распознавания на видео/в реальном времени, что не требует от модели нейронной сети быстроты работы, зато ставит самые высокие требования к точности. В связи с этим более рационально видится использование двухстадийных детекторов. Однако для подтверждения теории и научного интереса будут обучены модели обеих архитектур. Результаты, которые будут доступны на интерактивной доске TensorBoard, покажут, подтвердятся ли гипотезы относительно применимости той или иной архитектуры в контексте решаемой задачи, или нет.

* + 1. Повышение уровня абстракции

На сегодняшний день есть немало платформ для обучения нейронных сетей, предоставляющих более высокий уровень абстракции над фреймворком PyTorch. Например, они инкапсулируют многие гиперпараметры, подобранные так, чтобы достичь максимально быстрого и качественного обучения. Также, некоторые из них вычисляются автоматически, в том числе в самом процессе обучения. Подобные фреймворки значительно сокращают объём однотипного кода, который разработчики вынуждены писать раз за разом, работая с фреймворком PyTorch. Они позволяют сконцентрироваться на самом научном процессе, а не особенностях языка или библиотеки. Одной из таких платформ является detectron2. Кроме вышеперечисленного, главная его особенность заключается в том, что он содержит большое количество уже готовых архитектур нейронных сетей, предназначенных для разных задач, в частности, для задачи обнаружения объектов, которая и является главной в контексте данной выпускной квалификационной работы. В контексте detectron2 такой набор заранее подготовленных моделей называется model zoo. Импорт модели происходит так:

1. Импортируется yaml конфигурация модели
2. При необходимости скачиваются веса

Далее модель компилируется в коде. После этого она готова к обучению или предсказаниям. Очень простой механизм.

В следующих частях весь будет код написан с использованием фреймворка detectron2.

* + 1. Подготовка инфраструктуры

Весь процесс обучения начинается с подготовки инфраструктуры для разработки. Разработка будет вестись с использованием следующих инструментов:

* Jupyter Notebook. Это интерактивного блокнот, позволяющий дорабатывать код итеративно, с сохранением всех ранее определенных значений и ссылок. Jupyter Notebook стал де-факто стандартом при работе с искусственным интеллектом.
* TensorBoard. Интерактивная доска для визуального представление процесса обучения в виде графиков, чартов, таблиц и т.д. Этот инструмент позволит наглядно продемонстрировать тенденцию процесса обучения.
* Visual Studio Code. Лёгкая среда для разработки кода. Отлично подходит для данной задачи благодаря интеграции удалённой разработки.

Далее будет кратко описан упрощённый процесс установки и подключения всей необходимой для разработки инфраструктуры. Детали будут опущены, поскольку не являются значительными в контексте данной работы. Итак, процесс происходит так:

1. Jupyter Notebook разворачивается и запускается на сервере с вычислительными ресурсами
2. На том же сервере запускается TensorBoard, который настраивается на просмотр родительской директории над директориями с логами обучения
3. На локальной машине запускается VS code, который удалённо подключается к серверу к порту, на котором запущен Jupyter Notebook.

После данных шагов вся инфраструктура для продуктивной разработки и обучения моделей подготовлена.

* + 1. Установка зависимостей

Для дальнейшей работы необходимо установить и импортировать все зависимости. Стоит сказать, что изначально было неизвестно, какие импортируемые функции и классы понадобятся для работы, поэтому устанавливать они в процессе работы – от начала и до конца. Для упрощения и большой наглядности все зависимости, оказавшие нужными в итоге, выведены в самое начало. Так будет заранее чётко видно, какие функции и классы пригодятся в работе – не больше, не меньше.

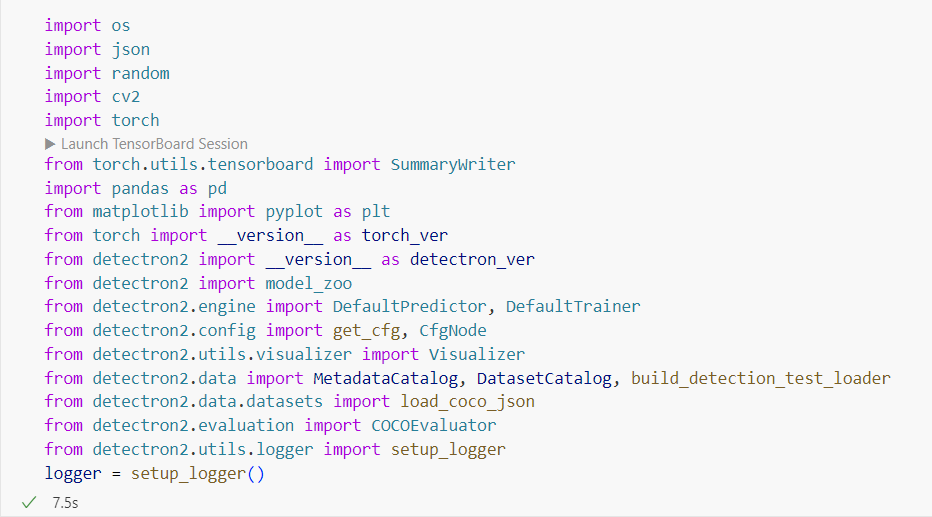
Итак, следующий код демонстрирует все зависимости, которые впоследствии пригодятся для работы: 

Рис. 7. Ячейка кода с установкой необходимых зависимостей.

Далее будет представлен следующий код, задача которого – загрузить выборки, проверить корректность данных и подготовить конфигурацию для последующего обучения моделей.

* + 1. Подготовка данных

В этой части будут приведены и описаны все ячейки с кодом, настраивающие константы, переменные, конфигурации и так далее.

Первым делом выводятся версии PyTorch и cuda, а также доступность cuda ядер (для обучения используется графический ускоритель Nvidia Tesla V100): 

Рис. 8. Ячейка кода с выводом версий библиотек и доступности gpu.

Как видно, PyTorch и detectron2 корректно установлены, имеют актуальные версии (на момент написание), PyTorch видит графический ускоритель и поддерживает работу с ним.

Далее определяются все необходимые в будущем константы: 

Рис. 9. Ячейка кода с определением констант.

Краткое пояснение:

* MODEL\_CONFIG указывает на yaml файл конфигурации данной модели
* MODEL определяет имя модели
* OUTPUT\_DIR назначает выходную директорию, в которую будут складываться все логи, веса и остальные результаты обучения
* DATASET\_NAME задаёт имя датасета
* DATASET\_PATH определяет директорию с фотографиями и json файлом аннотаций
* ANNOTATION\_FILE\_PATH задаёт путь до json файла с аннотациями
* TRAINING\_LISTS\_PATH задаёт путь до директории с тренировочными листами
* CATEGORIES – список категорий микроорганизмов, которые будут участвовать в обучении
* CATEGORIES\_DICT – словарь категорий и их идентификаторов

Далее следует вывод имени модели и выходной директории, чтобы удостовериться, что всё правильно.

Следующим шагом идёт код, который берёт случайное изображение из датасета (в промежутке до количества фотографий) и отображает его: 

Рис. 10. Визуализация случайной фотографии из датасета.

Этот код помогает удостовериться, что все пути заданы корректно.

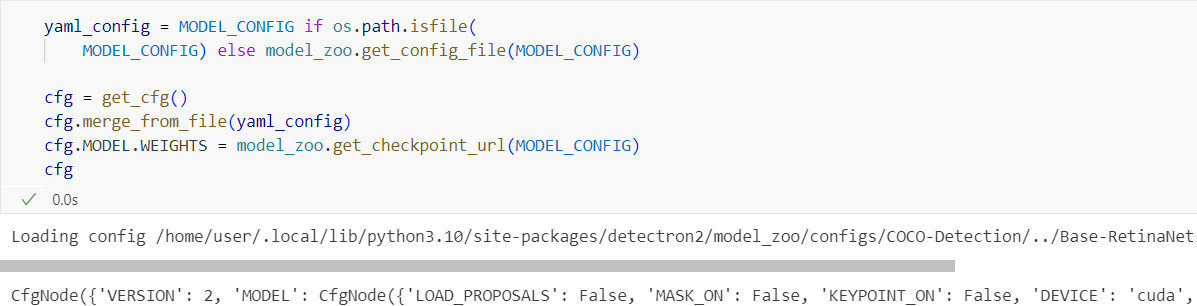
Далее следует код с созданием конфигурации из yaml файла, который либо находится локально, либо будет загружен из model zoo, и загрузкой весов, которые также будут скачаны из репозитория detectron2: 

Рис. 11. Создание конфигурации из yaml файла.

Файл конфигурации содержит стандартные настройки для текущей модели, которые были выставлены для обучения на датасете COCO. Впоследствии, они будут изменены или дополнены в соответствии с нуждами в контексте данной работы.

Вывод даёт понять, что конфигурация загрузилась корректно, а также позволяет проанализировать, какие настройки она содержит.

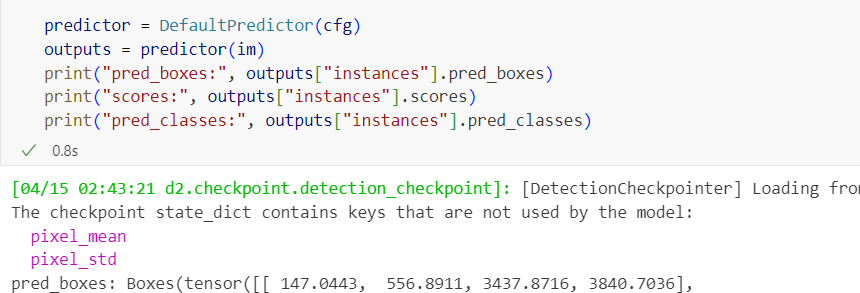
Следующим этапом идёт создание объекта предсказателя (DefaultPredictor) и, непосредственно, предсказание: 

Рис. 12. Создание предсказателя и предсказание объектов на изображении.

Вывод содержит тензоры предсказанных областей (bounding box), оценок (уверенности модели в предсказании) и предсказанных классов. Вывод был ограничен в целях экономии места.

Далее идёт создание визуализатора (Visualizer) и отображение фотографии с результатами предсказания модели: 

Рис. 13. Визуализация предсказания модели.

Удивительно, но модель даже что-то смогла распознать на этом изображении чашки. Наверняка, увидела в ней что-то похожее с некоторым объектом из датасета COCO, на котором была обучена. В целом, это не представляет особенного интереса – главное убедиться, что всё работает.

Следующая ячейка кода содержит определение функций, за счёт которых фотографии из датасета будут приведены к нужному формату, определённому во 2 главе, и загружены в оперативную память: 

Рис. 14. Ячейка кода с определением функций.

Функции носят говорящие названия. Первая функция (get\_ids\_by\_train\_list) принимает на вход путь до файла с тренировочным листом, а на выход отдаёт список идентификаторов фотографий, предназначенных для определённой выборки. Этот набор, как уже упоминалось ранее в части анализа датасета, определён разработчиками для более качественного обучения вследствие рационального распределения фотографий. Вторая функция (filter\_dataset\_by\_ids) предназначена для фильтрации датасета (в формате, который понимает detectron2) по соответствующим идентификаторам из списка, который будет получен из файла с тренировочным листом. Наконец, функция (filter\_annotations\_by\_categories) служит для фильтрации лишь тех аннотаций, которые содержат заданные категории. Таким образом исключаются аннотации, содержащие категории дефекта (defect) и загрязнения (contamination).

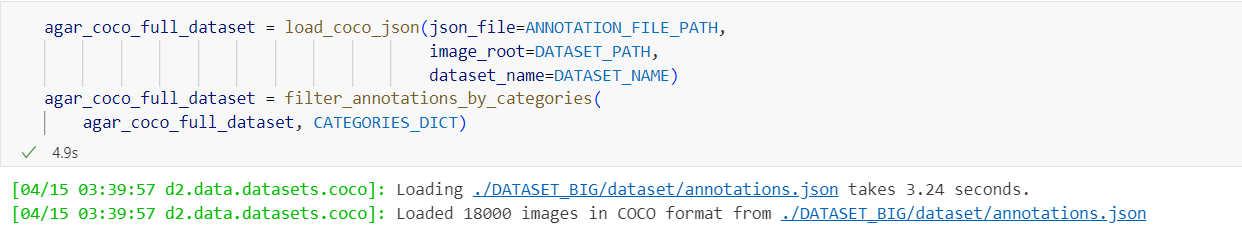
Следующая ячейка содержит код, который сначала загружает датасет в понятный для detectron2 формат (за счёт встроенной функции load\_coco\_json), а затем фильтрует в нём аннотации по заданным категориям: 

Рис. 15. Ячейка кода с загрузкой и фильтрацией датасета.

Вывод сообщает о том, что 18 тысяч фотографий в формате COCO были загружено успешно.

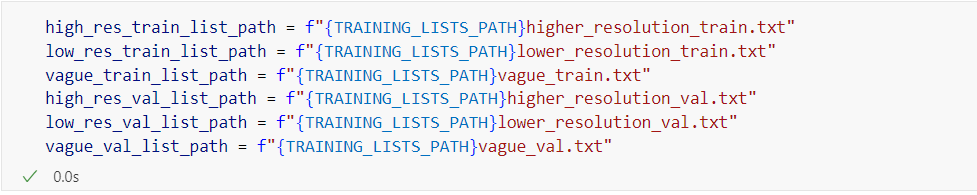
Следующим этапом идёт назначение путей ко всем файлам, которые содержат список идентификаторов для каждой из выборок: 

Рис. 16. Ячейка кода с назначением путей к файлам с тренировочными листами.

Далее следует довольно-таки сложный код, суть которого простая – зарегистрировать все выборки в оперативной памяти в формате detectron2: 

Рис. 17. Ячейка кода с регистрацией выборок.

DatasetCatalog и MetadataCatalog – объекты для хранения датасета и его метаинформации соответственно. Получать потом их можно будет с помощью метода get, передав ему параметром название датасета или метаинформации.

Объект метаинформации содержит:

1. Название датасета
2. Классы объектов
3. Путь до директории с фотографиями
4. Формат датасета

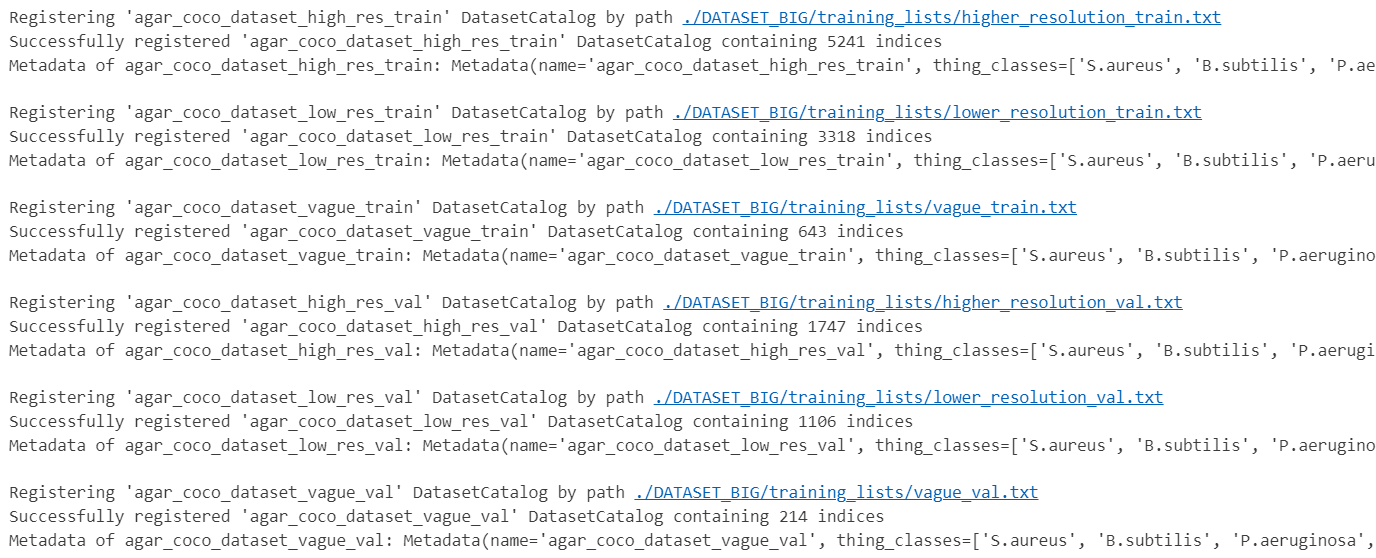
В результате печатается массивный вывод, свидетельствующий о том, что всё было зарегистрировано успешно: 

Рис. 18. Вывод после кода с регистрацией выборок.

Аналогичным образом регистрируются дополнительно 2 датасета, объединяющие все тренировочные и все валидационные выборки: 

Рис. 19. Ячейка кода и вывод с регистрацией полного валидационного датасета.



Рис. 20. Ячейка кода и вывод с регистрацией полного тренировочного датасета.

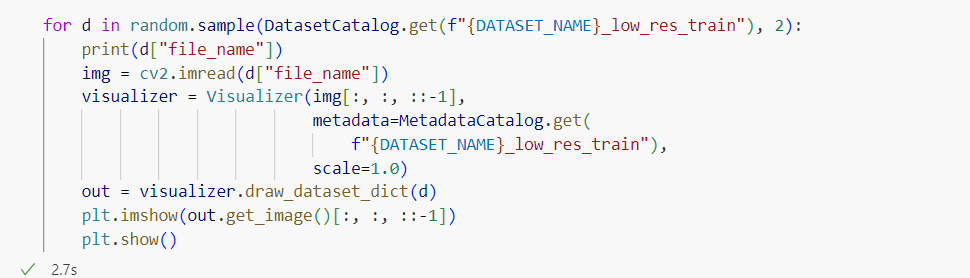
Далее, чтобы вновь убедиться, что только что зарегистрированный датасет в правильном формате, были выведены 2 случайные фотографии с аннотациями: 

Рис. 21. Ячейка кода для вывода размеченных фотографий.

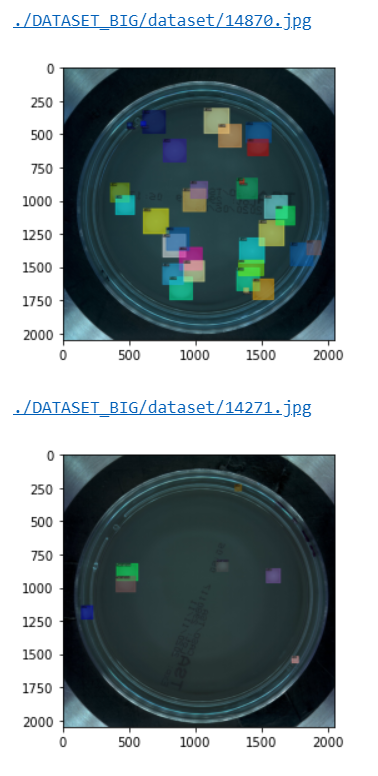
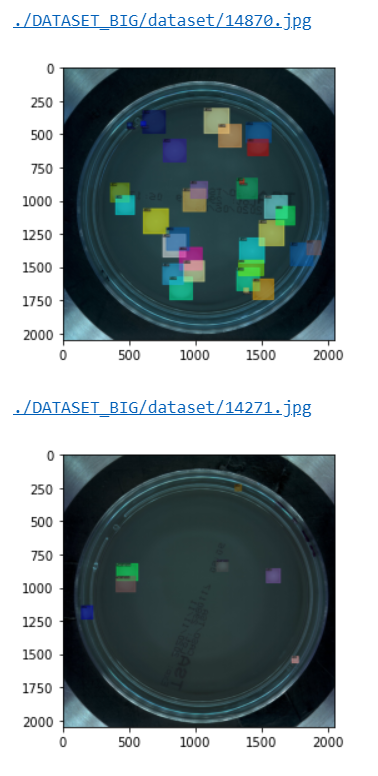
Вывод свидетельствует о том, что всё в порядке:  

Рис. 22. Вывод фотографий с аннотациями.

* + 1. Общая конфигурация для всех моделей

При обучении разных моделей нейронных сетей, для объективного и наглядного сравнения их производительности, снижения влияния посторонних факторов, а также упрощения и воспроизводимости результатов, было принято составить общую конфигурацию (набор гиперпараметров) для всех обучаемых моделей.

В этой части будет показана ячейка кода, которая содержит настройку общей конфигурации для всех моделей. Каждый из гиперпараметров будет достаточно подробно пояснён. Итак, ячейка кода с конфигурацией:

Рис. 23. Общая конфигурация для всех моделей.

Пояснение констант (набор гиперпараметров, общих для всех моделей) конфигурации:

* cfg.DATASETS.TRAIN – список выборок для обучения. Назначаются все созданные ранее в коде тренировочные выборки
* cfg.DATASETS.TEST – список выборок для тестирования. Список обнуляется, поскольку он содержит исходные выборки, на которых обучалась модель
* cfg.MODEL.WEIGHTS – веса модели. Берутся исходные веса предобученной модели
* cfg.OUTPUT\_DIR – выходная директория, в которую будут складываться логи (метрики) во время обучения
* cfg.SOLVER.IMS\_PER\_BATCH – количество изображений за одну итерацию. Для одного графического ускорителя рекомендуется выставлять 2
* cfg.SOLVER.MAX\_ITER – максимальное количество итераций обучения. Именно столько шагов будет длиться обучение, по 2 изображения на каждый шаг. Исходя из эмпирических наблюдений касаемо прогресса обучения, а также банальной интуиции, принято решения установить 50 тысяч итераций
* cfg.SOLVER.STEPS – список контрольных точек, равный количеству итераций, на которых будет происходить снижение скорости обучения (learning rate) на множитель gamma, который равняется 0.1 по умолчанию
* cfg.SOLVER.BASE\_LR – базовая скорость обучения, на которую модель выйдет после первых n шагов
* cfg.MODEL.ROI\_HEADS.NUM\_CLASSES – количество классов для предсказания. Равняется выходному тензору модели.
* cfg.TEST.DETECTIONS\_PER\_IMAGE – максимальное количество детекций на одной фотографии. Датасет не содержит фотографий, на которых размечено после 300 объектов – в этом случае их количество считается неисчислимым, а в понятиях микробиологии – чашка бракованная. Таким образом, 300 выглядит логичным значением.
* cfg.MODEL.DEVICE – устройство, на котором будет происходить обучение. В случае данной работы обучение будет происходить на графическом ускорителе Nvidia Tesla V100.
* cfg.INPUT.MIN\_SIZE\_TRAIN – минимальный размер тренировочных фотографий. Фотографии в датасете есть разного размера, но минимальные из них – 2024х2024. Это довольно-таки большой размер. Стоит его снизить в 2 раза – этого, во-первых, будет более, чем достаточно, а во-вторых, понижение размерности на кратный множитель ускоряет сей процесс
* cfg.INPUT.MAX\_SIZE\_TRAIN – максимальный размер фотографий. Равен минимальной стороне самых маленьких изображений в датасете
* cfg.MODEL.MASK\_ON – флаг, означающий, должна ли модель сегментировать объекты. Во-первых, такой задачи не стоит, во-вторых, датасет такой информации не содержит, поэтому устанавливается на False.

Некоторые гиперпараметры подобраны интуитивно. Данная конфигурация не претендует на самую оптимальную и эффективную, однако умеренный анализ и тестирование показали, что она достаточно работоспособна.

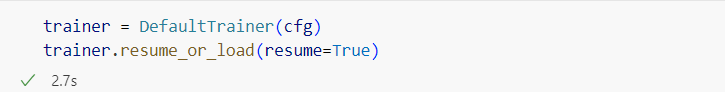
После установки всех гиперпараметров и формирования конфигурации следует создание объекта тренера (DefaultTrainer). Также устанавливается флаг resume=True, чтобы была возможность останавливать обучение в любой момент и продолжать его с последней сохранённой точки: 

Рис. 24. Ячейка с кодом создания тренера и установкой флага.

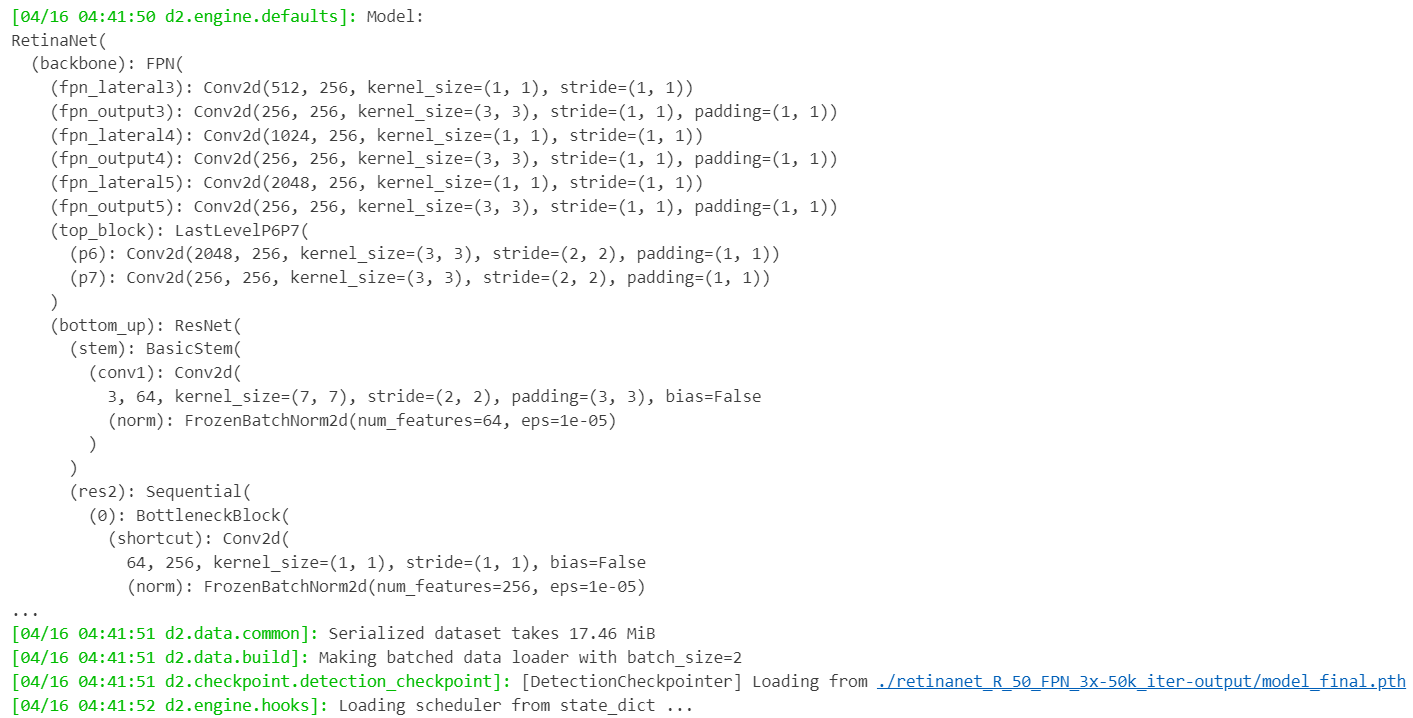
Вывод: 

Рис. 25. Вывод архитектуры модели.

Вывод показывает полную архитектуру модели в формате PyTorch. В целях экономии пространства он был сокращён.

На данном этапе вся подготовительная работа завершена. Далее следует уже непосредственно обучение.

Оно будет начато посредством запуска следующей ячейки: 

Рис. 26. Ячейка с кодом запуска обучения модели.

* 1. Обучение моделей и анализ метрик

Повествование в следующих частях, в которых будут обучены конкретные модели, будет построено следующим образом:

1. Сначала будет описана архитектура модели и её составляющих
2. Далее будет приведено обоснование, почему в контексте данной задачи имеет смысл опробовать именно эту модель
3. Следующим шагом начнётся, непосредственно, обучение модели
4. После обучения будут приведены метрики, отражающие процесс, как модель обучалась, и каких результатов в итоге достигла
5. Сводки и графики будут подкреплены тестовыми фотографиями с предположениями модели. Эти фотографии будут подобраны таким образом, чтобы наглядно отразить особенности работы именно этой обученной модели
6. Наконец, будет подведён краткий итог в виде комментария, как прошло обучение и чего в конце концов смогла достигнуть модель.
   * 1. RetinaNet ResNet-50 FPN

Первой моделью для обучения будет рассмотрена одностадийная модель RetinaNet, основанная на свёрточной сети ResNet-50 (Residual Network), которая имеет 50 слоёв. Модель основана на архитектуре FPN (Feature Pyramid Network). Основная информация по модели и основным её компонентам:

* ResNet (Residual Network) – это глубокая свёрточная нейронная сеть, в основе которой лежит концепция остаточных соединений, которая позволяет обучать очень глубокие сети, избегая проблемы исчезающего градиента (в глубоких нейронных сетях градиенты могут становиться очень малыми при обратном распространении через множество слоев. Это затрудняет обновление весов в ранних слоях сети и приводит к снижению производительности). Остаточный блок состоит из нескольких свёрточных слоев с функцией пропуска (shortcut connection). Функция пропуска добавляет вход блока к его выходу. Это позволяет градиентам легко распространяться через сеть, так как они могут "пропускать" слои через функцию пропуска.
* FPN (Feature Pyramid Network) – является архитектурным компонентом, который создает многомасштабную пирамиду признаков из одного изображения. Традиционные методы обнаружения объектов, основанные на свёрточных нейронных сетях (CNN), используют признаки с последнего слоя сети для предсказания. Однако, признаки на этом уровне имеют низкое разрешение и не содержат достаточно информации для обнаружения мелких объектов. Для решения этой проблемы, FPN использует иерархическую структуру CNN для построения пирамиды признаков, которая содержит информацию как о мелких, так и о крупных объектах. Говоря другими словами, FPN призвана объединить достоинства карт признаков нижних и верхних уровней (первые имеют высокое разрешение, но низкую семантическую, обобщающую способность; вторые — наоборот).

Ключевой инновацией сети ResNet является функция потерь Focal Loss. Она решает проблему дисбаланса классов, когда число фоновых объектов значительно превышает число целевых.

Модель имеет смысл обучить в контексте решаемой задачи потому, что она является типичным представителем одностадийных детекторов. Дальнейшее сравнение в виде графиков с метриками и фотографий с предположениями модели наглядно покажет разницу этих двух (одностадийных и двухстадийных) архитектур.

Итак, обучение окончено. Заняло оно чуть более 5 часов на 50 тысячах итераций. На момент обучения доступны метрики: loss\_cls, loss\_box\_reg, lr, num\_pos\_anchors, total\_loss. Расшифровка и интерпретация:

* loss\_cls: Ошибка классификации. Эта метрика отражает, насколько хорошо модель классифицирует anchor boxes как содержащие объекты определенного класса или фон. Низкое значение loss\_cls указывает на то, что модель хорошо различает объекты и фон.
* loss\_box\_reg: Ошибка регрессии ограничивающих рамок (bounding boxes). Эта метрика показывает, насколько точно модель предсказывает координаты ограничивающих рамок объектов. Низкое значение loss\_box\_reg означает, что модель точно локализует объекты.
* total\_loss: Общая функция потерь. Обычно это сумма loss\_cls и loss\_box\_reg, возможно с дополнительными весами. Снижение total\_loss указывает на общее улучшение производительности модели.

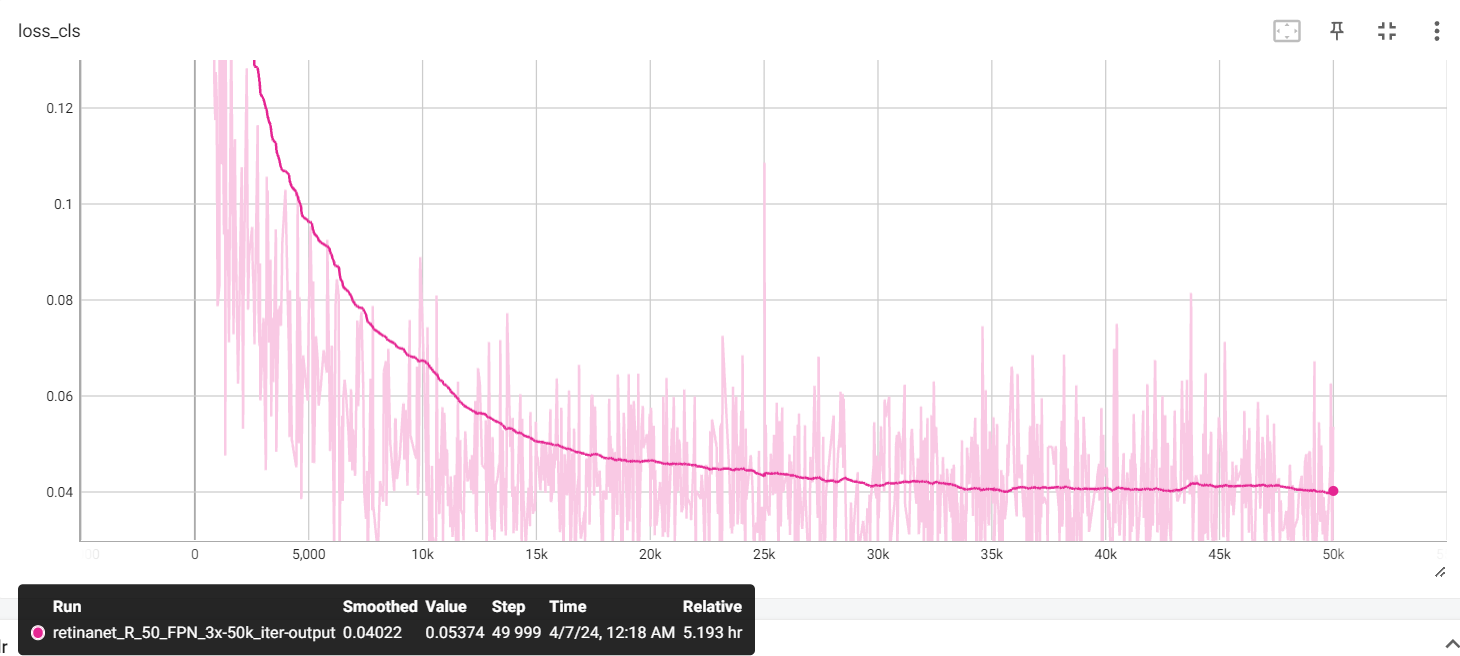
Графики: 

Рис. 27. График ошибки классификации модели RetinaNet (ResNet-50).

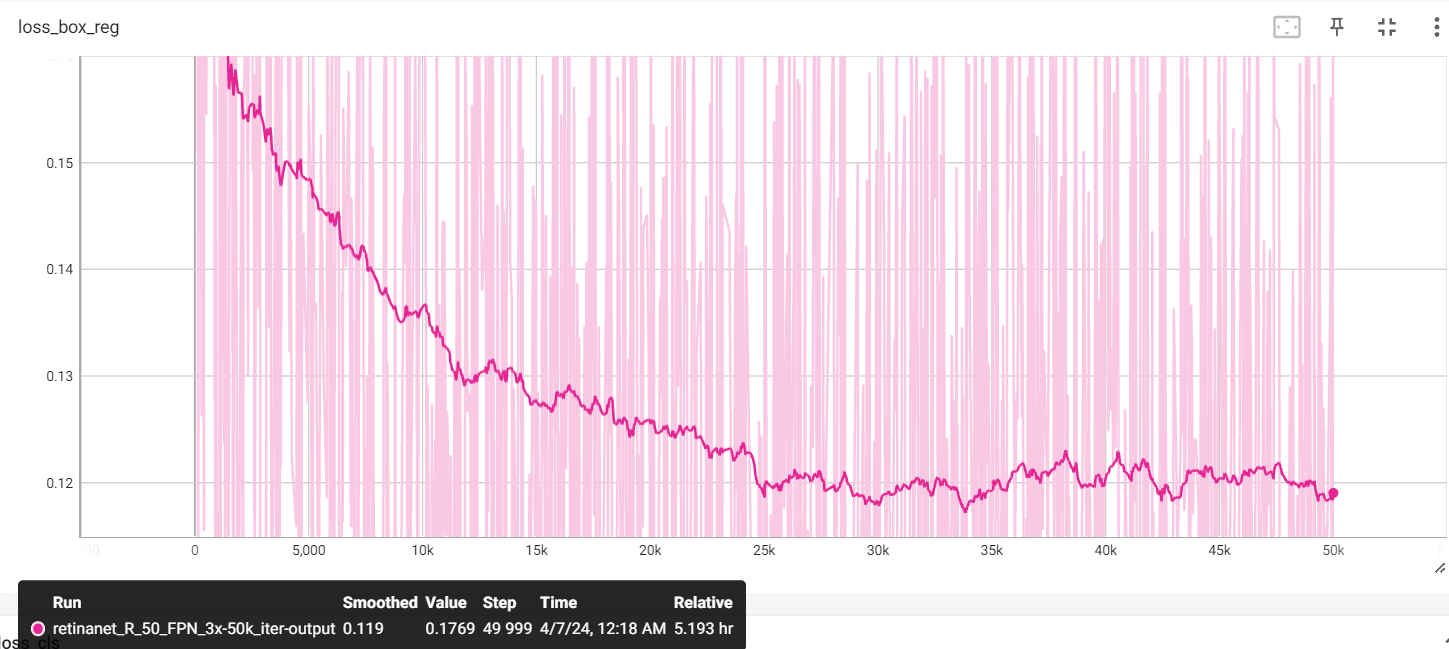


Рис. 28. График ошибки регрессии ограничивающих рамок модели RetinaNet (ResNet-50).

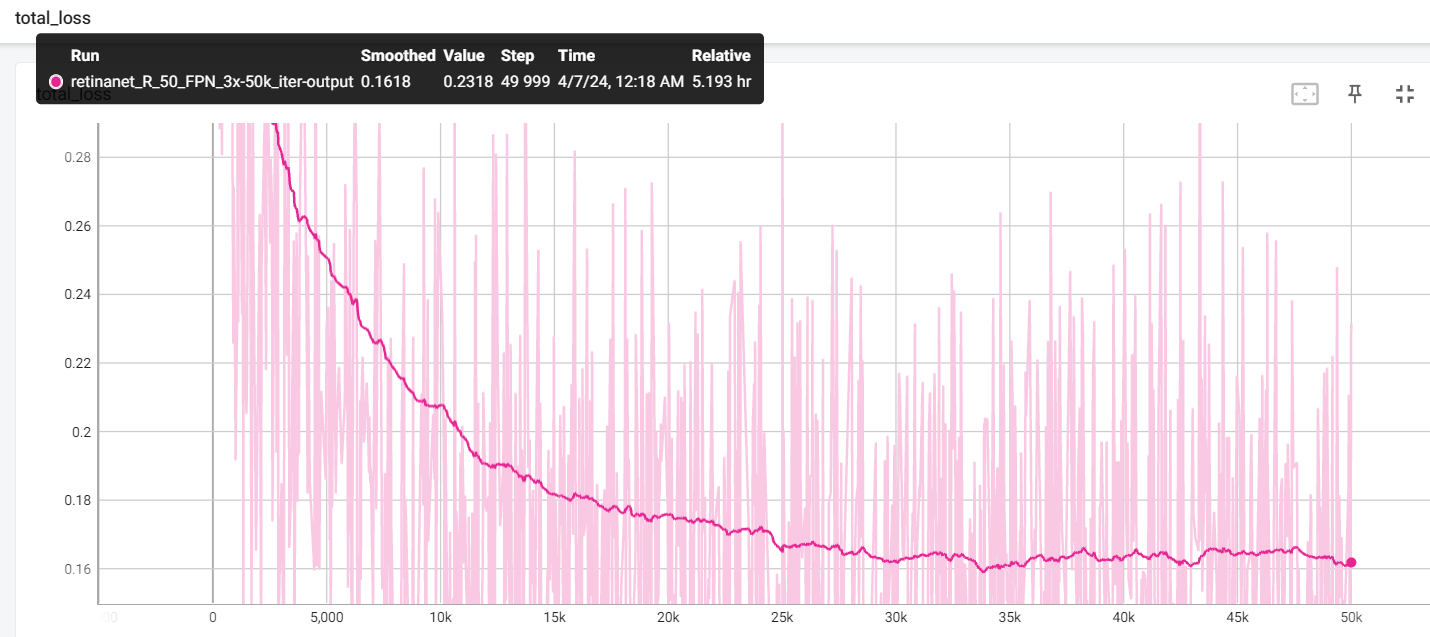


Рис. 29. График общей функции потерь модели RetinaNet (ResNet-50).

Как видно из графиков – значения сильно колеблются. Для того, чтобы разглядеть тенденцию, их приходится смягчать на максимальный уровень. Это может свидетельствовать о шуме в данных, малом размере батча, высокой скорости обучения и других факторах. Тем не менее, модель достигает неплохих результатов. Как видно, ошибка классификации снижается до 0.04, замирая примерно на 30 тысячах итераций. В то же время ошибка регрессии ограничивающих рамок перестаёт эволюционировать примерно после 25 тысяч итераций, и нормализуется на отметке 0.12. Общая функция потерь, в то же время, достигает отметки 0.16, прекращая снижаться с 30 тысяч итераций.

Анализ метрик:

1. Наличие шума/проблем в процессе обучения. "Рваные" графики, требующие сильного сглаживания, указывают на нестабильность процесса обучения. Это может быть вызвано шумом в данных, малым размером батча, высокой скоростью обучения или сложным ландшафтом функции потерь
2. Сходимость модели. Несмотря на колебания, наблюдается общая тенденция к снижению всех метрик, что свидетельствует о сходимости модели. Ошибка классификации (loss\_cls) и ошибка регрессии ограничивающих рамок (loss\_box\_reg) снижаются, хотя и с разной скоростью
3. Потенциальное плато. Около 30 тысяч итераций наблюдается замедление снижения метрик, особенно loss\_cls и total\_loss. Это может указывать на достижение плато в обучении, где дальнейшее улучшение модели затруднено
4. Дисбаланс обучения. loss\_box\_reg стабилизируется на более высоком уровне, чем loss\_cls. Это может свидетельствовать о том, что модель испытывает больше трудностей с точным предсказанием координат ограничивающих рамок, чем с классификацией объектов.

Таким образом, модель RetinaNet продемонстрировала сходимость и достигла приемлемых результатов в задаче обнаружения объектов. Однако, анализ метрик выявил нестабильность в процессе обучения и потенциальное наличие плато производительности. Наблюдается дисбаланс в обучении, где модель испытывает больше трудностей с точной локализацией объектов, чем с их классификацией.

* + 1. RetinaNet ResNet-101 FPN

После обучения RetinaNet с 50-слойной ResNet будет обучена аналогичная модель, содержащая вдвое больше слоёв. Это покажет тенденцию – склонна ли более сложная модель схожей архитектуры достигнуть разительно более высоких результатов, или же они останутся в пределах погрешности.

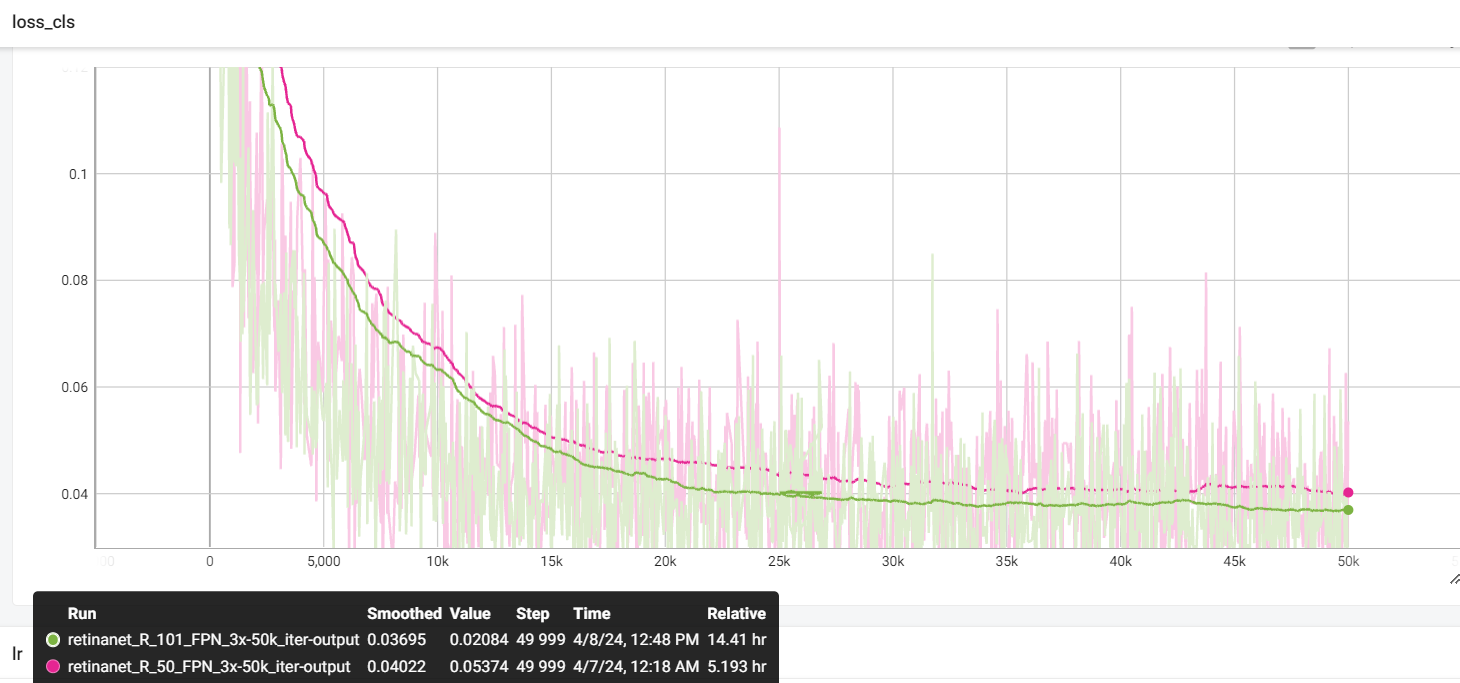
Итак, обучение RetinaNet с ResNet-101 заняло 14 с половиной часов. Результаты относительно меньшей модели представлены на графиках: 

Рис. 30. Ошибка классификации модели RetinaNet (ResNet-101).

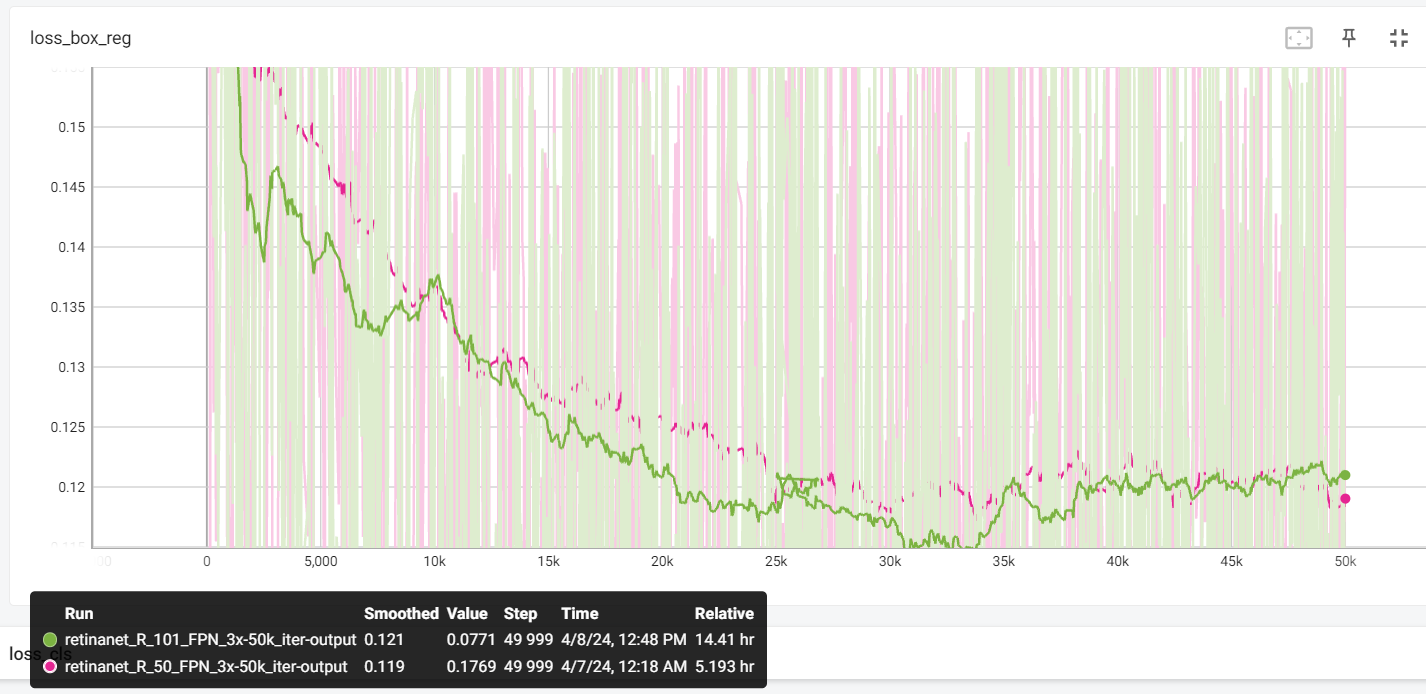


Рис. 31. Ошибка ограничивающих рамок модели RetinaNet (ResNet-101).

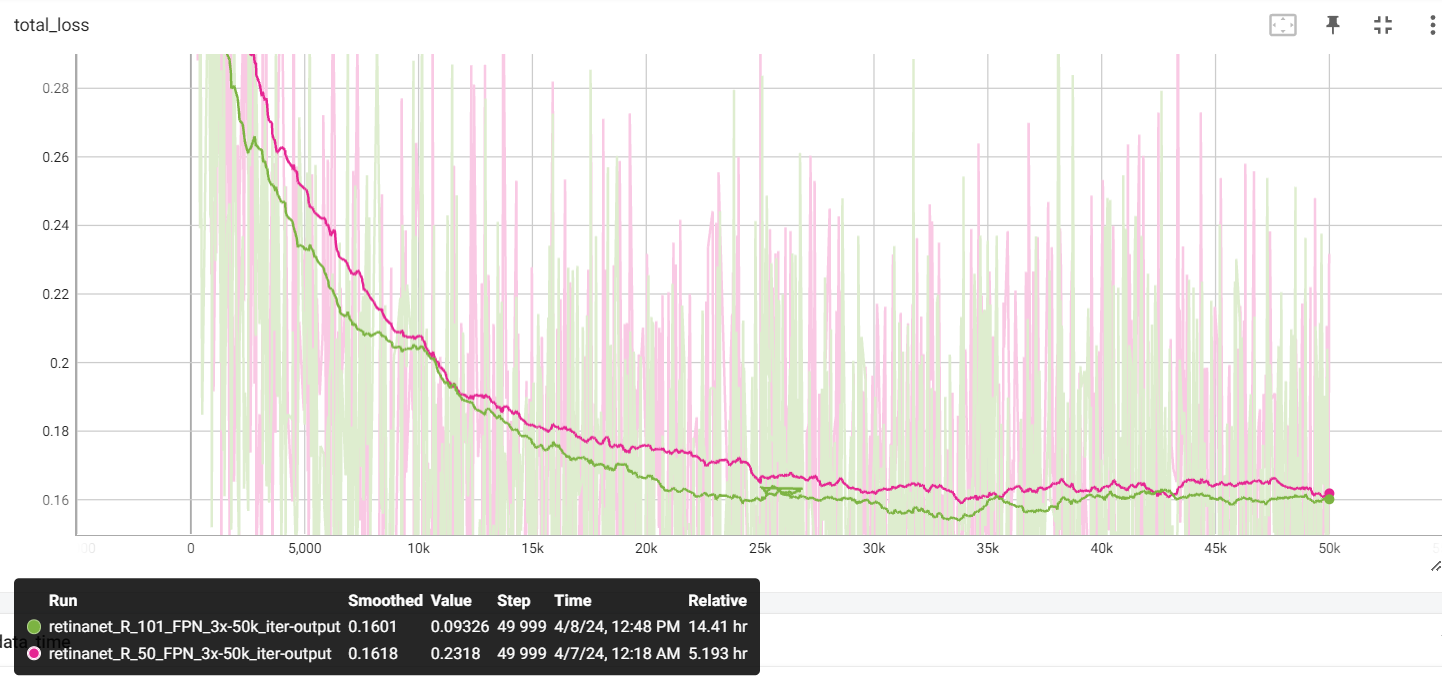


Рис. 32. График общей функции потерь модели RetinaNet (ResNet-101).

Из графиков можно сделать вывод, что более сложная модель не достигла значительных улучшений – значения расходятся в пределах погрешности. RetinaNet с более мощной архитектурой демонстрирует те же самые проблемы, что и меньшая.

Таким образом, использовании более мощной базовой ResNet-101 модели RetinaNet в задачи классификации и локализации колоний микроорганизмов в чашке Петри не имеет смысла.

* + 1. Faster R-CNN ResNet-50 FPN
    2. Faster R-CNN ResNet-101 FPN
    3. Faster R-CNN ResNeXt-101 32x8d FPN
    4. Mask R-CNN ResNet-101 FPN
    5. Cascade R-CNN ResNet-50 FPN
    6. Cascade R-CNN ResNeXt-152 32x8d FPN
    7. Итоговый анализ и выбор лучшей модели
  1. Разработка интеллектуального сервиса
  2. Интеграция в микробиологию

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Информационно-правовой портал [Электронный ресурс]. – Режим доступа: http://government.ru/docs/all/138589/ (дата обращения: 26.11.2023)
2. Минцифры России [Электронный ресурс]. – Режим доступа: <https://ит-гранты.рф/novolabsystem>, сноска “О проекте” (дата обращения: 26.11.2023)
3. Платформа Saturn Cloud [Электронный ресурс]. – Режим доступа: <https://saturncloud.io/glossary/onnx-open-neural-network-exchange/>, (дата обращения: 15.03.2024)
4. Хамидулин Т.Г. Сравнение программного обеспечения глубокого обучения. // Вестник Поволжского Государственного Университета. Серия: Технические науки. – 2022 г. (дата обращения: 16.03.2024)
5. Mateusz Opala. Deep Learning Frameworks Comparison – Tensorflow, PyTorch, Keras, MXNet, The Microsoft Cognitive Toolkit, Caffe, Deeplearning4j, Chainer/ [Электронный ресурс]: Режим доступа: https://www.netguru.com/blog/deep-learning-frameworks-comparison, – Загл. с экрана. (дата обращения: 16.03.2024)
6. Hands-On AI Part 5: Select a Deep Learning Framework / [Электронный ресурс]: Режим доступа: https://software.intel.com/en-us/articles/hands-on-aipart-5-select-a-deep-learning-framework, – Загл. с экрана. (дата обращения: 16.03.2024)
7. Тематический сайт с научными статьями Habr [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/709432/>, (дата обращения: 17.03.2024)
8. Вроцлавский научно-технический университет [Электронный ресурс]. – Режим доступа: <https://agar.neurosys.com/>, (дата обращения: 13.04.2024)