Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»
Институт радиоэлектроники и информационных технологий-РТФ

ДОПУСТИТЬ К ЗАЩИТЕ

_____ А.В.Созыкин

(подпись) (Ф.И.О.)

«_____»_____2023 г.

ИТОГОВАЯ АТТЕСТАЦИОННАЯ РАБОТА

по программе профессиональной переподготовки
«Разработка приложений искусственного интеллекта»

на тему: Разработка веб-приложения анализа изображений

Слушатели: Вдовкина Елена Геннадьевна
Журенков Олег Викторович
Калинникова Анна Дмитриевна
Шелюгина Ольга Александровна
Группа: РПИ-1
Руководитель итоговой аттестационной работы: Созыкин Андрей Владимирович

Екатеринбург
2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Одной из базовых задач в машинном зрении является задача классификации изображения – определения категорий объектов, который находится на изображении. Термин «классификация изображений» относится к широкому классу задач, входными данными для которых являются изображения, а выходными могут быть как изображения, так и наборы связанных с ними характерных признаков. Существует множество вариантов: классификация, сегментация, аннотирование, обнаружение объектов и т. п. В данной работе используется классификация изображений, которая представляет собой задачу присвоения входному изображению одной метки из фиксированного набора. Это одна из базовых проблем компьютерного зрения, которая, несмотря на свою простоту, имеет большое разнообразие практических применений. Более того, многие другие, казалось бы, различные задачи компьютерного зрения (такие как обнаружение объектов, сегментация) могут быть сведены к классификации изображений.

Сфера применения технологии распознавания образов очень широка, и тематика проектов, основанных на компьютерном зрении, очевидно будет только расширяться. Конкретная реализация этих многочисленных проектов требует не только сложной, ресурсоемкой проектной деятельности, но и быстрых решений, использования удобных высокоуровневых технологий, дающих возможность быстрой сборки приложения для распознавания образов.

Объектом данного исследования являются технологии быстрого конструирования веб-приложений на основе модели глубокого обучения для классификации объектов на изображении. Цель работы – разработка на основе предварительно обученной нейросетевой модели web-приложения классификации графических изображений для использования в образовательном процессе.

Задачи:

- - составить обзор технологий распознавания образов и выбрать подходящую предварительно обученную модель машинного обучения;
- - составить обзор инструментов создания приложения для задач машинного обучения и избрать инструменты для разработки web-приложения классификации графических изображений;
- - организовать командную работу над созданием web-приложения;
- - реализовать web-приложение классификации изображений на основе предварительно обученной модели машинного обучения;
- - разместить код приложения в репозитории GitHub.

Данная работа содержит анализ методов и инструментов решения задачи, основанных на глубоком обучении, использующихся при решении задач определения объектов на изображении. Также в работе раскрыты особенности проектирования и реализации web-приложения, определены возможные сферы использования приложения. Результатом выполнения работы является разработанное web-приложение, классифицирующее и определяющее объекты на изображениях.

1. ОБЗОР ТЕХНОЛОГИЙ АНАЛИЗА ИЗОБРАЖЕНИЙ

1.1. Обзор задачи классификации изображений

Распознавание изображений является одним из методов, который широко используется в настоящее время. Из-за своей популярности и постоянного использования он сталкивается со многими сложными проблемами. Эти проблемы заключаются в следующем:

1) Искажение.

Объекты не меняются, даже если они искажены. Система учится на исходном изображении и формирует представление о том, что этот объект может иметь только определенную форму. В реальном мире форма меняется, и в результате возникают неточности, когда система сталкивается с искаженным изображением объекта.

2) Межклассовая вариация.

Определенное изменение объекта внутри класса. Они могут быть разного размера, формы, но все равно представляют один и тот же класс. Например, бутылки, пуговицы, сумки, стулья бывают разных размеров и внешнего вида.

3) Изменение точки зрения.

Когда изображения (в которых объекты выровнены в другом направлении) передаются в систему, она предсказывает неточные значения. Система не может понять, что изменение выравнивания изображения, такого как левое, правое, нижнее и верхнее, не изменит его, и это создает проблемы при распознавании.

4) Изменение масштаба.

На классификацию объекта влияет изменение размера объекта. Чем ближе мы рассматриваем объект, тем больше он кажется в размерах, и наоборот.

5) Окклюзия.

Некоторые объекты препятствуют просмотру изображения в полном объеме и приводят к тому, что система получает неполную информацию. Необходимо разработать алгоритм, чувствительный к этим вариациям и содержащий широкий диапазон выборок данных.

Задача классификации изображений состоит в том, чтобы взять массив пикселей, представляющий одно изображение, и присвоить ему метку. Все этапы могут быть формализованы следующим образом:

- Ввод: входные данные состоят из набора N изображений, каждое из которых помечено одним из K различных классов. Эти данные называются обучающей выборкой.
- Обучение: задача состоит в том, чтобы использовать обучающую выборку, для выяснения, как выглядит каждый из классов. Этот шаг называется обучением классификатора или обучением модели.
- Оценка: в конце концов, оценивается качество классификатора, путем попыток предсказания меток для нового набора изображений, которые не присутствовали в обучающей выборке. После чего сравниваются истинные метки этих изображений с предсказанными классификатором. Хорошо обученный классификатор должен обеспечить то, что многие предсказания совпадут с правдивыми ответами (которые называются истинными).

Классификация изображений – это контролируемая задача обучения. Классификация изображений выполняется с помощью предварительно обученной модели.

Хорошая модель классификации изображений должна быть инвариантна к различным вариациям перечисленных выше проблем, сохраняя при этом чувствительность к межклассовым различиям.

1.2. Обзор предварительно обученных моделей машинного обучения

Предварительно обученные модели — это модели машинного обучения, которые были обучены на больших объемах данных с целью решения конкретных задач. Такие модели обучаются на огромных объемах данных (в том числе на эталонном наборе данных ImageNet), чтобы получить представление о мире и отношениях между различными факторами, которые могут влиять на конечный результат. Предварительно обученные модели используются для решения различных задач, таких как распознавание изображений, обработка естественного языка, генерация текста и другие.

Отсутствие необходимости полного обучения моделей на новых данных позволяет сократить время и затраты на обучение моделей. Кроме того, предварительно обученные модели могут быть использованы для извлечения признаков из данных, что может быть полезно для различных задач, таких как классификация и кластеризация данных.

В число наиболее известных предварительно обученных моделей входят GPT (Generative Pre-trained Transformer) для генерации текста, BERT (Bidirectional Encoder Representations from Transformers) для обработки естественного языка, VGG-19, Inceptionv3, ResNet50, EfficientNet, которые можно использовать для задач классификации изображений.

VGG-19 – предварительно обученная модель глубокого обучения для распознавания изображений, которая была разработана исследовательской группой Visual Geometry Group (VGG) в Университете Оксфорда. Эта модель является улучшенной версией более ранней модели VGG-16.

VGG-19 использует сверточные нейронные сети (CNN) для обработки изображений и распознавания объектов на них. Она состоит из 19 слоев, включая 16 сверточных слоев и 3 полносвязных слоя, а также 5 слоев MaxPool и 1 слой SoftMax.

Для обучения сети VGG-19 использовалось более чем 1 миллиона изображений из базы данных ImageNet. Естественно, можно импортировать модель с обученными весами от ImageNet. Эта предварительно обученная сеть может классифицировать до 1000 объектов. Сеть обучалась на цветных изображениях размером 224×224 пикселей.

Одной из примечательных особенностей VGG-19 является однородная архитектура, при которой все сверточные слои имеют одинаковое количество фильтров и одинаковый размер фильтра. Это облегчает понимание и воспроизведение сетевой архитектуры. Еще одна особенность VGG-19 заключается в том, что он использует меньшие фильтры по сравнению с другими популярными архитектурами, такими как AlexNet, что делает эту модель более эффективным.

Модель VGG-19 достигла высокой производительности в различных задачах компьютерного зрения, включая классификацию изображений, обнаружение объектов и сегментацию изображений.

Inceptionv3 (GoogLeNet) – архитектура глубокой сверточной нейронной сети, которая была представлена Google в 2015 году. Это третья итерация архитектуры Inception, она предназначена для задач классификации и распознавания изображений. Модель Inceptionv3 характеризуется использованием так называемых "модулей начала" ("Inception modules"), которые позволяют сети изучать как локальные, так и глобальные особенности изображения. Модуль начала состоит из нескольких сверточных фильтров разных размеров, применяемых к одному и тому же входу. Выходные данные каждого фильтра объединяются и используются в качестве входных данных для следующего слоя. Inceptionv3 также использует пакетную нормализацию и различные методы регуляризации для повышения производительности и предотвращения переобучения. Модель имеет относительно глубокую архитектуру с 48 слоями обучаемых параметров. Размер входного изображения этой сети составляет 299×299 пикселей, что больше, чем у сети VGG19. Модель Inceptionv3 достигла высоких результатов в задаче классификации ImageNet в 2015 году, обойдя при этом VGG-16 и ResNet-50. Она используется в широком спектре других задач компьютерного зрения, включая обнаружение объектов, семантическое сегментирование и визуальное ответы на вопросы. Кроме того, Inceptionv3 использовалась в качестве предварительно обученной модели для трансферного обучения по различным задачам, что позволяет исследователям и практикам извлечь выгоду из способности сети изучать общие функции изображения.

ResNet50 (Residual Network) – архитектура глубокой сверточной нейронной сети, которая была представлена Microsoft в 2015 году. Модель является частью семейства архитектур ResNet, которые предназначены для решения проблемы исчезновения градиентов в очень глубоких нейронных сетях. ResNet50 состоит из 50 слоев обучаемых параметров, включая серию сверточных слоев, слоев пакетной нормализации и остаточные блоки (residual blocks).

Остаточные блоки являются ключевой особенностью ResNet50 и позволяют сети изучать остаточные функции, которые сопоставляют входные данные непосредственно с выходами, минуя промежуточные уровни. Это облегчает сети изучение глубоких представлений и позволяет избежать проблемы исчезновения градиентов.

Модель ResNet50 широко используется в качестве предварительно обученной модели для трансферного обучения по широкому кругу задач компьютерного зрения. Архитектура была адаптирована и расширена для создания других моделей ResNet, таких как ResNet101 и ResNet152, которые имеют еще больше уровней и лучшую производительность для некоторых задач. Несмотря на то, что сеть ResNet50 имеет меньше параметров, она показывает более высокие результаты, чем VGG-19.

EfficientNet - это семейство высокопроизводительных нейронных сетей, применяемых в широком спектре задач компьютерного зрения. Модели EfficientNet были впервые представлены в статье Mingxing Tan and Quoc V. Le в 2019 году [18]. Ключевая идея EfficientNet заключается в более принципиальном масштабировании нейронных сетей, а не просто увеличении их глубины, ширины или разрешения. Авторы предложили метод комплексного масштабирования, который равномерно масштабирует глубину, ширину и разрешение сети, контролируя при этом вычислительные затраты. Это позволяет им достичь более высокой точности, чем существующие модели, используя меньше параметров и меньше вычислений.

Семейство EfficientNet состоит из нескольких моделей, от EfficientNet-B0 до EfficientNet-B7, каждая из которых имеет разный уровень вычислительной стоимости и точности. Модели были предварительно обучены большим наборам данных изображений, таким как ImageNet, и точно настроены для конкретных задач, таких как обнаружение объектов, семантическая сегментация и классификация изображений. Было показано, что EfficientNet превосходит другие современные модели, такие как ResNet и DenseNet, на нескольких тестах компьютерного зрения, будучи намного меньше и быстрее.

Рисунок 1 – Архитектура начальной модели EfficientNet-B0

Архитектура, представленная на рис. 1, включает MBConv (от Mobile Inverted Residual Bottleneck Convolution), ключевые компоненты архитектуры EfficientNet. Это тип глубокоразделимой свертки, которая предназначена для того, чтобы быть более эффективной и точной, чем традиционные свертки. Блок MBConv состоит из трех основных операций: глубинная свертка, точечная свертка и линейное соединение с узким местом. Глубинная свертка применяет отдельный фильтр к каждому каналу входа, уменьшая количество параметров и требуемые вычисления. Затем точечная свертка применяет свертку 1x1 для объединения выходов глубинной свертки, позволяя применять нелинейные преобразования к картам признаков. Наконец, линейное соединение с узким местом соединяет вход с выходом, позволяя информации проходить непосредственно через блок без преобразования. Основным преимуществом блока MBConv является то, что он высокоэффективен с точки зрения вычислений и использования памяти, сохраняя при этом высокую точность задач компьютерного зрения. В целом, MBConv стал популярным строительным блоком для многих архитектур нейронных сетей, особенно в области компьютерного зрения, благодаря своей эффективности и результативности.

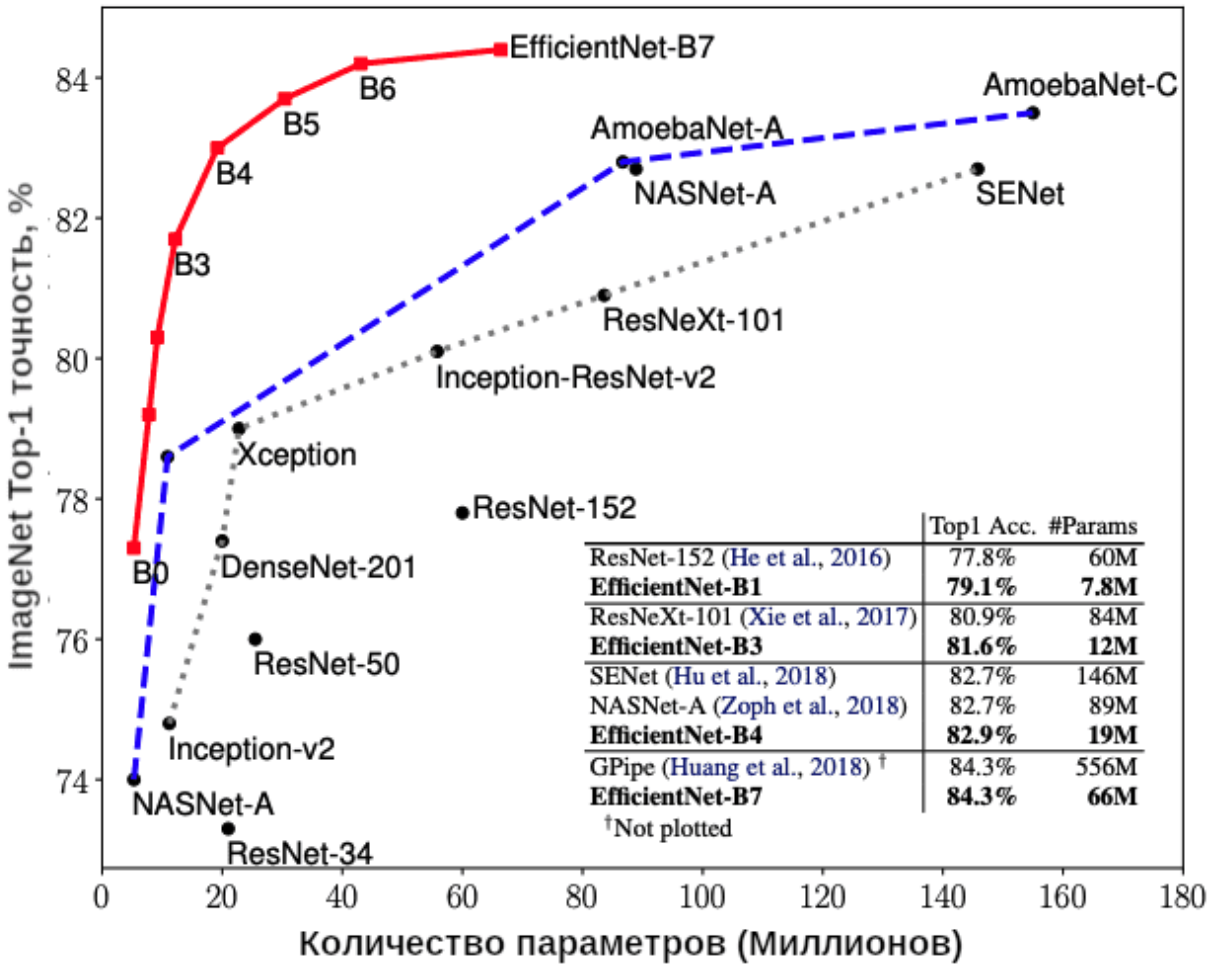


Рисунок 2 – Оценки точности моделей Machine Learning

Существует восемь реализаций EfficientNet — от B0 до B7, отсчитываются по мере увеличения сложности архитектуры (рисунок 2). Но даже самая первая из них — EfficientNetB0 — показывает достойные результаты. EfficientNetB0 является самым маленьким и легким вариантом семейства EfficientNet..

EfficientNetB0 состоит из серии сверточных слоев, включая несколько блоков MBConv, за которыми следует глобальный слой подвыборки и полносвязный выходной слой. Модель разработана таким образом, чтобы быть масштабируемой, что означает, что ее можно легко изменить, чтобы соответствовать широкому диапазону размеров входных изображений без ущерба для производительности.

EfficientNetB0 прошла предварительное обучение на больших наборах данных изображений, таких как ImageNet. В целом, EfficientNetB0 - это мощная и эффективная архитектура нейронных сетей, которая стала популярным выбором для практического применения в промышленности и научных кругах. Ее

способность достигать высокой точности, требуя при этом меньшего количества параметров и меньше вычислений, чем другие модели, делает EfficientNetB0 привлекательным вариантом для использования с ограниченными ресурсами. Именно эта модель была избрана для реализации приложения распознавания изображений.

1.3. Обзор инструментов создания приложения машинного обучения

Фреймворки машинного обучения — один из источников натренированных моделей. Существует множество фреймворков машинного обучения, которые помогают разработчикам быстро и эффективно создавать и обучать модели машинного обучения. Вот некоторые из самых популярных:

Tensorflow

TensorFlow - это библиотека программного обеспечения с открытым исходным кодом для численных вычислений и машинного обучения, разработанная Google Brain Team. Целью разработчиков было создание гибкого, масштабируемого и портативного программного обеспечения. TensorFlow позволяет исследователям и разработчикам создавать и развертывать модели машинного обучения для широкого спектра приложений, от распознавания изображений и речи до обработки естественного языка и робототехники.

TensorFlow предоставляет интерфейс программирования для создания и обучения моделей машинного обучения, основанный на концепции графов потоков данных. Эти графы описывают, как данные перемещаются по сети узлов, каждый из которых выполняет математическую операцию с входными данными. TensorFlow позволяет пользователям определять эти графы с помощью высокоуровневого API на Python или других поддерживаемых языках, а затем эффективно выполнять их на процессорах, графических процессорах или других аппаратных ускорителях.

Одной из ключевых особенностей TensorFlow является его способность автоматически вычислять градиенты для сложных математических операций, что позволяет легко обучать глубокие нейронные сети с большим количеством слоев.

TensorFlow также включает в себя множество готовых моделей и инструментов для общих задач машинного обучения, а также интеграцию с другими популярными библиотеками и фреймворками.

В целом, TensorFlow стала одной из самых популярных и широко используемых библиотек машинного обучения с большим и активным сообществом разработчиков и исследователей. Для нее написано большое количество руководств и документации.

В дополнение к своей основной функциональности, TensorFlow также включает в себя ряд API и инструментов более высокого уровня, которые облегчают создание и обучение моделей машинного обучения. Например, TensorFlow Estimators предоставляют высокоуровневый API для построения и обучения моделей с использованием готовых алгоритмов и функций предварительной обработки. TensorFlow Hub позволяет пользователям легко искать и повторно использовать предварительно обученные модели из различных источников.

TensorFlow также поддерживает распределенное обучение, позволяя пользователям обучать модели на больших наборах данных на нескольких машинах или кластерах. Это может значительно сократить время обучения и повысить производительность модели для сложных задач.

В целом, TensorFlow сыграл ключевую роль в развитии современного машинного обучения и помог сделать его более доступным для более широкого круга пользователей. Его постоянное развитие и поддержка со стороны сообщества гарантируют, что он будет оставаться основным инструментом в области искусственного интеллекта на долгие годы. К недостаткам этой платформы можно отнести более высокий входной порог для начинающих, чем другие популярные решения — PyTorch или Keras, а также есть ограничения по поддержке языков, полностью поддерживаемый язык – Python.

PyTorch

PyTorch - это библиотека машинного обучения с открытым исходным кодом. Он предназначен для обеспечения гибкой и эффективной платформы для создания и обучения моделей глубокого обучения. PyTorch построен вокруг

концепции динамических вычислительных графов, которая позволяет разработчикам определять и изменять вычислительные графы на лету во время выполнения. Это позволяет легко создавать сложные модели с динамическими или переменными входами, такими как рекуррентные нейронные сети.

PyTorch также включает в себя ряд API более высокого уровня и инструментов для создания и обучения моделей машинного обучения. Например, фреймворк PyTorch TorchScript позволяет пользователям экспортировать модели PyTorch в портативный формат, который можно использовать в производственных средах. Платформа PyTorch Lightning предоставляет легкий интерфейс для создания и обучения моделей с использованием лучших практик, одновременно абстрагируя многие низкоуровневые детали.

Одной из ключевых особенностей PyTorch является простота использования и гибкость. Библиотека предоставляет простой и интуитивно понятный интерфейс для создания и обучения моделей, позволяя разработчикам настраивать и настраивать свои модели по мере необходимости. PyTorch также включает в себя ряд готовых моделей и инструментов для общих задач машинного обучения, а также интеграцию с другими популярными библиотеками и фреймворками.

Как и TensorFlow, PyTorch также поддерживает распределенное обучение, позволяя пользователям обучать модели на больших наборах данных на нескольких машинах или кластерах.

Keras

Keras — это библиотека глубокого обучения с открытым исходным кодом, написанная на Python. Она разработана таким образом, чтобы быть удобным, модульным и расширяемым инструментом, что позволяет разработчикам легко создавать и экспериментировать с различными архитектурами нейронных сетей. Библиотека была разработана инженером компании Google, Франсуа Шолле, в целях ускорения экспериментов.

Keras предоставляет высокоуровневый API для создания и обучения моделей глубокого обучения, что облегчает новичкам начало работы с машинным обучением. Он поддерживает широкий спектр архитектур нейронных

сетей, включая сверточные сети, рекуррентные сети и комбинации этих архитектур.

Одной из ключевых особенностей Keras является его способность работать поверх различных бэкэндов, включая TensorFlow, Theano и CNTK. Это позволяет пользователям выбирать бэкэнд, который наилучшим образом соответствует их потребностям, и легко переключаться между бэкэндами без необходимости изменять свой код. Keras также включает в себя множество готовых моделей и инструментов для общих задач машинного обучения, таких как классификация изображений, обработка естественного языка и прогнозирование последовательностей. Кроме того, у него есть большое и активное сообщество разработчиков, которые внесли много дополнительных инструментов и расширений в библиотеку.

В целом, Keras стал популярным выбором для разработчиков и исследователей, которые хотят быстро создать прототипы и экспериментировать с моделями глубокого обучения благодаря простоте использования, гибкости и обширной документации. Keras разработан как модульная библиотека, что позволяет пользователям быстро создавать и изменять различные архитектуры нейронных сетей без необходимости переписывать всю свою кодовую базу. Keras обеспечивает обратную совместимость с предыдущими версиями, позволяя разработчикам легко переносить свой код на более новые версии без каких-либо серьезных изменений. Keras включает в себя встроенные инструменты для визуализации структуры и производительности нейронных сетей, облегчающие отладку и понимание поведения модели.

В целом, Keras - это мощная и гибкая библиотека глубокого обучения, доступная как начинающим, так и продвинутым пользователям. Его модульная конструкция, удобный API и обширная документация делают его популярным выбором для разработки моделей глубокого обучения на Python. Преимуществами также можно считать быстрое прототипирование, простой и интуитивно-понятный интерфейс, встроенная поддержка для обучения на нескольких GPU, поддерживает GPU от NVIDIA, TPU от Google, GPU с Open-CL,

такие как AMD. К недостаткам можно отнести слишком высокоуровневую архитектуру и отсутствие возможностей для тонкой настройки.

Caffe и Caffe2

Caffe и Caffe2 — это платформы глубокого обучения с открытым исходным кодом. Хотя обе структуры предназначены для создания и обучения глубоких нейронных сетей, они имеют некоторые ключевые различия в своей архитектуре и особенностях.

Caffe - это платформа глубокого обучения, которая оптимизирована для задач классификации изображений и используется во многих приложениях компьютерного зрения. Она использует статический граф вычислений для определения архитектур нейронных сетей и полагается на файл конфигурации на основе protobuf для указания структуры сети, слоев и гиперпараметров. Caffe поддерживает вычисления как CPU, так и GPU, а также предоставляет предварительно обученные модели для нескольких популярных задач классификации изображений, включая ImageNet.

Caffe2, с другой стороны, представляет собой более универсальную структуру глубокого обучения, которая поддерживает более широкий спектр задач, включая компьютерное зрение, обработку естественного языка и обучение с подкреплением. Платформа использует динамический график вычислений для определения архитектур нейронных сетей, что обеспечивает большую гибкость в построении и изменении сетей на лету. Caffe2 поддерживает вычисления как CPU, так и GPU и предоставляет несколько высокоуровневых API для создания и обучения моделей, включая популярный API PyTorch. К недостаткам можно отнести ограниченную поддержка сообществом.

В целом, как Caffe, так и Caffe2 являются мощными и гибкими платформами глубокого обучения, каждая из которых имеет свои сильные стороны и сценарии использования. В то время как Caffe оптимизирован для задач классификации изображений и использует статический граф вычислений, Caffe2 является более общей структурой, которая поддерживает более широкий спектр задач и использует график динамических вычислений.

Основные характеристики, которые могут быть важны при выборе фреймворка для машинного обучения:

1. Выбор на основе языка программирования: многие фреймворки предназначены для работы с определенными языками программирования, такими как Python или JavaScript. Выбор фреймворка может зависеть от того, какой язык программирования предпочитается использовать.
2. Важен тип задач, поддерживаемых фреймворком: многие фреймворки поддерживают различные типы задач машинного обучения, такие как классификация, регрессия, кластеризация, обработка естественного языка или компьютерное зрение и т. п.
3. Удобство использования.

Из различных фреймворков машинного обучения, каждый имеет свои преимущества и недостатки. В целом, фреймворки машинного обучения позволяют разработчикам создавать и обучать модели машинного обучения быстро и эффективно, используя различные алгоритмы, библиотеки и инструменты.

Некоторые фреймворки (TensorFlow, PyTorch) имеют большую и активную пользовательскую базу, которая обеспечивает поддержку и разработку новых функций и инструментов. Они также предоставляют различные уровни абстракции для упрощения работы с моделями машинного обучения, от более высокоуровневых API до более низкоуровневых возможностей оптимизации. Фреймворки Caffe и MXNet предоставляют возможности оптимизации для обработки изображений и масштабируемости для работы с несколькими GPU. Существуют фреймворки, например, Scikit-learn, которые предназначены для машинного обучения на высоком уровне и предоставляют готовые алгоритмы для обработки данных.

Один из самых удобных способов представить результаты работы в области машинного обучения – web-приложения. Многих специалистов по данным, не имеющих какого-либо опыта веб-разработки, отпугивает идея

создания подобных приложений. Однако с помощью фреймворков с этой задачей справиться намного легче.

Фреймворки – это программные продукты, которые упрощают создание и поддержку технически сложных либо нагруженных проектов. Они содержит только базовые программные модули, а все специфичные компоненты реализуются программистом на их основе. Таким образом достигается высокая скорость разработки и её упрощение.

Django

Django - это популярный веб-фреймворк с открытым исходным кодом для Python, предназначенный для того, чтобы помочь разработчикам быстро и эффективно создавать масштабируемые и поддерживаемые веб-приложения. Он обеспечивает высокоуровневую архитектуру «модель-представление-контроллер» (MVC) и богатый набор инструментов и библиотек для общих задач веб-разработки. Django предоставляет объектно-реляционный маппинг (ORM), который позволяет разработчикам взаимодействовать с базами данных, используя объекты Python вместо SQL-запросов, облегчая работу с базами данных и уменьшая объем повторяющегося кода.

Django включает готовый интерфейс администратора, который позволяет разработчикам управлять содержимым сайта и учетными записями пользователей без написания кода, гибкую систему маршрутизации URL-адресов, которая позволяет легко сопоставлять URL-адреса с конкретными представлениями и шаблонами, движок шаблонов Django, который позволяет разработчикам легко отображать HTML-шаблоны с динамическим контентом, встроенную систему обработки форм, которая упрощает проверку и обработку форм.

Django включает в себя множество встроенных функций безопасности, таких как защита от подделки межсайтовых запросов (CSRF), предотвращение SQL-инъекций и хеширование паролей.

Django разработан для масштабирования, что позволяет разработчикам легко добавлять новые функции и обрабатывать большие

Хорошая возможность в Django – выделять логически обособленный функционал приложения в отдельное приложение внутри фреймворка. По сути, проект легко конструируется из приложений. Каталог плагинов включает надстройки, которые решают разные задачи, в том числе развёртывание полноценной CMS и создания REST API.

Django развивается и поддерживается огромным сообществом. Django Software Foundation – это некоммерческая организация, которая поддерживает обучение фреймворку, его развитие и распространение.

Для небольших проектов функционал Django избыточен, для работы с легкими или асинхронными проектами обычно используют другие фреймворки.

Flask

Flask - это легкий и гибкий веб-фреймворк Python, который разработан так, чтобы быть простым и удобным в использовании. В отличие от Django, Flask не предоставляет много встроенных функций, что делает его более легким и гибким. Flask предоставляет простую и интуитивно понятную систему маршрутизации, которая позволяет разработчикам сопоставлять URL-адреса с функциями Python, которые генерируют ответы.

Flask поставляется со встроенным движком шаблонов под названием Jinja2, который позволяет разработчикам отделять логику приложения от HTML-кода.

Flask работает с большим количеством сторонних расширений, доступных для добавления таких функций, как аутентификация, интеграция с базами данных и многое другое. Flask включает в себя встроенную поддержку тестирования, что позволяет легко писать модульные тесты и интеграционные тесты для приложения.

Flask-RESTful — популярное расширение, которое позволяет разработчикам легко создавать RESTful API с помощью Flask.

В целом, Flask - это мощный и гибкий веб-фреймворк, который идеально подходит для создания небольших и средних веб-приложений. Он подходит для разработчиков, которые хотят быстро и легко создавать веб-приложения, не будучи привязанными к большому количеству встроенных функций и зависимостей.

Streamlit

Streamlit — это библиотека Python с открытым исходным кодом, предназначенная для создания интерактивных веб-приложений и визуализации данных. Это позволяет разработчикам быстро и легко создавать веб-приложения, используя знакомый код Python. Streamlit разработан, чтобы быть простым в использовании, с простым и интуитивно понятным API, который позволяет разработчикам быстро и легко создавать веб-приложения.

Streamlit предоставляет различные интерактивные виджеты, такие как ползунки, выпадающие меню и текстовые поля ввода, которые позволяют пользователям взаимодействовать с приложением. Фреймворк включает в себя различные инструменты визуализации данных, такие как диаграммы, графики и карты, которые позволяют разработчикам легко создавать визуализации данных.

Streamlit позволяет легко делиться веб-приложениями с другими, предоставляя простую команду для развертывания приложения в Интернете.

В целом, Streamlit - это мощный инструмент для создания интерактивных веб-приложений и визуализации данных на Python. Для создания приложений требуется написание нескольких строк кода с использованием предоставляемых фреймворком API. По мере внесения изменений в код приложение обновляется автоматически.

Streamlit позволяет легко и быстро создавать приложения и управлять ими. Streamlit является примитивом кэша, использующимся как постоянное, по умолчанию неизменное хранилище данных, которое позволяет приложениям Streamlit легко и безопасно повторно использовать информацию.

При работе Streamlit весь сценарий запускается заново для каждого взаимодействия с пользователем, при этом каждой переменной присваивается актуальное значение с учётом состояния виджета. Благодаря кэшированию Streamlit пропускает избыточные вычисления и выборки данных, что оптимизирует работу приложений.

Streamlit является бесплатной библиотекой с открытым исходным кодом, поэтому обслуживание созданных с её помощью приложений может осуществляться локально.

Виджеты в библиотеке Streamlit задаются при помощи API, представляющего собой набор подпрограмм, протоколов, функций и/либо команд, используемых для облегчения взаимодействия между различными программными службами. API-интерфейсы позволяют одному приложению получать доступ к данным из другого приложения или программы без необходимости знания о механизме его работы. Streamlit содержит справочник по API, организованный по типу действия, к примеру по отображению данных или оптимизации производительности. Каждый раздел включает связанные с типом действия методы и содержит примеры, что значительно облегчает создание приложений при помощи Streamlit.

Таким образом, Streamlit представляет собой фреймворк Python, позволяющий создавать удобные приложения с применением технологий машинного обучения без необходимости написания большого объёма кода и значительных временных затрат.

2. РАЗРАБОТКА ПРИЛОЖЕНИЯ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

2.1. Описание желаемых функций приложения

В данной работе реализуется приложение, которое будет способно распознавать и классифицировать изображение. Это могут быть изображения разных классов и тем, однако проектная идея предполагает разработку приложения для использования в образовательном процессе вуза, в процессе подготовки студентов творческих направлений — будущих дизайнеров, иллюстраторов, стилистов.

Web-приложение можно использовать для задач систематизации учебных работ, в том числе для следующих кейсов:

а) при оценивании студенческих работ зачастую необходимо убедиться в уникальности графического решения, вследствие этого возникает необходимость в создании цифрового архива студенческих работ по классу рисунка, живописи, компьютерной графики. Классификатор изображений поможет разметить тегами набор изображений. Теги помогут сориентироваться в наличии или отсутствии изображений по темам, схожим с предъявленной для оценивания работой;

б) созданная на основе классификации система тегов для изображений, демонстрирующих студенческие работы, может быть полезна в смысле систематизации методического фонда, обобщения работ, связанных с одними и теми же постановками, работ близкой тематики. Такая система удобна для поиска референсов - изображений, служащих источником вдохновения в творческой деятельностит.

Таким образом, создается web-приложение для аннотации изображений. Целью проекта на данном этапе является разработка инструмент для сегментации изображений, предусматривающий возможность загрузить изображение, провести сегментацию и сохранить результат на персональный компьютер. Однако для максимальной автоматизации удобно хранить уже аннотированные

изображения в самом приложении. В следующих проектных итерациях предполагается реализовать автоматическую аннотацию изображений, то есть процесс, в котором система автоматически присваивает метаданные цифровому изображению, используя соответствующие ключевые слова для описания его визуального содержания, и сохраняет их в том же приложении.

2.2. Описание архитектуры приложения

Разрабатываемое приложение должно классифицировать вводимые изображения используя EfficientNetB0. Для реализации проекта используются библиотеки Streamlit, TensorFlow и утилита PyEnv.

Для работы над приложением был выбран Keras - один из наиболее широко используемых фреймворков машинного обучения. Насчитывая около 2,5 миллионов разработчиков по состоянию на начало 2023 года, Keras находится в центре огромного сообщества и экосистемы.

Каждое приложение Keras принимает определенный тип предварительной обработки входных данных. Для эффективной сети предварительная обработка входных данных включена как часть модели (как уровень масштабирования), и, таким образом, tf.keras.applications.efficientnet.preprocess_input фактически является сквозной функцией. Эффективные новые модели ожидают, что их входные данные будут представлять собой тензоры пикселей с плавающей точкой со значениями в диапазоне [0-255].

Аргументы Keras:

include_top: следует ли включать полностью подключенный уровень в верхней части сети. По умолчанию установлено значение True.

weights: одно из значений None (случайная инициализация), 'imagenet' (предварительная подготовка в ImageNet) или путь к загружаемому файлу весов. По умолчанию используется значение "imagenet".

input_tensor: необязательный тензор Keras (т.е. вывод слоев.Input()) для использования в качестве входного изображения для модели.

input_shape: необязательный кортеж формы, указывается только в том случае, если include_top имеет значение False. У него должно быть ровно 3 входных канала.

pooling: Необязательный режим объединения для извлечения объектов, когда значение include_top равно False. По умолчанию значение равно None. - None означает, что выходные данные модели будут 4D-тензорными выходными данными последнего сверточного слоя. - среднее значение означает, что объединение глобального среднего значения будет применено к выходным данным последнего сверточного слоя, и, таким образом, выходными данными модели будет 2D-тензор. - max означает, что будет применен глобальный максимальный пул.

classes: Необязательное количество классов для классификации изображений, указывается только в том случае, если include_top имеет значение True и если аргумент weights не указан. Значение по умолчанию равно 1000 (количество классов ImageNet).

classifier_activation: str или вызываемый объект. Функция активации для использования на "верхнем" уровне. Игнорируется, если только include_top=True. Установите classifier_activation=None, чтобы вернуть логиты "верхнего" уровня. По умолчанию используется значение "softmax". При загрузке предварительно подготовленных весов classifier_activation может быть только None или "softmax".

EfficientNet, впервые представленная в Tan и Le в 2019 году, является одной из наиболее эффективных моделей (т.е. требует наименьшего количества ошибок для вывода), которая обеспечивает высочайшую точность как в imagenet, так и в обычных задачах обучения передаче классификации изображений.

Самая маленькая базовая модель похожа на MnasNet, которая достигла near-SOTA со значительно меньшей моделью. Внедряя эвристический способ масштабирования модели, EfficientNet предоставляет семейство моделей (от B0 до B7), которые представляют собой хорошее сочетание эффективности и точности в различных масштабах. Такая эвристика масштабирования (compound-scaling, подробности см. Tan and Le, 2019) позволяет базовой модели,

ориентированной на эффективность (B0), превосходить модели в любом масштабе, избегая при этом обширного поиска гиперпараметров по сетке.

Существуют различные варианты EfficientNet начиная B0 заканчивая B7. При работе над данным приложением была использована EfficientNetB0. Далее будут рассмотрены особенности "составного масштабирования".

Во первых то какое разрешение будет применяться, так например разрешения, не кратные 8, 16 и т.д., приводят к заполнению нулем границ некоторых слоев, что приводит к пустой трате вычислительных ресурсов. Это особенно относится к меньшим вариантам модели, поэтому входное разрешение для B0 и B1 выбрано равным 224 и 240.

Во вторых также важно учитывать такие параметры как глубина и ширина потому что строительные блоки EfficientNet требуют, чтобы размер канала был кратен 8.

В третьих существует такая вещь как ограничение ресурсов: ограничение памяти может привести к ограничению разрешения, когда глубина и ширина все еще могут увеличиваться. В такой ситуации увеличение глубины и/или ширины при сохранении разрешения все равно может повысить производительность.

В результате глубина, ширина и разрешение каждого варианта моделей EfficientNet были подобраны вручную и доказали, что дают хорошие результаты, хотя они могут значительно отличаться от формулы комплексного масштабирования.

Таким образом, реализация keras предоставляет только эти 8 моделей, вместо того, чтобы разрешать произвольный выбор параметров ширины / глубины / разрешения.

Особенности внедрения Keras. Реализация Efficient Net B0 поставляется с tf.keras начиная с TF2.3. Чтобы использовать EfficientNet B0 для классификации 1000 классов изображений из imagenet, было необходимо запустить

```
56 lines (46 sloc)  2 KB
```

```python
import io
import streamlit as st
from PIL import Image
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.applications.efficientnet import decode_predictions
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import EfficientNetB0
import numpy as np


@st.cache(allow_output_mutation=True)
def load_model():
    return EfficientNetB0(weights='imagenet')

```

Эта модель принимает входные изображения формы (224, 224, 3), и входные данные должны находиться в диапазоне [0, 255]. Нормализация включена как часть модели.

Потому что обучение эффективной сети требует огромного количества ресурсов и нескольких методов, которые не являются частью самой архитектуры модели. Следовательно, реализация Keras по умолчанию загружает предварительно обученные размеры, полученные с помощью обучения с авто аугментацией.

Модель ожидает, что их входные данные будут представлены в виде пикселей тензора с плавающей точкой со значениями в диапазоне от 0 до 255. Для базовой модели B0 входные формы равны 224.

```python
@st.cache(allow_output_mutation=True)
def load_model():
    return EfficientNetB0(weights='imagenet')


def preprocess_image(img):
    img = img.resize((224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    return x
```

В качестве рабочего инструмента по созданию приложений был выбран Streamlight так как, эта библиотека Python с открытым исходным кодом, позволяет легко создавать пользовательские веб-приложения для машинного обучения. Для нормальной работы библиотеки Stremlit был установлен Python 3.9.16

В качестве пользовательского клиента при работе с приложением, как оптимальный вариант, было принято решение использовать браузер.

2.3. Описание реализации приложения

1. На первом шаге были импортированы все необходимые библиотеки. Импортированы библиотеки tensorflow, keras, numpy, следующим образом. Также подключаем EfficientNetB0.

```
import io
import streamlit as st
from PIL import Image
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.applications.efficientnet import decode_predictions
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import EfficientNetB0
import numpy as np

```

2. Были заданы размеры входных форм, которые равны 224, так как была использована модель EfficientNetB0. Функция выполняет предварительную обработку изображения для подготовки к распознаванию.

```
@st.cache(allow_output_mutation=True)
def load_model():
    return EfficientNetB0(weights='imagenet')


def preprocess_image(img):
    img = img.resize((224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    return x
```

3. Средствами Streamlit создаётся простое Web-приложение, которое позволяет загрузить изображение.

```
def load_image():
    """Создание формы для загрузки изображения"""
    # Форма для загрузки изображения средствами Streamlit
    uploaded_file = st.file_uploader(
        label='Выберите изображение для распознавания')
    if uploaded_file is not None:
        # Получение загруженного изображения
        image_data = uploaded_file.getvalue()
        # Показ загруженного изображения на Web-странице средствами Streamlit
        st.image(image_data)
        # Возврат изображения в формате PIL
        return Image.open(io.BytesIO(image_data))
    else:
        return None

```

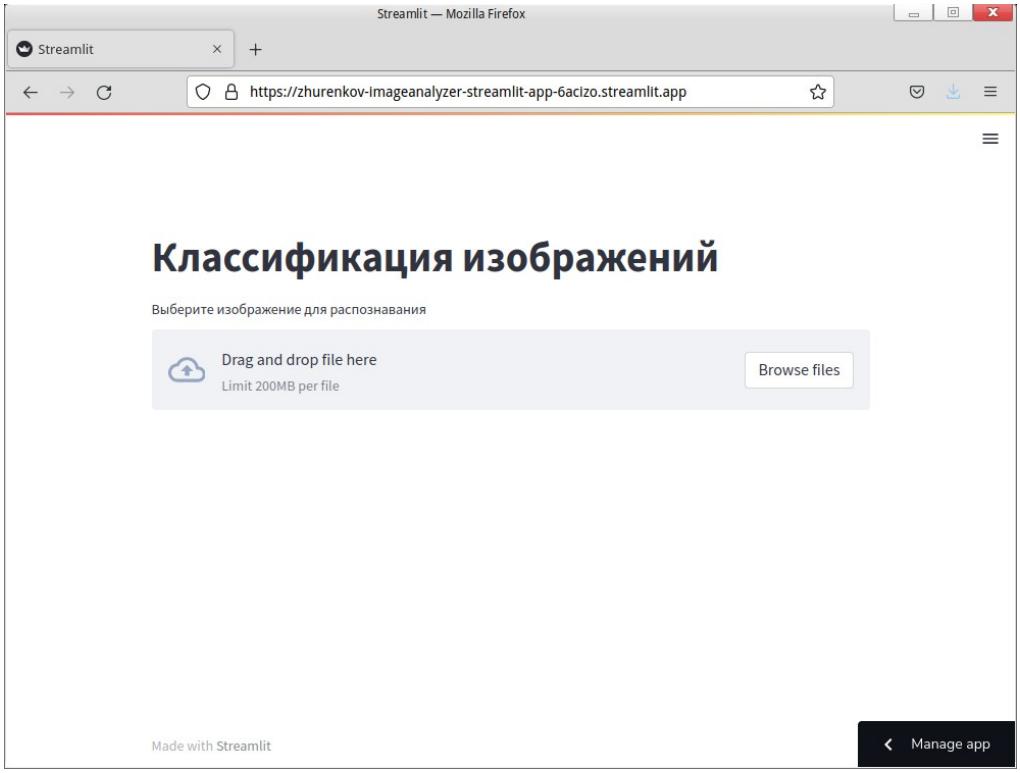
5. Были написаны 3 класса с самой большой вероятностью.

```
def print_predictions(preds):
    classes = decode_predictions(preds, top=3)[0]
    for cl in classes:
        st.write(cl[1], cl[2])

```

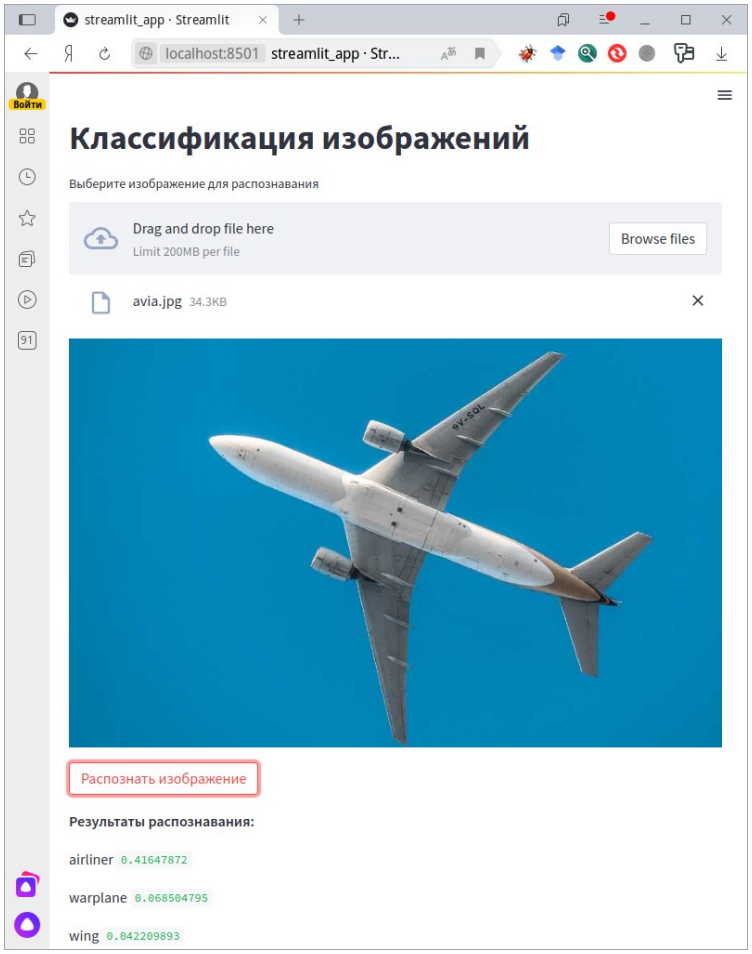
6. Было продолжено написание страницы приложения с помощью Streamlit.

```
# Выводим заголовок страницы средствами Streamlit
st.title('Классификация изображений')
# Вызываем функцию создания формы загрузки изображения
img = load_image()

result = st.button('Распознать изображение')
if result:
    x = preprocess_image(img)
    preds = model.predict(x)
    st.write('**Результаты распознавания:**')
    print_predictions(preds)
```

7. Приложение запущенное на сервере Streamlit будет выглядеть следующим образом.



8. В область выбора файла можно перетащить картинку, тогда она загрузится на сервер и будет показана на странице.



Таким образом, был написан код приложения классификации изображений. Использовалась библиотека TensorFlow. Затем скрипт был преобразован в Web-приложение с помощью библиотеки Streamlit. После чего создано приложение на облачной платформе Streamlit Cloud.

3. ОРГАНИЗАЦИЯ КОМАНДНОЙ РАБОТЫ

До начала работы над проектом была организована команда. Все участники команды работают в одном вузе (по 2 человека с двух кафедр). Несмотря на работу в одной организации, учитывая подвижный график работы, организовывать очные встречи оказалось очень проблематичным. Поэтому было принято решение использовать для командной работы сервис Битрикс24.

Сайт создан и находится по адресу https://asu-urfu.bitrix24.ru, все участники команды зарегистрированы. На этой платформе создан проект с общим диском для проектных артефактов (документов и различных материалов), кроме проектного кода. Все участники команды приглашены в проект, все участвуют в общем (проектном) чате (рис. 1).

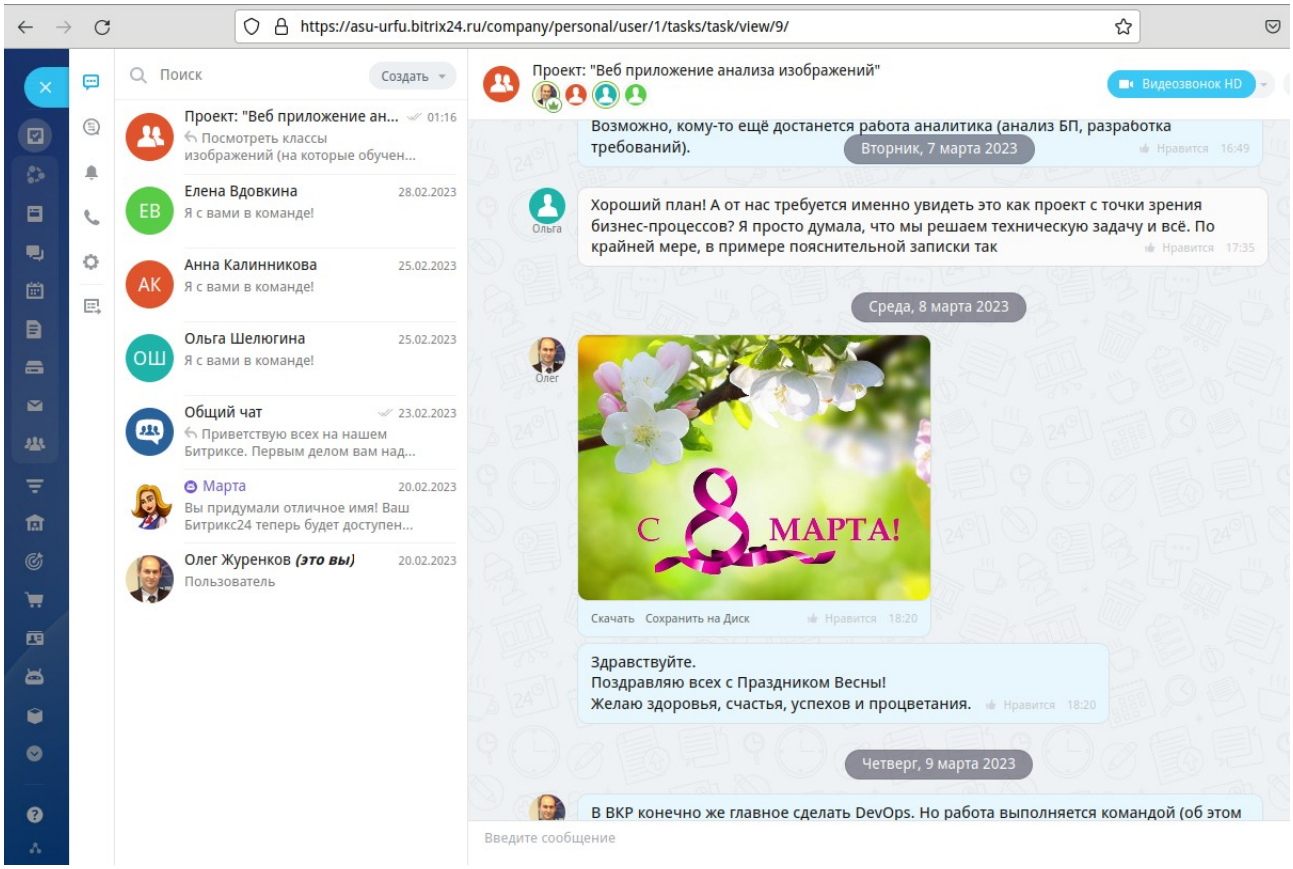


Рисунок 1: Обсуждение проекта в нашем Битрикс24

3.1. Репозиторий с кодом

Для размещения программного кода был выбран сервис GitHub, репозиторий находится по адресу https://github.com/zhurenkov/imageAnalyzer (рис. 2). Для организации командной работы в репозиторий были приглашены все члены команды. Поскольку команда небольшая (4 чел.) и на момент создания репозитория существовало чёткое разделение ответственности, было выбрано простое решение (для малых команд) — всем выданы полные права на репозиторий (рис. 3). В случае дальнейшего развития проекта и роста команды, можно создать из участников организацию с отдельными зонами доступа к артефактам проекта. Можно было бы создать и проект GitHub, однако изучив документацию, мы пришли к выводу, что организация проектной работы лучше сделана в Битрикс24, в то время, как управление версиями и непрерывная интеграция — несомненно в GitHub.

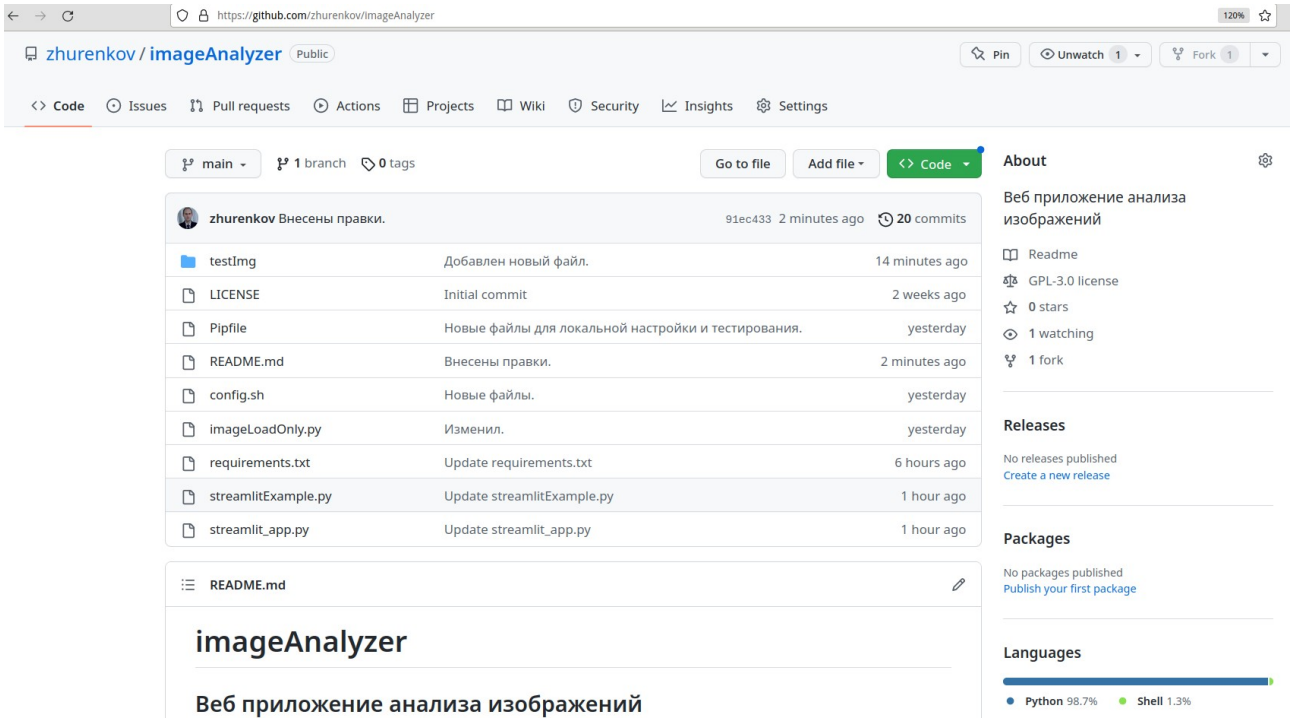


Рисунок 2: Наш репозиторий на GitHub

Основная работа проводилась на локальном компьютере, в локальном репозитории, который синхронизировался с удалённым (на GitHub). Примеры работы с репозиторием представлены на рис. 5, 6.

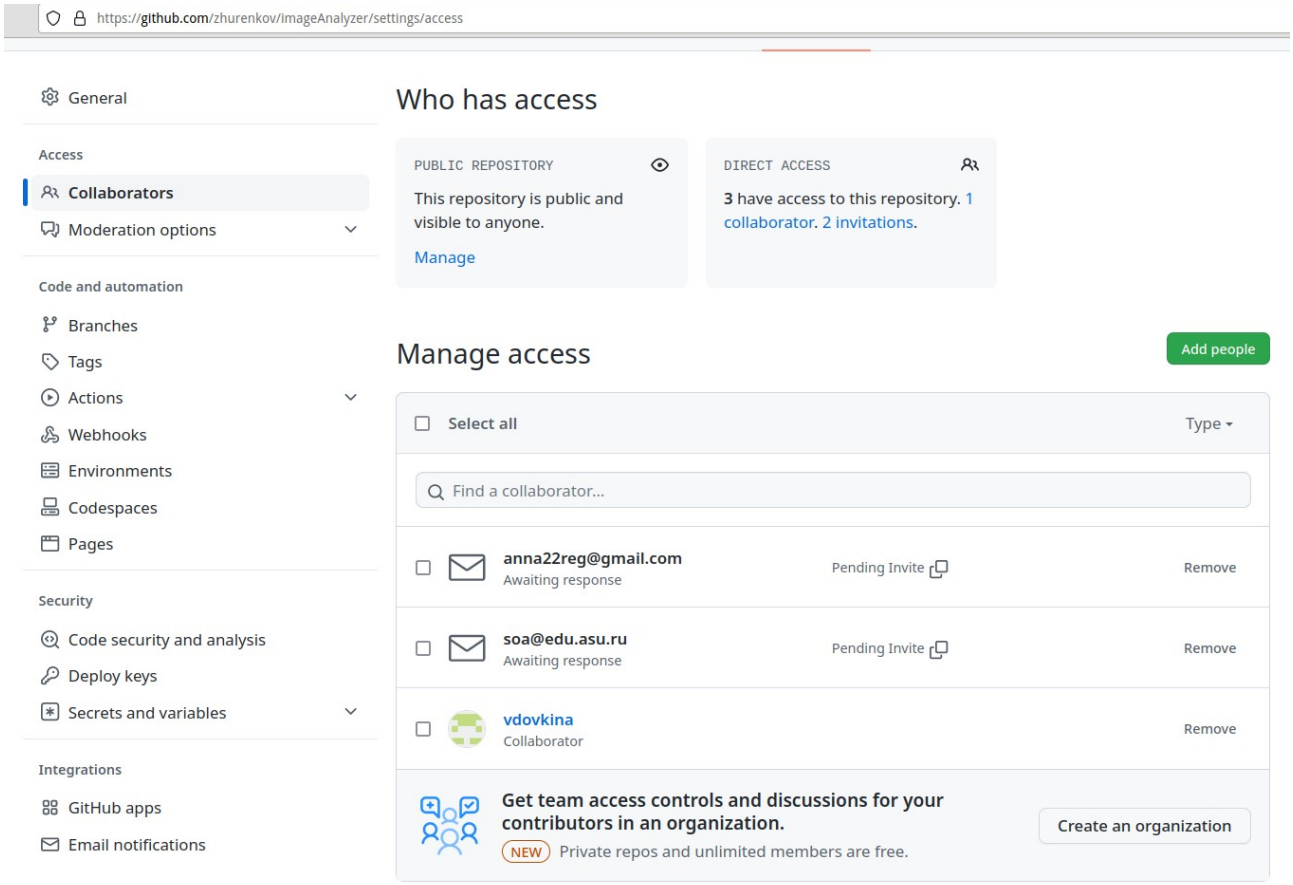


Рисунок 3: Доступ к репозиторию для всех участников проекта

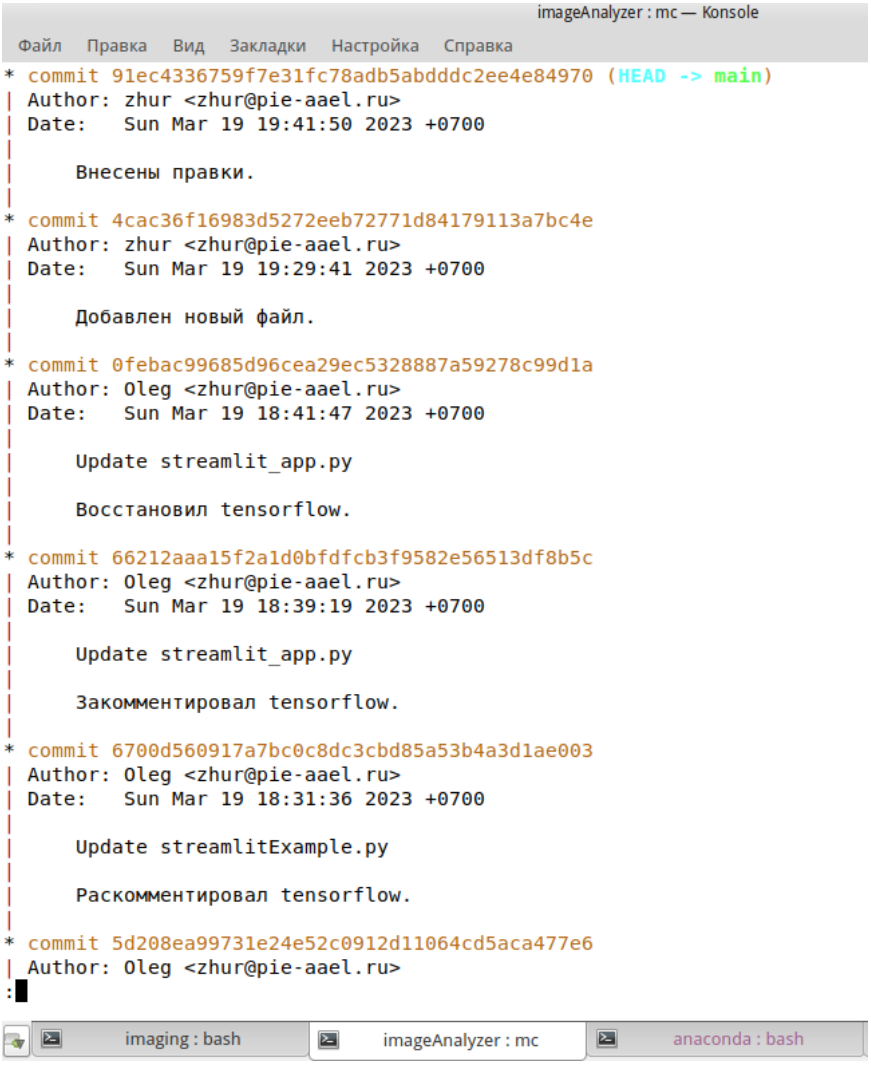


Рисунок 4: Работа с локальным репозиторием в терминале

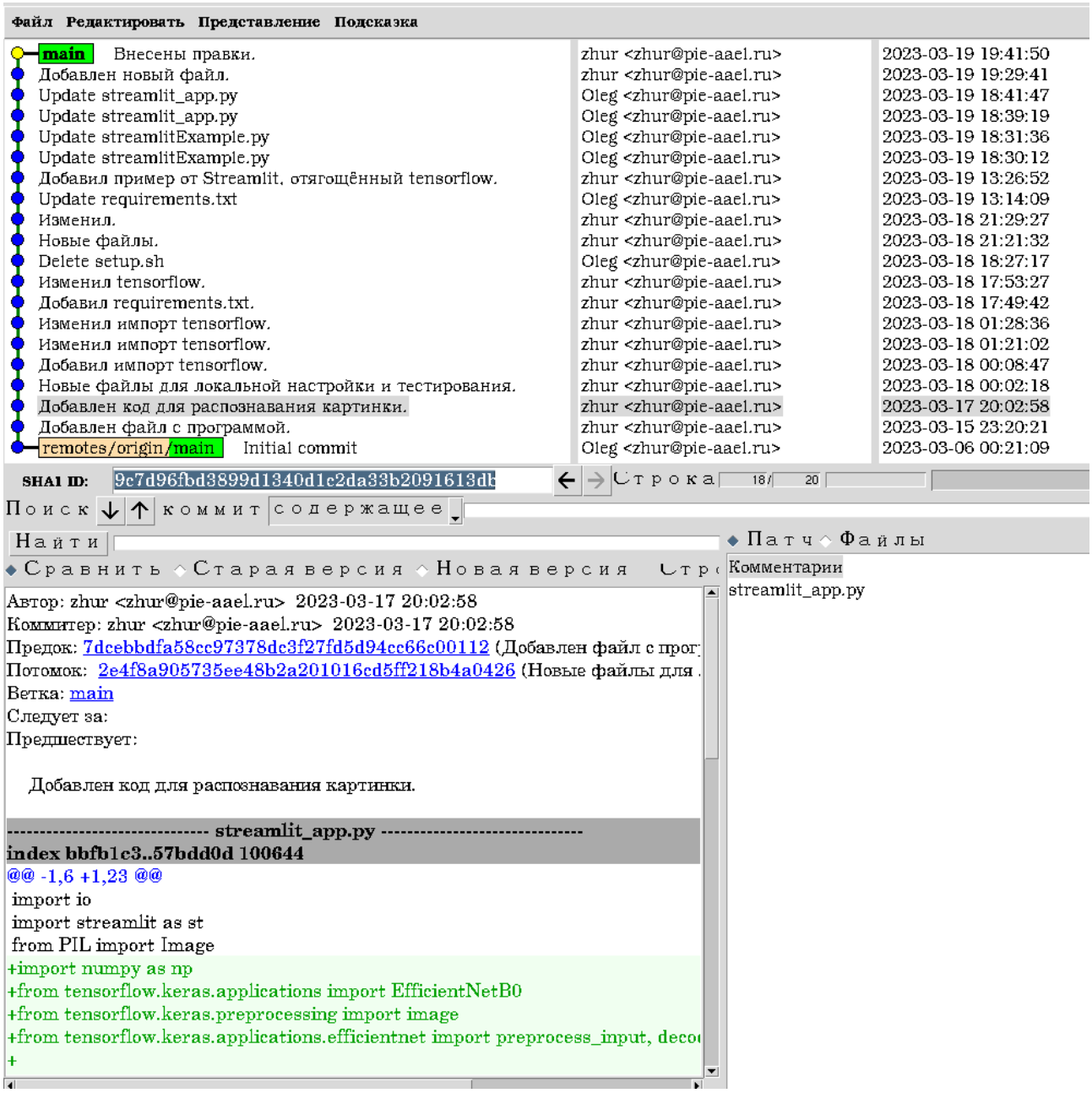


Рисунок 5: Локальный репозиторий в gitk

3.2. Тестирование системы

Разработка и тестирование системы проводились на локальном компьютере. Для организации среды выполнения скриптов Python (загрузка и классификация изображений, запуск веб-приложения, …) было принято решение установить версию Python 3.9 (среднюю для streamlit), а для этого надо

установить pyenv.

Для правильной установки pyenv потребовалось дополнительно установить ряд пакетов, согласно рекомендациям разработчиков (https://github.com/pyenv/pyenv/wiki/common-build-problems#prerequisites). Нам понадобились: zlib-devel bzip2-devel readline-devel openssl-devel tk-devel libffi-devel libuuid-devel gdbm-devel.

После этого установили python 3.9.16 и сделали его локальной версией для нашего проекта. Затем установили и протестировали библиотеку streamlit.

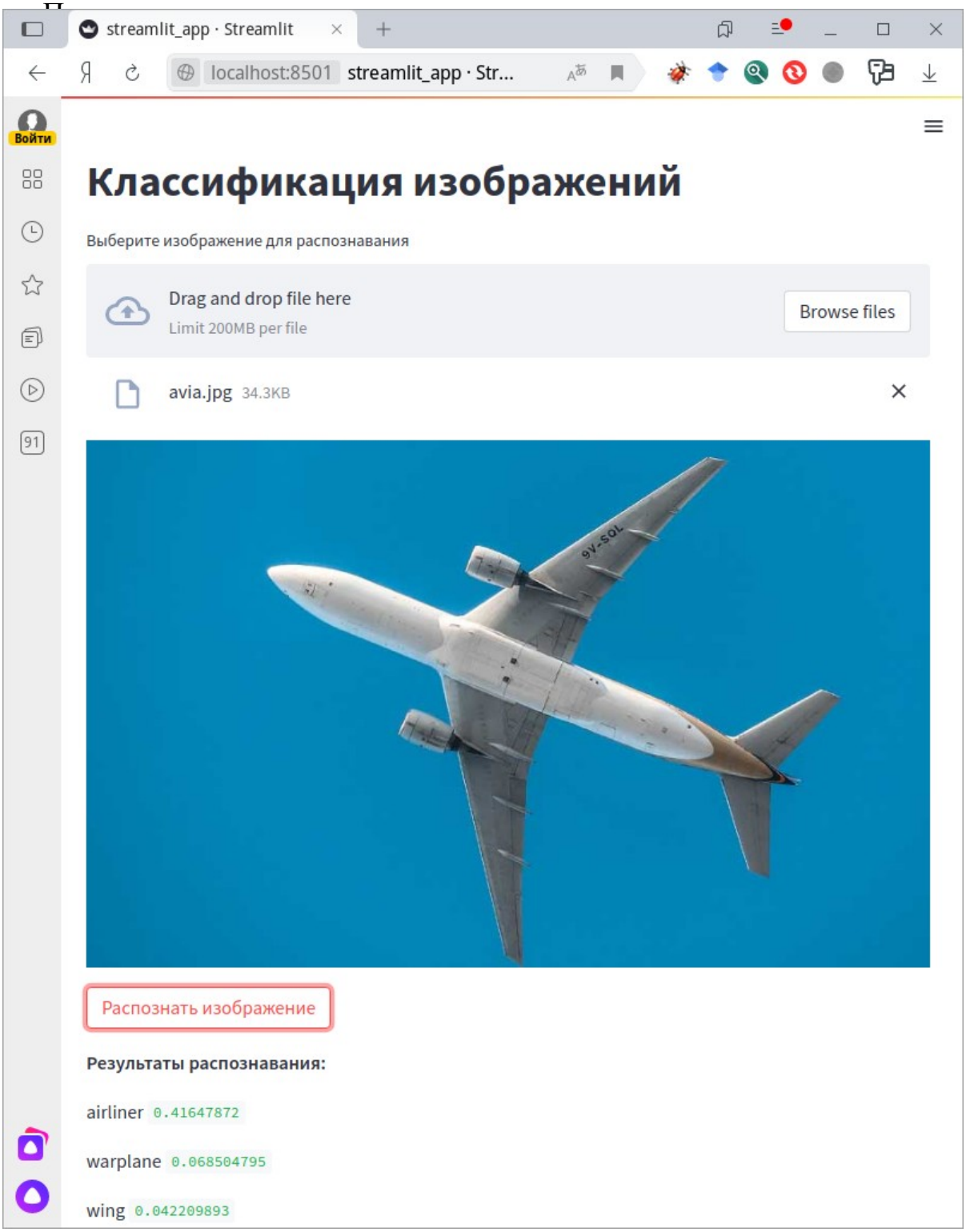


Рисунок 6: Распознавание пассажирского самолёта на чистом фоне

среду на своём сервере, доступном в Интернете, однако на нём возникли проблемы с установкой streamlit. После, якобы успешной установки, streamlit не работала. Поиск ошибки привёл к неустранимой проблеме, связанной с особенностями процессора.

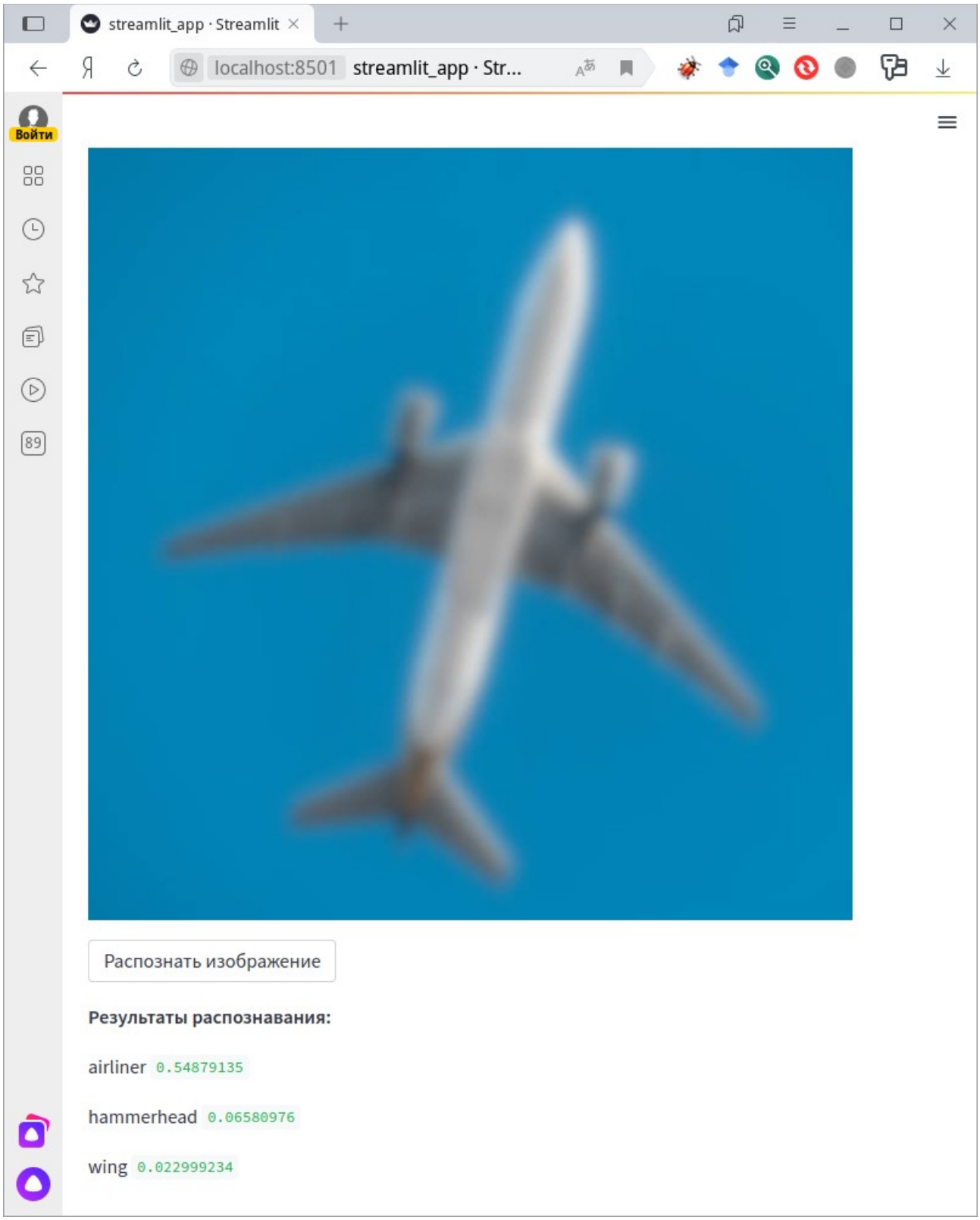


Рисунок 7: Распознавание расфокусированного пассажирского самолёта на чистом фоне

Следующим этапом выполнялась установка tensorflow в локальную среду. Локально удалось установить и протестировать, модель работает. Некоторые примеры тестирования показаны на скриншотах рис. 6–9.

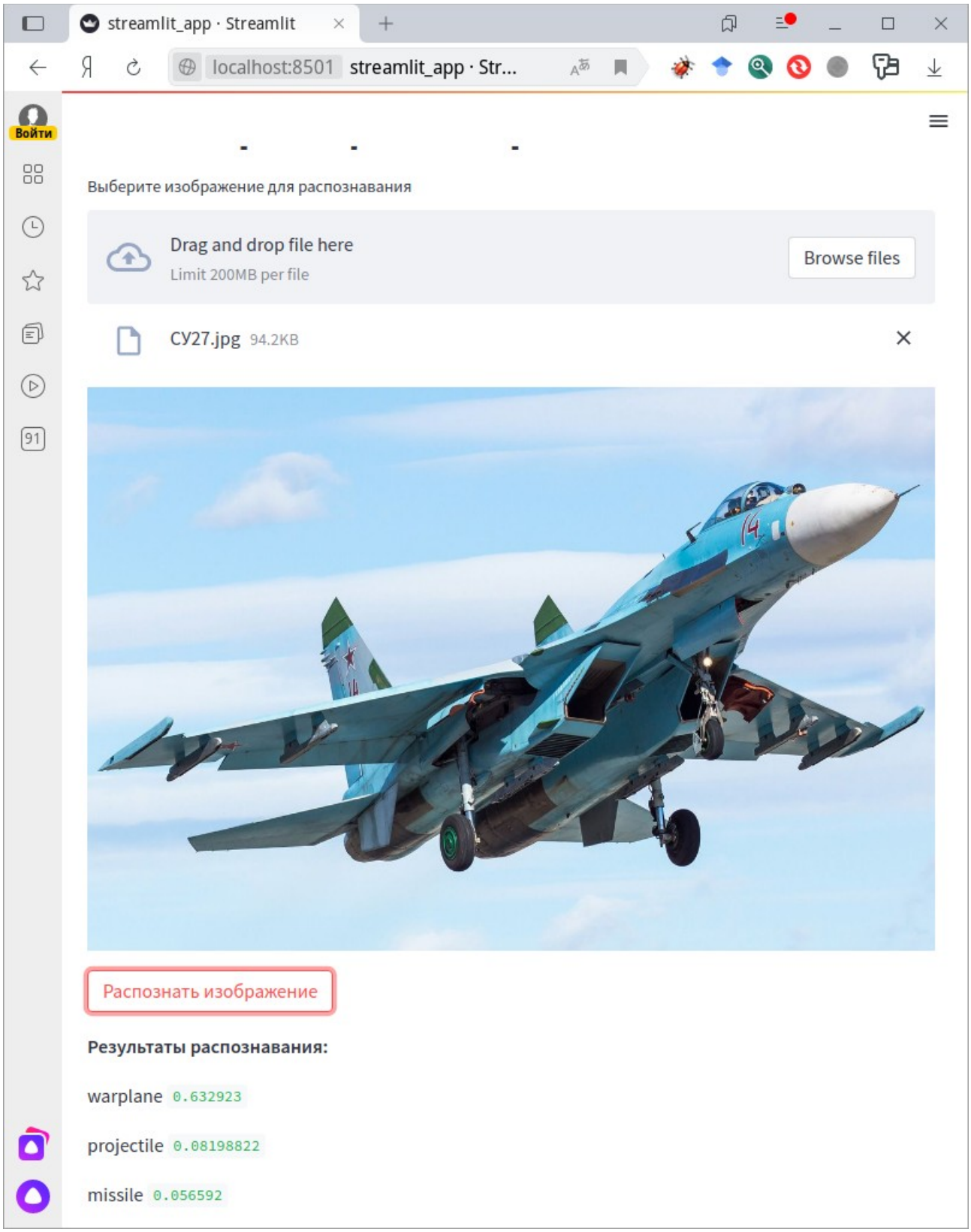


Рисунок 8: Распознавание военного самолёта

На предпоследнем примере (рис. 8) видно, что нейросеть «распознала» не только сам объект, но и его детали, благодаря которым самолёт распознан, как военный (warplane). На последнем примере (рис. 9) видно, что нейросеть выделила 3 объекта и по отдельности их «распознала».

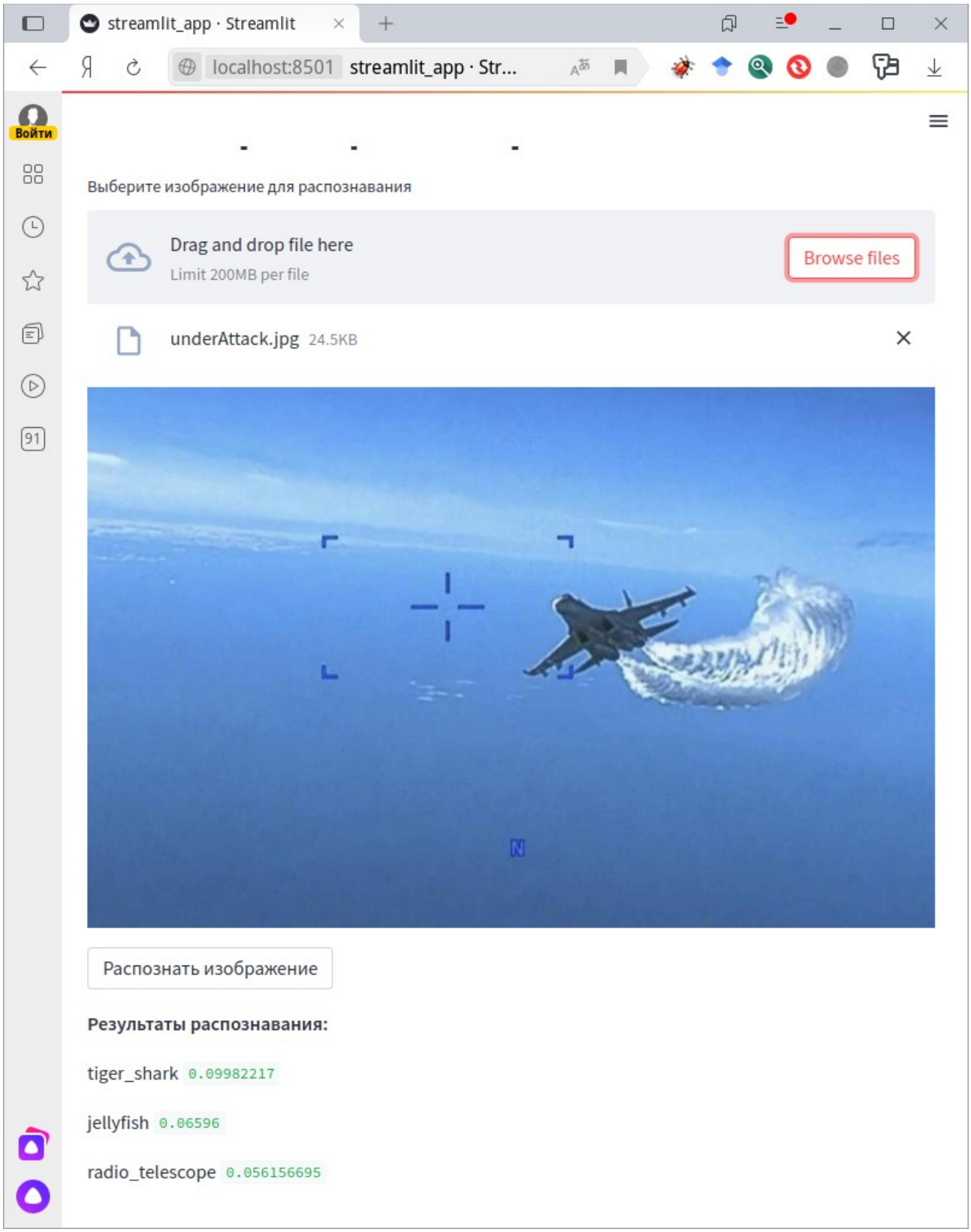


Рисунок 9: Распознавание военного самолёта, снятого камерой БПЛА

3.3. Непрерывная интеграция

Наш проект небольшой и программный код пока располагается в одном файле (не считая импортируемых библиотек и модулей). Однако, в случае роста проекта проблем со сборкой не должно быть, так как git и GitHub поддерживают CI (непрерывную интеграцию). Для развёртывания приложения используется вебхук с отправкой POST-запроса адрес https://share.streamlit.io/hook с подробной информацией о событиях git push в формате JSON. Таким образом, при любом изменении репозитория (в том числе и программного кода), эти изменения передаются в место сборки приложения, в данном случае — на платформу Streamlit (см. рис. 10).

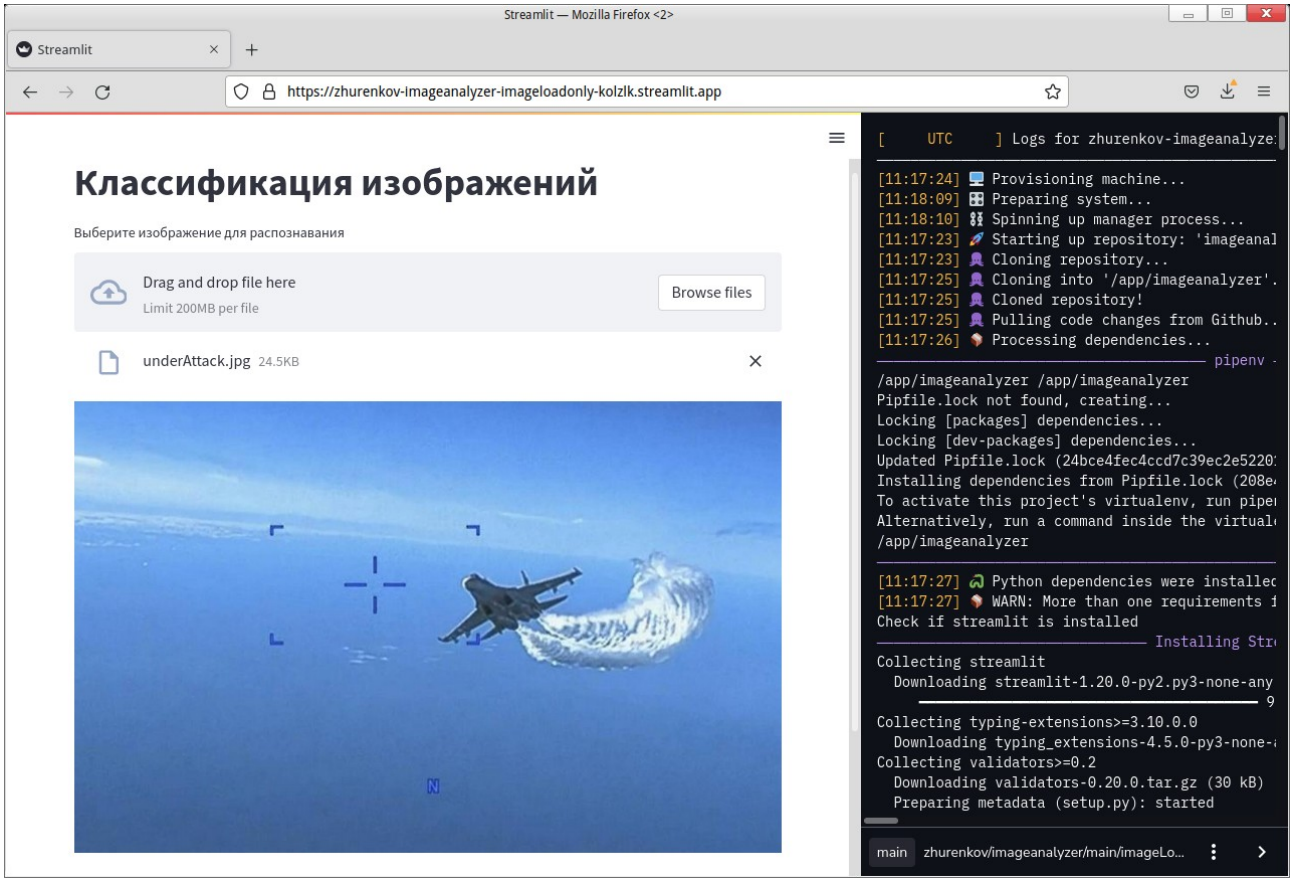


Рисунок 10: Непрерывная интеграция в Streamlit Cloud

3.4. Развертывание приложения в облаке

Для осуществления непрерывной интеграции и развёртывания в облаке была выбрана платформа Streamlit Cloud. Для этого была выполнена регистрация (с использованием аккаунта GitHub). Наше приложение доступно в облаке по адресу https://zhurenkov-imageanalyzer-streamlit-app-7fxwk0.streamlit.app/.

Однако на этой платформе с развёртыванием приложения возникли проблемы. Готовое, работающее, протестированное приложение на Streamlit Cloud вызывает ошибку ModuleNotFoundError: No module named 'tensorflow' (рис. 11). Ошибка возникает на первой строке с загрузкой модуля из tensorflow. Для выявления и устранения ошибки было проведено множество экспериментов, в том числе с тестовым примером от Streamlit. Обнаружился интересный факт. После добавления в пример от Streamlit всех строк (от своего приложения) с загрузкой из tensorflow и добавления tensorflow в requirements.txt — приложение работает. Однако тот же файл, загруженный из нашего репозитория не работает. В связи с этим, могу предположить, что для новых пользователей сервисом в локальное окружение запрещено добавлять библиотеку tensorflow. Пользователи, регистрировались ранее, получили виртуальное окружение без такого ограничения.

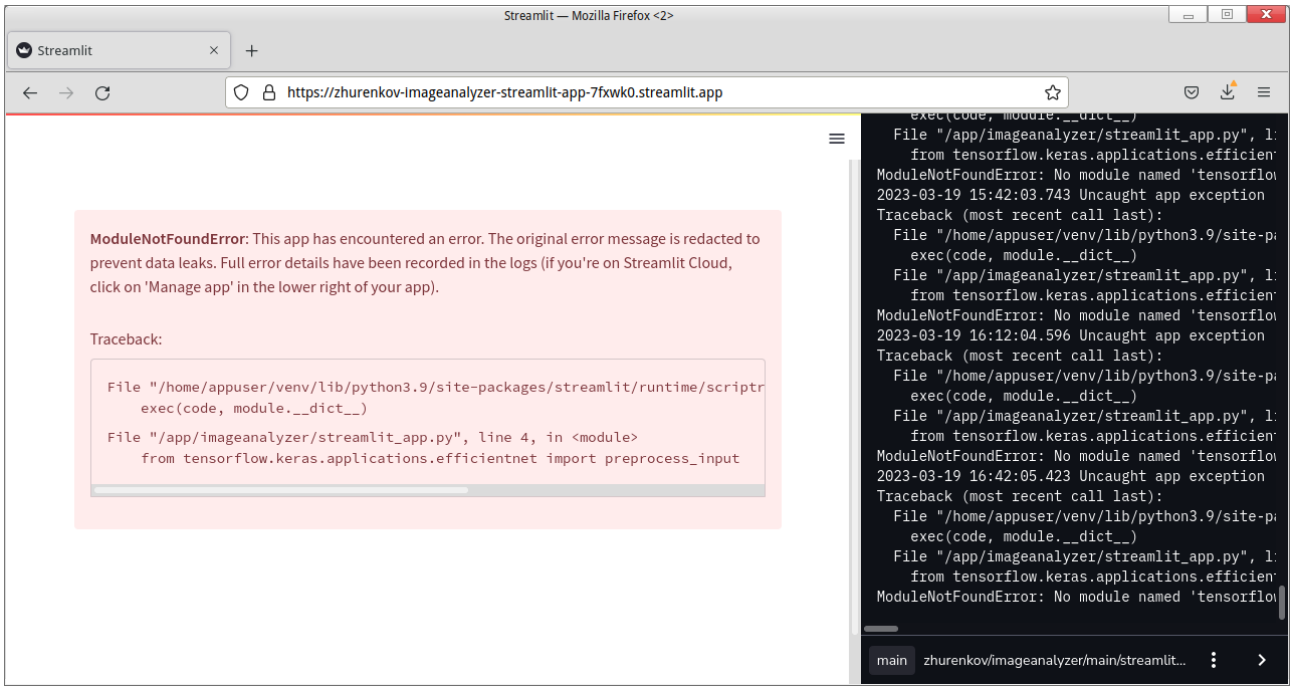


Рисунок 11: Ошибка на Streamlit Cloud

ЗАКЛЮЧЕНИЕ

Одной из базовых задач в машинном зрении является задача классификации изображения – определения категорий объектов, который находится на изображении. В данной работе была использована классификация изображений, которая представляет собой задачу присвоения входному изображению одной метки из фиксированного набора. Это одна из базовых проблем компьютерного зрения, которая, несмотря на свою простоту, имеет большое разнообразие практических применений.

Разрабатываемое приложение должно было классифицировать вводимые изображения для этого использовался EfficientNetB0, так как наиболее хорошо подходила для достижения поставленной цели и решения конкретных задач. Для реализации проекта были использованы библиотеки Streamlit, TensorFlow и утилита PyEnv.

В качестве инструмента создания приложения была использована Streamlit, библиотека Python с открытым исходным кодом, предназначенная для создания интерактивных веб-приложений и визуализации данных. Это позволяет разработчикам быстро и легко создавать веб-приложения, используя знакомый код Python.

В данной работе было реализовано приложение, которое способно распознавать и классифицировать изображение. Это могут быть изображения разных классов и тем, однако проектная идея предполагает использование приложения в образовательном процессе вуза, в процессе подготовки студентов творческих направлений — будущих дизайнеров, иллюстраторов, стилистов.

Таким образом, было создано web-приложение для аннотации изображений. Целью проекта на данном этапе является разработка инструмент для сегментации изображений, предусматривающий возможность загрузить изображение, провести сегментацию и сохранить результат на персональный компьютер. Однако для максимальной автоматизации удобно хранить уже аннотированные изображения в самом приложении. В следующих проектных итерациях предполагается реализовать автоматическую аннотацию изображений,

то есть процесс, в котором система автоматически присваивает метаданные цифровому изображению, используя соответствующие ключевые слова для описания его визуального содержания, и сохраняет их в том же приложении.

Первым этапом работы над проектом была организована команда. Все участники команды работают в одном вузе (по 2 человека с двух кафедр). Для командной работы было принято решение использовать сервис Битрикс24. Репозиторий находится по адресу https://github.com/zhurenkov/imageAnalyzer . Для организации командной работы в репозиторий были приглашены все члены команды. Поскольку команда небольшая (4 чел.) и на момент создания репозитория существовало чёткое разделение ответственности, было выбрано простое решение (для малых команд) — всем выданы полные права на репозиторий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 4 лучших предварительно подготовленных модели для классификации изображений с помощью кода Python [Электронный ресурс]. URL: https://dev-gang.ru/article/-luczshih-predvaritelno-podgotovlennyh-modeli-dlja-klassifikacii-izobrazhenii-s-pomosczu-koda-python-00gddb9o3g/?ysclid=lfdn2y85s3239946020 (дата обращения: 20.03.2023)
2. Вигерс К., Битти Дж. Разработка требований к программному обеспечению. 3-е изд., дополненнное. — СПб.: БХВ, 2022. — 736 с.
3. Дудалова, Е. А. Создание и запуск веб-приложения с помощью Streamlit / Е. А. Дудалова // Фундаментальные и прикладные аспекты развития современной науки : Сборник научных статей по материалам X Международной научно-практической конференции, Уфа, 17 января 2023 года. – Уфа: Общество с ограниченной ответственностью "Научно-издательский центр "Вестник науки", 2023. – С. 52-55. – EDN BYDDMQ.
4. Классификация изображений в Pytorch – использование предварительно обученных моделей [Электронный ресурс]. URL: https://pythonpip.ru/pytorch/klassifikatsiya-izobrazheniy-v-pytorch-ispolzovanie-predvaritelno-obuchennyh-modeley (дата обращения: 20.03.2023)
5. Калмыков, М. Д. Возможности фреймворка Streamlit для созданий и развертывания интерактивных веб-приложений / М. Д. Калмыков // Актуальные вопросы эксплуатации систем охраны и защищенных телекоммуникационных систем : Сборник материалов Всероссийской научно-практической конференции, Воронеж, 10 июня 2021 года. – Воронеж: Воронежский институт Министерства внутренних дел Российской Федерации, 2021. – С. 19-21. – EDN UMGGLT.
6. Конова П.С. Бибиблиотека Streamlit как инструмент обработки и визуализации больших данных // Столыпинский вестник. 2022. №3. URL: https://cyberleninka.ru/article/n/biblioteka-streamlit-kak-instrument-obrabotki-i-vizualizatsii-bolshih-dannyh (дата обращения: 22.03.2023).

7. Корнипаев И. Требования для программного обеспечения: рекомендации по сбору и документированию. — М.: Книга по требованию, 2013. — 118 с.
8. Обзорный анализ Python веб-фреймворков [Электронный ресурс]. URL: https://tproger.ru/articles/obzornyj-analiz-python-veb-frejmvorkov/ (дата обращения: 20.03.2023)
9. Русскоязычная документация Keras [Электронный ресурс]. URL: https://ru-keras.com/applications/?ysclid=lfdmjgtqtk162782578 (дата обращения: 20.03.2023)
10. Сквозная платформа машинного обучения с открытым исходным кодом TensorFlow [Электронный ресурс]. URL: https://www.tensorflow.org/ (дата обращения: 20.03.2023).
11. Создание интерактивных панелей с Streamlit и Python [Электронный ресурс]. – URL: https://proglib.io/p/sozdanie- interaktivnyh-paneley-s-streamlit-i-python-2021-06-21. (дата обращения: 20.03.2023).
12. Траск Эндрю. Грокаем глубокое обучение. — СПб.: Питер, 2020. — 352 с.: ил.
13. Фарли Д. Современная программная инженерия. ПО в эпоху эджайла и непрерывного развертывания. — СПб, Питер, 2023. — 288 с.
14. Харбанс Ришал. Грокаем алгоритмы искусственного интеллекта. — СПб.: Питер, 2023. — 368 с.
15. Шолле Франсуа. Глубокое обучение на Python. – Питер, 2020. – 400 с. – ISBN 978-5-4461-0770-4.
16. AI, практический курс. Обзор нейронных сетей для классификации изображений [Электронный ресурс]. URL: https://habr.com/ru/company/intel/blog/415811/ (дата обращения: 20.03.2023)
17. Going deeper with convolutions [Электронный ресурс]. URL: https://arxiv.org/pdf/1409.4842.pdf (дата обращения: 20.03.2023)
18. Mingxing Tan, Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks // International conference on machine learning, 2019 URL: https://arxiv.org/abs/1905.11946

19. Turn Python Scripts into Beautiful ML Tools [Электронный ресурс]. URL: https://towardsdatascience.com/coding-ml-tools-like-you-code-ml-models-ddba3357eace (дата обращения: 20.03.2023)
20. Treuille A. Turn Python Scripts into Beautiful ML Tools [Электронный ресурс] // Medium. - 2019. - URL: https://towardsdatascience.com/coding-ml-tools-like-you-code-ml-models-ddba3357eace

ПРИЛОЖЕНИЕ 1. КЛЮЧЕВЫЕ ФРАГМЕНТЫ КОДА ПРИЛОЖЕНИЯ

streamlit_app.py

```
import io
import streamlit as st
from PIL import Image
from tensorflow.keras.applications.efficientnet import preprocess_input
from tensorflow.keras.applications.efficientnet import decode_predictions
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import EfficientNetB0
import numpy as np


@st.cache(allow_output_mutation=True)
def load_model():
    return EfficientNetB0(weights='imagenet')


def preprocess_image(img):
    img = img.resize((224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    return x

def load_image():
    """Создание формы для загрузки изображения"""
    # Форма для загрузки изображения средствами Streamlit
    uploaded_file = st.file_uploader(
        label='Выберите изображение для распознавания')
    if uploaded_file is not None:
        # Получение загруженного изображения
        image_data = uploaded_file.getvalue()
        # Показ загруженного изображения на Web-странице средствами Streamlit
        st.image(image_data)
        # Возврат изображения в формате PIL
        return Image.open(io.BytesIO(image_data))
    else:
        return None

def print_predictions(preds):
    classes = decode_predictions(preds, top=3)[0]
    for cl in classes:
        st.write(cl[1], cl[2])


model = load_model()
```

```
# Выводим заголовок страницы средствами Streamlit
st.title('Классификация изображений')
# Вызываем функцию создания формы загрузки изображения
img = load_image()

result = st.button('Распознать изображение')
if result:
    x = preprocess_image(img)
    preds = model.predict(x)
    st.write('**Результаты распознавания:**')
    print_predictions(preds)
```

config.sh

```
pipenv --python 3.9
pyenv local 3.9.16
pipenv shell
python -V
```