МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИСТЕТ им. Р.Е. АЛЕКСЕЕВА»

(НГТУ)

Институт _____радиоэлектроники и информационных технологий_____

Направление подготовки (специальность) 09.03.01 «Информатика и вычислительная техника»_____

Направление (профиль) образовательной программы Вычислительные машины машины, комплексы, системы и сети_____

Кафедра _____«Вычислительные системы и технологии»_____

ВЫПУСНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

_____бакалавра_____
(бакалавра, магистра, специалиста)

Студента _____Ефоде Ирины Михайловны_____ Группы _____14-В-1____

На тему «Программная система создания наборов размеченных данных для обучения алгоритмов компьютерного зрения»_____

СТУДЕНТ

_____ __Ефоде И.М.__
(подпись) (фамилия, и., о.)

(дата)

РУКОВОДИТЕЛЬ

_____ ____Гай В.Е.___
(подпись) (фамилия, и., о.)

(дата)

РЕЦЕНЗЕНТ

_____ _____.___
(подпись) (фамилия, и., о.)

(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ

_____ Кондратьев В.В
(подпись) (фамилия, и., о.)

(дата)

КОНСУЛЬТАНТЫ:

1. По _____

_____ _____
(подпись) (фамилия, и., о.)

(дата)

2. По _____

_____ _____
(подпись) (фамилия, и., о.)

(дата)

ВКР защищена ____(дата)

Протокол №_____

С оценкой _____

Оглавление

Введение

Практически каждый человек в современном мире слышал термин «Большие данные» (англ. «Big data»), но все ли действительно понимают сущность данного явления? Применяют ли данное определение правильно? Оценивают ли корректно объем информации, собирающейся из дня в день?

Сегодня под термином «Большие данные» программисты и аналитики понимают огромное количество многообразных структурированных и неструктурированных данных, обрабатываемых программным и аппаратным инструментарием. Но экономисты и социологи подразумевают под данным определением социально-экономический феномен, связанный с появлением технологических возможностей анализа больших массивов разнообразных данных и, как следствие, их трансформации. В научной литературе большие данные характеризуются так называемыми 3V – объем (англ. «Volume»), скорость (англ. «Velocity») и многообразие (англ. «Variety»). Многие современные источники добавляют к этому жизнеспособность (англ. «Viability»), ценность (англ. «Value»), достоверность (англ. «Veracity»), переменчивость (англ. «Variability») и визуализацию (англ. «Visualization»). Но не смотря на все это, неизменными и максимально употребляемыми остается именно правило 3V.

Впервые термин «Большие данные» был применен редактором журнала Nature Клиффордом Линчем 3 сентября 2008 года в статье «Как могут повлиять на будущее науки технологии, открывающие возможности работы с большими объемами данных?» к феномену взрывного роста объемов и разнообразия

поступающих на обработку данных и технологических перспективах, открывающихся в связи с появлением данного явления. Спустя всего год данное название стало широко употребляемым в прессе, а немного позже ведущие компании-разработчики, такие как Microsoft, Google, Intel, Dell, HP, IBM, Amazon и другие стали применять это определение и к своим решениям, как аппаратным, так и программным.

Действительно, если изучить статистику поступающих на хранение данных, то любой человек, хоть немного привлеченный к математическим наукам, отследит экспоненциальную зависимость количества хранимых данных от течения времени. Причем, стоит отметить тенденцию к сильному увеличению поступления именно цифровых, а не аналоговых данных, поступающих с различных источников.

Также в это время сильный толчок к развитию и широкому использованию отрасли получили такие отрасли информационных технологий как аналитика данных (англ. «Data science»), искусственный интеллект (англ. «Artificial Intelligence») (а именно искусственные нейронные сети (англ. «Artificial neural networks») и компьютерное зрение (англ. «Computer vision»)), а также имитационное моделирование и многие другие.

Сегодня основными источниками самых разнообразных данных, стекающихся в огромные центры хранения данных самых крупных игроков на рынке IT-отрасли, являются интернет вещей, телеметрия, социальные сети и медиа, а также сбор статистической информации. Почти в ста процентах случаев «сырые» данные являются неструктурированными или плохо структурированными, а значит, как следствие, требуют обработки. Именно для решения данной задачи и нужна разметка, а именно ручная разметка данных с целью последующей автоматизации структуризации огромных массивов данных с помощью алгоритмов

машинного обучения, а зачастую и компьютерного зрения. В данной статье рассматривается именно второй случай, то есть разметка потоков изображений и видео с целью последующего распознавания образов на поступающих потоках алгоритмами компьютерного зрения.

Огромные компании уже создали или еще только проектируют свои кластеры для хранения данных, разработано множество алгоритмов машинного обучения и компьютерного зрения, но головной болью для разработчиков по-прежнему является создание наборов размеченных данных, способных обучить систему распознавать и классифицировать данные по заранее определенному набору признаков, поэтому тема разметки «сырых» данных так интересна для рассмотрения.

1. Техническое задание

1.1. Назначение разработки и область применения

В последнее время транснациональные корпорации такие как Яндекс (Яндекс Toloca, Data Factory и другие), Microsoft (Microsoft Azure), Google (Google Cloud), Intel (Go SDK) и многие другие, заинтересованы в создании целых программных комплексов, включающих в себя полный набор компонентов для алгоритмов машинного обучения и компьютерного зрения – от сбора данных и преобразования их в форматы, доступные для разметки, и до программ, позволяющих распознавать данные с помощью уже обученного алгоритма.

Прежде всего для любого алгоритма необходим некоторый набор входных данных. В случае алгоритмов машинного обучения с учителем, а чаще всего применяется именно этот случай, нам необходимы данные, из которых нужно выделить необходимую информацию, а также, собственно ответы - то есть уже размеченные данные, которые помогают установить коэффициенты связи между нейронами в искусственной нейронной сети, то есть собственно структурировать и классифицировать данные.

Вообще говоря, можно выделить несколько этапов подготовки данных для алгоритмов машинного обучения:

- Сбор данных – автоматизированное коллекционирование «сырых» данных, полученных с различных источников (например, изображения с камер; набор трехмерных точек, характеризующих окружающее пространство, с внешних

сенсоров; данные о погоде и так далее). Кроме реального сбора данных для алгоритмов машинного обучения также можно воспользоваться готовыми дата-сетами, собрать данные из сети Интернет, а также симулировать данные при помощи специализированного программного обеспечения, которые помогают создавать данные полностью идентичные тем, которые могут быть получены в реальных условиях;

- Разметка данных – классификация данных, полученных на этапе сбора;
- Доступ к данным и их хранение – аккумулирование размеченных данных в дата-центрах с использованием специализированных баз данных в виде пар типа – исходное изображение в формате JPEG размеченные данные в специализированном формате типа JSON, XML и другие;
- Подготовка данных для обучения конкретного алгоритма – накопление, просмотр размеченных данных с целью корректировки, а также их отбор для алгоритмов (например, чтобы избежать переобучение алгоритма), а также адаптация данных для конкретного алгоритма;
- Визуализация данных – отображение вручную (но может быть и автоматически) размеченных данных с использованием специального программного обесчечения.

Отсюда можно сделать вывод, что для автоматизации обработки больших данных прежде всего необходимо разметить данные для алгоритмов. Разрабатываемая система позволит размечать некоторые входные данные (а именно изображения) с целью последующего обучения алгоритмов, конечной целью которого является автоматическое детектирование алгоритмом объектов, типы и атрибуты которых и будут определены на этапе разметки, на изображениях

поступающих на обработку.

То есть в ходе разработки программной системы, обеспечивающую разметку изображений, нужно:

- Проработать системные требования;
- Выбрать инструменты, наиболее подходящие для девелопмента программной системы, позволяющей осуществлять разметку данных;
- определить архитектуру приложения (а именно компоненты и их взаимодействие);
- определить типы файлов, поступающих на вход приложения в качестве данных для разметки;
- Формат хранения размеченных данных;
- способ их хранения;
- внутреннее представление;
- типы и атрибуты размечаемых объектов;
- детально проработать UI и UX-дизайн разрабатываемого программного обеспечения для того, чтобы разметчику было удобно использовать данное приложение;
- протестировать его на некоторых тестовых данных.

Таким образом, для решения задачи нужно детально проработать решение проблем, которые встречаются при разработке программных систем.

1.2. Технические требования

Разрабатываемое программное обеспечение должно соответствовать следующим требованиям:

В пояснительной записке должны быть:

1. Приложение должно работать под управлением заранее определенных операционных систем (Windows10, Ubuntu16.04);
2. В качестве входных данных программная система получает заранее определенный набор файлов поддерживаемого расширения (.jpeg, .png) и выводит данный файл в графическом интерфейсе программной системы в пригодном для разметки виде (изображение);
3. Пользователь программной системы является разметчиком данных, поступающих на вход программы, а следовательно, полностью определяет содержимое выходного файла, обеспечивая достаточную точность разметки и корректность введенных им в интерфейсе инструмента значений;
4. Выходными данными работы системы является файл, содержащий в себе информацию о размеченных объектах и их атрибутах, а также абсолютный путь к размечаемому изображению в данной файловой системе как атрибут в выходном файле заранее определенного формата (.json).

2. Анализ технического задания

2.1. Выбор операционной системы

Для определения необходимого скопа работ кроме определения требований нужно определить операционную системы, для которой будет реализован данный проект. Наиболее популярными сейчас являются Windows10, Ubuntu16.04, Ubuntu14.04, Ubuntu17.04, Mint, Fedora, Debian, Android, OS X (Mac OS). Рассмотрим подробнее факторы, которые могут повлиять на выбор операционной системы:

- Стабильность работы системы (по количеству отказов системы).

  По собственному опыту разработки и отзывам множества других разработчиков UNIX-подобные системы являются более стабильными в использовании (гораздо реже появляются системные ошибки, а так же случается полный отказ системы (так называемые «Экраны смерти» и «Crashes»)). Это в первую очередь связано с архитектурой построения операционной системы. Linux с самого начала разработки являлась модульной, что значит, что каждая компонента несет ответственность лишь за предоставляемую функциональность;

- Поддержка языка программирования

  Windows-подобные системы не имеют встроенной поддержки языков программирования, в отличие, например, от OS X или Linux-подобных систем, то есть при использовании Windows пользователь должен найти и установить специализированное ПО – компиляторы, отладчики и т.д. В Linux же достаточно выполнить команду, которая скачает и установит нужные пакеты, а

чаще всего в стандартной сборке уже предоставлено такое ПО (например, компиляторы GCC, отладчики CDB, интерпретаторы Python и т. д.)

- Стоимость

  Некоторые Linux-подобные системы (напрмер, Mint, Ubuntu, Fedora) являются свободно распространяемыми с открытым исходным кодом, в отличие от той же Windows, за лицензию для которой нужно платить.

- Безопасность

  Так как некоторые Linux-подобные системы являются системами с открытым исходным кодом, то и некоторые возникающие проблемы чинятся пользователями, в другом же случае пользователи отправляют дефекты разработчикам, после чего они приоритизируются и чинятся инженерными командами.

- Доступность

  Согласно последней статистике, операционной системой-лидером по распространению в мире является Android, второе сето отдано Windows, третье – iOs, пятое занимает OS X, Linux же лишь шестое место. Среди распространения Linux-дистрибутивов: первое место занимает Debian, второе – CentOS, а Ubuntu лишь третье. Одноко при рассмотрении операционных систем в данном контексте стоит учесть и стоимость лицензии для использования данной операционной системы.

- Точность ввода данных

  Такие системы как Android и iOs практически в 100% случаев устанавливается на устройства, поддерживающими только сенсорный ввод со стороны пользователя

Из приведенных систем сразу можно отбросить платные операционные системы, использующих Linux ядро (Debian), Android, который устанавливается в

большей части на мобильных устройствах с сенсорным управлением (точность разметки данных в таком случае снижается, так как касание человеческого пальца на области небольшого экрана мобильного устройства даст очень большую погрешность, что является критичным для данной системы), а также OS X, которая устанавливается поставщиком лишь на устройствах Macintosh. Так же можно отбросить довольно старые системы, поддержка которых скоро заканчивается, такие как, например Ubuntu14. Из оставшихся Ubuntu16.04 была выбрана как платформа для разработки программной системы, так как данная система имеет встроенный отладчик, компилятор, встроенную систему контроля версий, бесплатна и стабильна, а кроме этого в стандартную поставку клиентской версии Ubuntu входит и графическая оболочка, что ускорит разработку программной системы. Платформой выполнения программы выбраны Ubuntu16.04 (так как является платформой разработки) и Windows10 (является максимально распространенной среди операционных систем для настольных компьютеров).

2.2. Выбор языка программирования

В данной работе рассматривается десктопное приложение, ориентированное на разметку пользователем поступающих на вход данных. Одним из главных требований к данной программной системе является наличие интуитивно понятного и удобного графического интерфейса и его интерактивный ответ на любое действие пользователя. Кроме этого, хотелось бы иметь набор библиотек для работы с данными для удобства программиста. Рассмотрим языки программирования, наиболее подходящие для разбработки бек-энда данного приложения:

- C++11 + Qt Framework. Это один из самых популярных языков программирования, по мнению популярного сайта stackoverflow, C++ занимает седьмое место по популярности использования. Область его применения включает со-

здание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также развлекательных приложений (игр). Для данного языка был создан фремворк Qt, который включает множество библиотек, кроме этого созданная на основе этого фреймворка интегрированная система разработки поддерживает компилятор GCC и отладчик CDB, включенные в стандартный пакет поставки операционной системы Ubuntu. Главным минусов чистого C++ является системозависимость.

- Java - сильно типизированный объектно-ориентированный язык программирования. Приложения, написанные на Java обычно транслируются в специальный байт-код, поэтому они могут работать поверх любой компьютерной архитектуры и работающей на ней операционной системы, с помощью виртуальной Java-машины. Один из недостатков – невысокая скорость работы.
- Python - высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен, но стандартная библиотека включает большой объём полезных функций. Чаще всего этот язык программирования используется в сфере аналитики данных, сложных метематических вычислений и DevOps-практике для обработки конкретных сценариев работы. Python имеет встоенную поддержку json-формата, а код, написанный на данном языке программирования, является кроссплатформенным. Один из недостатков – невысокая скорость работы.

А также обратимся к скриптовым языкам, позволяющим описать графический интерфейс, используя подход декларативного программирования:

- QML – декларативный язык программирования, основанный на JavaScript, предназначенный для дизайна приложений, делающих основной упор на

пользовательский интерфейс. Является частью Qt Quick, среды разработки пользовательского интерфейса, распространяемой вместе с Qt, поэтому широко распространен в приложениях, разработанных с использованием фреймворка Qt.

- JavaScript - JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности вебстраницам, также часто применяется и для быстрого девелопмента серверной части Web-приложений с использованием фремворка Node.JS

После изучения недостатков и преимуществ всех рассмотренных языков в качестве языка разработки бек-энда был выбран C+, а для написания фронтенда – QML, так как выбранная комбинация ускоряет разработку программной системы с использованием кроссплатформенного фреймворка Qt. Данные языки являются необходимыми средствами создания программной системы, однако не могут представлять собой полный набор инструментов для реализации задуманной идеи, в связи с чем рассмотрим дополнительные программные средства, предназначенные для разработки программного обеспечения.

2.3. Выбор системы автоматизации сборки проекта

Система автоматизации сборки программного обеспечения из исходного кода позволяет автоматизировать сборку всего исходного кода в единый объектный файл, с целью последующего получения исполнимого файла.
Среди систем автоматизации сборки рассмотрим:

- Cmake - кроссплатформенная система автоматизации сборки программного обеспечения из исходного кода. CMake генерирует файлы управления сборкой из файлов CMakeLists.txt.

- QMake - утилита из состава Qt, которая помогает облегчить процесс сборки приложения на разных платформах. Qmake автоматически генерирует make-файлы, основываясь на информации в файлах проекта(*.pro). Главным преимуществом является непосредственная интеграция с Qt, а вот из минусов стоит отметить сложность написания и слабую конфигуряемость проекта.

После оценки всех преимуществ и недостатков рассмотренных систем, решено было использовать CMake, так как она предоставляет широкие возможности работы с проектом, а также позволяет полностью сконфигурировать состав проекта вплоть до включения/отключения функциональности и тестов, что очень удобно.

2.4. Выбор системы контроля версий

Система контроля версий - программное обеспечение для облегчения работы с быстро изменяющимся в процессе разработки исходным кодом. Система управления версиями позволяет хранить несколько версий – коммитов и ветвей, одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

В данном проекте это позволит не только сохранять исходный код с историей разработки, но и в случае ошибок откатывать систему до некоторого стабильного состояния.

Рассмотрим некотрые примеры:

- Git - распределённая система управления версиями, разработанная Линусом Торвальдсом (создателем Линукс). Git хранит лишь разности между файлами на различных ветках и между каждыми коммитами и файлы конфигурации, поэтому работает очень быстро, а при хранении репозиторий занимает немного места. Отлично умеет работать с удаленными репозиториями. Его

синтаксис прост и минималистичен, поэтому git является одной из самых популярных систем контроля версий.

- Mercurial - кроссплатформенная распределённая система управления версиями, разработанная для эффективной работы с очень большими репозиториями кода. В отличие от git является консольной программой.
- SVN - свободная централизованная система управления версиями, официально выпущенная в 2004 году компанией CollabNet.В отличие приведенных ранее систем контроля версий git и mercurial ориентирована на хранение файлы, а не их изменения.

Взвесив «плюсы» и «минусы» используемых в качестве примера систем контроля версий, мной было принято решение использовать git, так как его командный язык прост и понятен, количество дополнительно хранящейся информации минимально, а работа с удаленным репозиторием удобна, проста и не доставит дополнительных трудностей при реализации проекта.

2.5. Хостинги для хранения репозиториев

.

Про локальные репозитории и систему контроля версий для них мы оговорили ранее. Что же касается удаленных репозиториев, то для их хранения создано огромное количество веб-сервисов для их хостинга. Рассмотрим примеры:

- GitHub – крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Основан на системе контроля версий Git, а разработан на Ruby. Сервис абсолютно бесплатен для проектов с открытым исходным кодом и предоставляет им все возможности, а для частных проектов установлены различные платные тарифные планы. GitHub ориентирован на область со-

циализации программирования, то есть позволяет разрабочикам общаться, делать ревью и так далее.

- GitLab - веб-приложение для хостинга исходного кода проектов, основанное на системе контроля версий Git. Своим функционалом GitLab очень напоминает GitHub, однако заточен под командную работу, в то время как GitHub отдает предпочтение индивидуальной работе.

- BitBucket - веб-сервис для хостинга проектов и их совместной разработки, основанный на системе контроля версий Mercurial и Git. По назначению и основным предлагаемым функциям аналогичен GitHub, от которого отличается с меньшей пользовательской базой и возможностью их бесплатного хостинга приватных репозиториев с ограничением на размер команды не более пяти человек и меньшая арендная плата при большем размере команды, а также управлением правами доступа на уровне отдельных ветвей проекта. (англ. social coding), Bitbucket ориентирован на небольшие закрытые команды разработчиков. Слоган сервиса — Bitbucket is the Git solution for professional teams («Bitbucket — это Git-решение для профессиональных команд»).

После оценки нескольких веб-сервисов для хостинга, таковым для проекта был выбран GitHub, так как данный сервис предоставляет широкие возможности для разработки программного обеспечения, бесплатен и наиболее широко известен.

2.6. Выбор компилятора

Сегодня существует огромное количество компиляторов для языка C++, как свободно распространяемых, так и имеющих платную лицензию на использование. Они поддерживают работу под управлением различных операционных систем и интегрированных систем разработки программного обеспечения, однако,

как и раньше, главной функцией компилятора является перевод программы из человеко-читаемого вида в машинные коды, а также оптимизация написанных программ. Среди примеров рассмотрим:

- Intel C++ Compiler - оптимизирующий компилятор, разрабатываемый фирмой Intel для процессоров семейств x86, x86-64 и IA-64. Основным достоинством компилятора являются выполняемые им высокоуровневые, а также целевые оптимизации под процессоры Intel. Компилятор работает под ОС Linux, Windows, Mac OS X. Бесплатен для некоммерческого использования.
- Microsoft Visual Studio Compiler – компилятор, разработанный компанией Microsoft, автоматически встраивается в Microsoft Visual Studio, а также поддерживает .NET Framework. Бесплатен для некоммерческого использования.
- MinGW compiler - компилятор, GNU Compiler Collection (GCC) под Windows вместе с набором свободно распространяемых библиотек импорта и заголовочных файлов для Windows API. MinGW позволяет разработчикам создавать родные (native) приложения Windows. В MinGW включены расширения для библиотеки времени выполнения Microsoft Visual C++ для поддержки функциональности C99.
- GCC - набор компиляторов для различных языков программирования, разработанный в рамках проекта GNU. GCC является свободным программным обеспечением, распространяется фондом свободного программного обеспечения (FSF) на условиях GNU GPL и GNU LGPL и является ключевым компонентом GNU toolchain. Он используется как стандартный компилятор для свободных UNIX-подобных операционных систем. Важной особенностью данного компилятора является поддержка стандартов C++, в отличие от остальных компиляторов.

Для разработки программной системы мной был сделан выбор в пользу компиляторов GCC и MSVS Compiler, так как они поддерживают современные стандарты C++, а также широко распространены, кроме этого для моего проекта не нужны оптимизация и высокая скорость выполнения, предоставляемые Intel C++ Compiler.

2.7. Выбор среды разработки

Практически любой программист может заниматься разработкой программного обеспечения, используя текстовые редакторы и терминал, в который встроены компилятор и редактор связей. Однако на рынке сегодня представлено огромное количество Интегрированных систем разработки программного обеспечения, дающих колоссальные возможности для инжинеринга и кодинга.
Мне, как разработчику программного обеспечения, необходимо использование такой функциональности как :

- Подсветка синтаксиса
- Удобные комбинации клавиш для быстрого девелопмента
- Читабельный и интерактивный отладчик
- Возможности автодополнения
- Понятный и удобный интерфейс
- Функциональность перехода между файлами, функциями и переменными
- Встроенная возможность работы с системой контроля версий тоже будет дополнительным плюсом

Рассмотрим те IDE, кото рые могут быть использованы для реализации задумки:

- Microsoft Visual Studio 2017- линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментальных средств. Данные продукты позволяют разрабатывать как консольные приложения, так и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms.

- Qt Creator - кроссплатформенная свободно распространяемая интегрированная среда разработки для девелопмента на C, C++ и QML, а также вставками на других языках. Разработана Trolltech (Digia) для работы с фреймворком Qt. Включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса как с использованием QtWidgets, так и QML. Поддерживаемые компиляторы: GCC, Clang, MinGW, MSVC, Linux ICC, GCCE, RVCT, WINSCW.

- CLion - интегрированная среда разработки для языка программирования «C++». Подходит для операционных систем «Windows», «macOS», и «Linux», однако не является свободно распространяемой.

Для разработки программной системы максимально подходит QtCreator, так как данная система обладает достаточной гибкостью в работе с компиляторы, сторонними системами автоматизации сборки, поддерживает графический интерфейса, а кроме этого имеет встроенную поддержку фреймворка Qt.

2.8. Обзор инструментов для разметки данных

В своей основе, все приложения для разметки имеют схожий функционал и одну главную цель – разметить данные. Однако разметка может быть осуществлена самыми разными способами, а в приложениях для разметки может поддерживаться различный функционал - от банальной двухмерной разметки видеостримов и изображений, до лейблинга звуковых дорожек и трехмерных объектов.

Рассмотрим уже существующие решения как от ведущих компаний в индустрии, так и до решений, которые были разработаны маленькими командами для решения конкретных задач. Для получения информации обратимся к порталу Хабрахабр, который указан в списку литературы под номером 4.

2.8.1. Alp's Labeling Tools for Deep Learning

Так называется одно из приложений для рабочих станций, предназначенное для разметки набора изображений в Kitti-формате (то есть это есть набор изображений и текстовый файл, содержащий таймстемпы для них). По сути своей данное приложение представляет собой набор плагинов вокруг ядра программной системы – Fiji (ImageJ). Работает под управлением операционных систем Windows и Ubuntu, а также не требует обязательной установки.

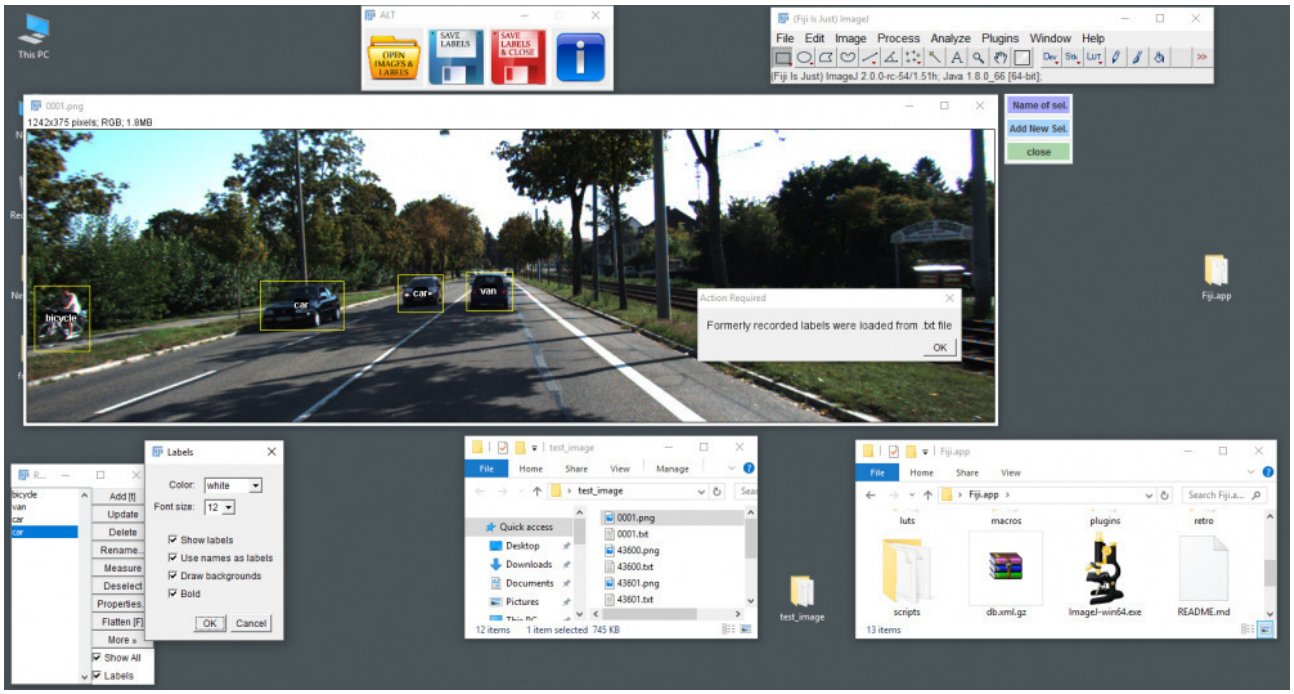

Рисунок 2.1. Скриншот программной системы "Alp's Labeling Tools for Deep Learning"

Разметка изображений осуществляется рисованием bounding boxes поверх изображения и заполнением атрибутов для выделенного объекта. Данный инструмент является свободно распространяемым. Однако одним его из главных

минусов, в принципе как и плюсов, является дополнительная установка множества плагинов к ядру системы. Отрицательная сторона этого явления состоит в том, что пользователю нужно кроме основного дистрибутива скачать еще множество побочных, без которых не поддерживается основная функциональность, а также разобраться с добавлением плагинов непосредственно в сам инструмент. Положительная сторона – это то, что пользователь сам полностью может сконфигурировать приложение. Также следует отметить, что Alp's Labeling Tools for Deep Learning имеет не слишком удобный графический интерфейс, представленный множеством, на первый взгляд, никак не связанных окон, что может с самого начала напугать пользователя, поэтому требует привыкания к данной системе.

Минусом, на мой взгляд, является и то, что размеченный стрим выводится в формате txt. Это чревато тем, что данный файл придется парсить с целью переформатирования, так как общепринятыми форматами размеченных данных являются ".json" и ".xml" Однако, плюсом данного решения является поддержка сегментации изображений.

2.8.2. LabelMe

Веб-приложение для разметки изображений. Распространяется свободно (при условии регистрации на сервере), а репозиторий, содержащий исходный код, является публичным на хостинге "GitHub".

Одним из главных плюсов этой программной системы является наличие уже размеченных датасетов, которыми можно воспользоваться как для тренировки разметчика, так и воспользоваться просто как примеров для создания собственного.

Несомненный факт, что поддержка полигонов для выделения объекта, является преимуществом данного решения для более точной калибрации данных и выделения объектов. Сложно не согласится, что полигоны достаточно точно опи-

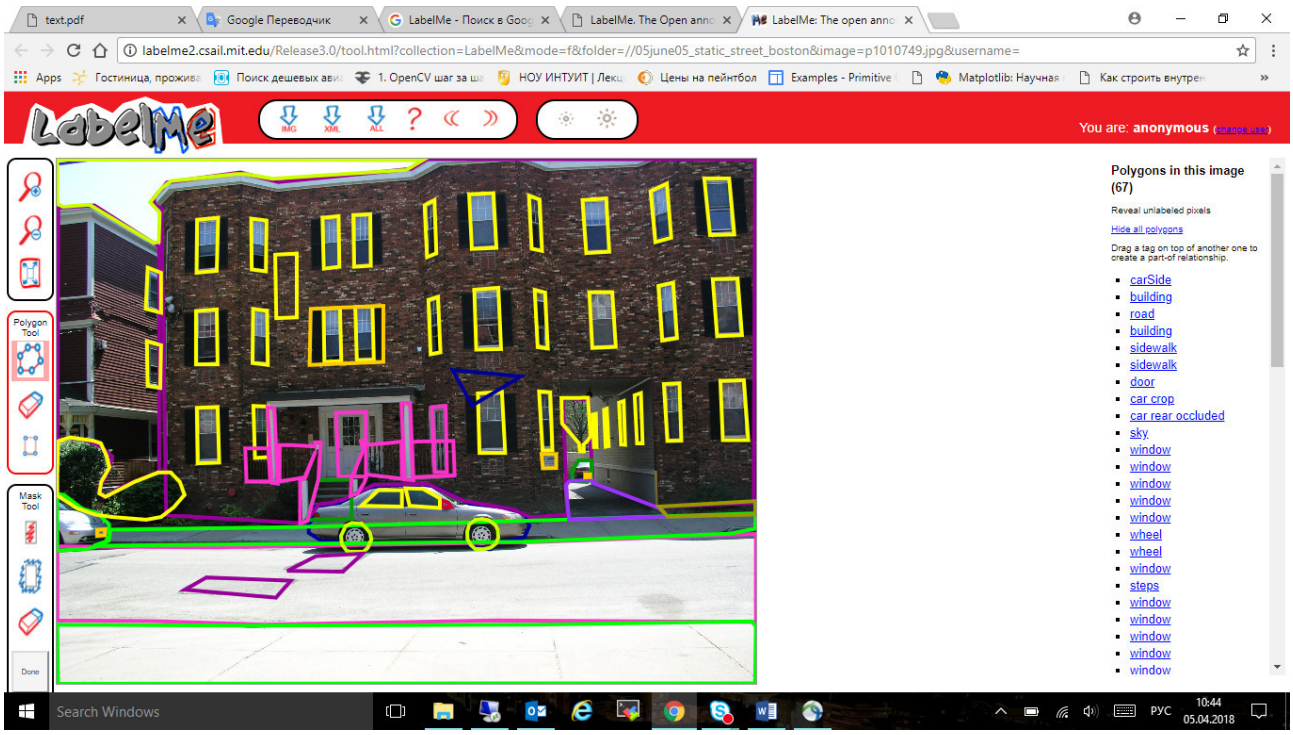

Рисунок 2.2. Скриншот веб-приложения "LabelMe"

сывают форму объекта, в отличие, например, от тех же прямоугольников. Графический интерфейс интуитивно понятен и удобен для использования, однако не могу сказать, что для он оказался слишком приятен. Главным его недостатком является дерево объектов, смотря на которое совершенно не понятно, как нм различать объекты одного типа, кроме как по координатам точек.

Кроме этого при масштабировании окна браузера, все объекты графического интерфейса «поехали» и перекрыли друг друга, что может сразу же отпугнуть пользователя.

Также не слишком понятной функциональностью является наличие масок, которые вроде бы должны отобрать неразмеченные области, а в итоге отображают сомнительные прямоугольники.

Однако, выходной стрим отформатирован как XML, что позволяет работать с ним как для обучения алгоритмов компьютерного зрения, так и просто парсить его в понятном для пользователя виде.

2.8.3. Яндекс.Толока

Веб-приложение, ориентированное не только на заказчиков разметки данных, но и для лейблеров, которые могут зарабатывать деньги, размечая данные (в индустрии такому явлению дано определение – CrowdSourcing - привлечение к решению задач инновационной производственной деятельности широкого круга лиц для использования их способностей, знаний и опыта по типу субподрядной работы на добровольных началах с применением инфокоммуникационных технологий.)

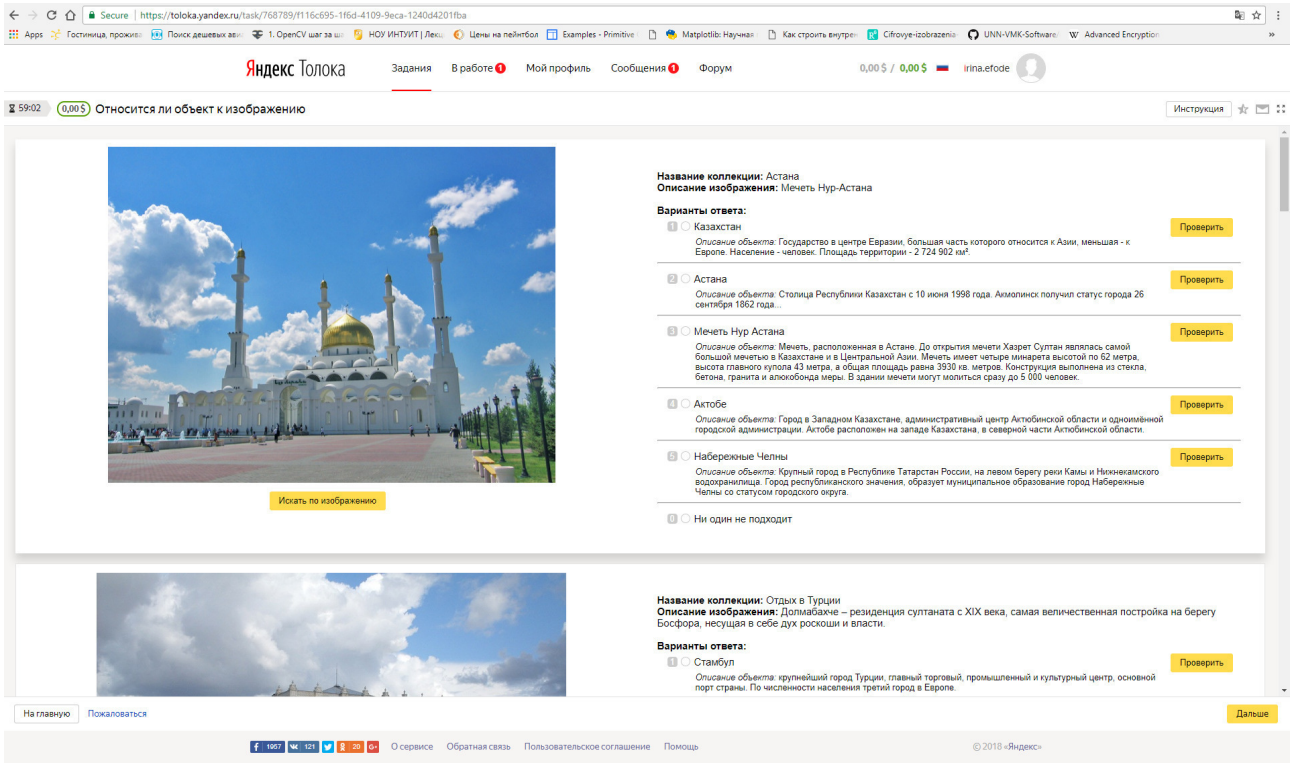

Рисунок 2.3. Скриншот программной системы "Яндекс.Толока"

Если рассматривать решение от Яндекса с точки зрения заказчика, то Яндекс дает некоторые гарантии на разметку, так как разметчик проходит обучение перед выполнением реальных задач (для того чтобы приступить к разметке данных, обучение должно быть выполнено правильно на 99,99 % правильно). Обучение заключается в разметке тестовой выборки согласно инструкции, содержащей в себе набор требований к разметке, которая дается пользователю перед

началом обучения. Главный нюанс в том, что данные предоставленные для обучения уже были размечены. Таким образом, к реальным задачам допускаются люди, которые реально могут справится с задачей разметки и выдать на финише корректные данные, которые соответствуют реальности.

Самое интересное в данном сервисе, что данные для разметки могут быть самые разные – от изображений и выделении на них объектов, то тегирования контента контекстной рекламы. Для выполнения задания выделяется некоторое время, а предложения сортируются по рейтингу, цене и давности.

Заказчик же разметки гарантирует наличие критериев разметки и наличие самих данных. Положительной стороной данного приложения является очень удобный пользовательский интерфейс, а также поддержка основных геометрических форм для обозначения объектов.

2.8.4. CrowdFlower

Этот инструмент – стандарт де-факто для разметки, принятый многими разметчиками. Лейблеры размечают данные вручную, однако данное решение предоставляет более продвинутые инсттрументы, чем например, Яндекс.Толока, для упрощения процесса разметки данных.

Помимо стандартных bounding boxes, семантической сегментации, полигонов, в CrowdFlower можно размечать такими геометрическими примитивами, как точки, что будет удобно, например, для разметки данных для обучения алгоритмов компьютерного зрения распознавания объектов на фотографиях складов или полок в магазинах.

Вообще, данное решение является довольно удобным, так как имеет интуитивно понятный и красивый графический интерфейс. Еще одной положительной стороной данного решения является то, что данный инструмент является веб-

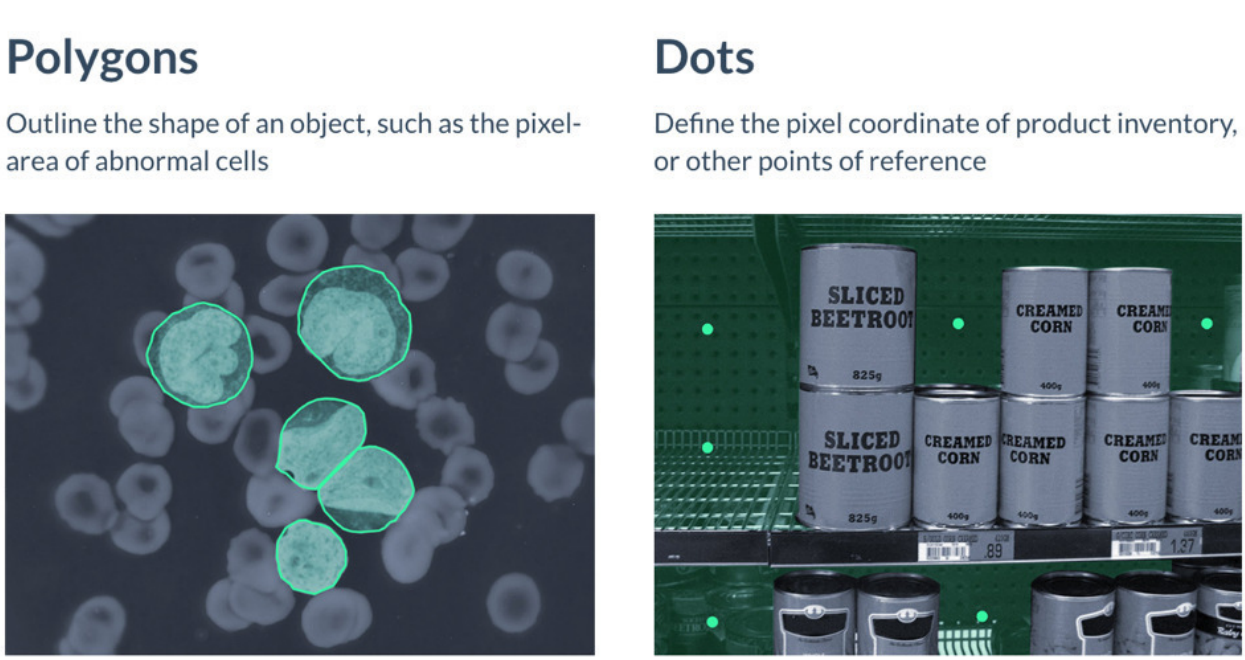

Рисунок 2.4. Скриншот программной системы "CroudFlower"

приложением и не требует установки на рабочую станцию. Большой стрим для разметки может быть поделен на несколько мелких задач, как и в другом инструменте – Intel® Labeler/Visualizer for Automotive.

3. Разработка структуры программной системы для разметки данных

В процессе разработки программной системы нужно проработать множество проблем, которые могут возникнуть на этапе девелопмента, а также проработать архитектуру приложения.

Разметка данных представляет собой многократное выделение объектов на изображении, а также заполнение их атрибутов, 3.1 представляет графическое отображение алгоритма работы программных систем создания наборов размеченных данных.

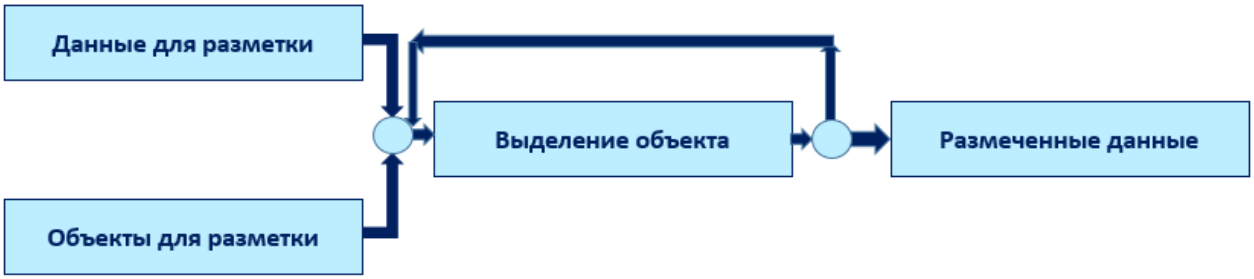

Рисунок 3.1. Алгоритм работы программных систем создания наборов размеченных данных

Правильная разметка данных позволит обучить алгоритм машинного обучения на тестовой выборке, размеченной вручную. Роль разметки данных для алгоритмов машинного обучения отображена на 3.2

Для правильной разметки данных при создании программной системы нужно предусмотреть наличие следующих компонент:

- Блок сериализации меток из внутреннего представления программной системы в JSON-файл, сохраняющий в себе размеченные данные;

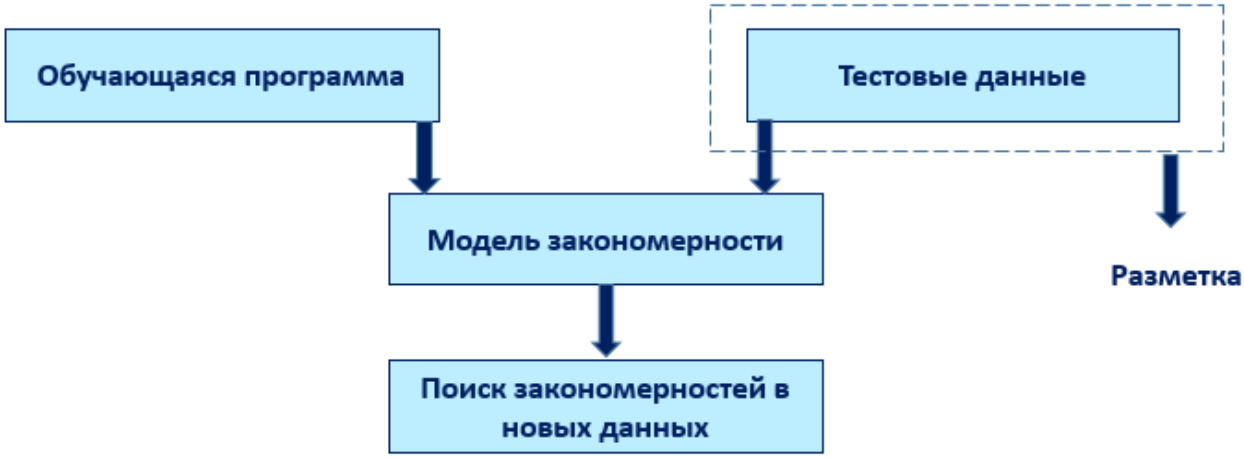

Рисунок 3.2. Алгоритм работы алгоритмов машинного обучения и компьютерного зрения, в частности

- Блок хранения меток во внутреннем представлении программной системы;
- Блок графического интерфейса – самый большой блок блок в данной программной системе, включающий в себя:
  - Тулбар – область, включающая в себя инструменты для создания и работы с метками;
  - Канвас – область для рисования прямоугольников, выделяющих область, в которой существует размечаемый объект;
  - Менюбар – область управления работой приложения, а также отображающая его основную функциональность;
  - Таббар – область, включающая в себя вкладки для редактирования и создания объекта, а также для отображения существующих объектов;
  - Область отображения изображения – область, поверх которой будет создана область рисования;
- Блок связи графического фронт-энда и бек-энда.

Графический интерфейс будет относится к фронт-энду (клиентской части приложения), а все остальные части приложения будут являются частями бек-энда (то есть программная часть программной системы).

Таким образом структурной схемой программной системы будет являться следующая структура: Опишем приведенную схему:

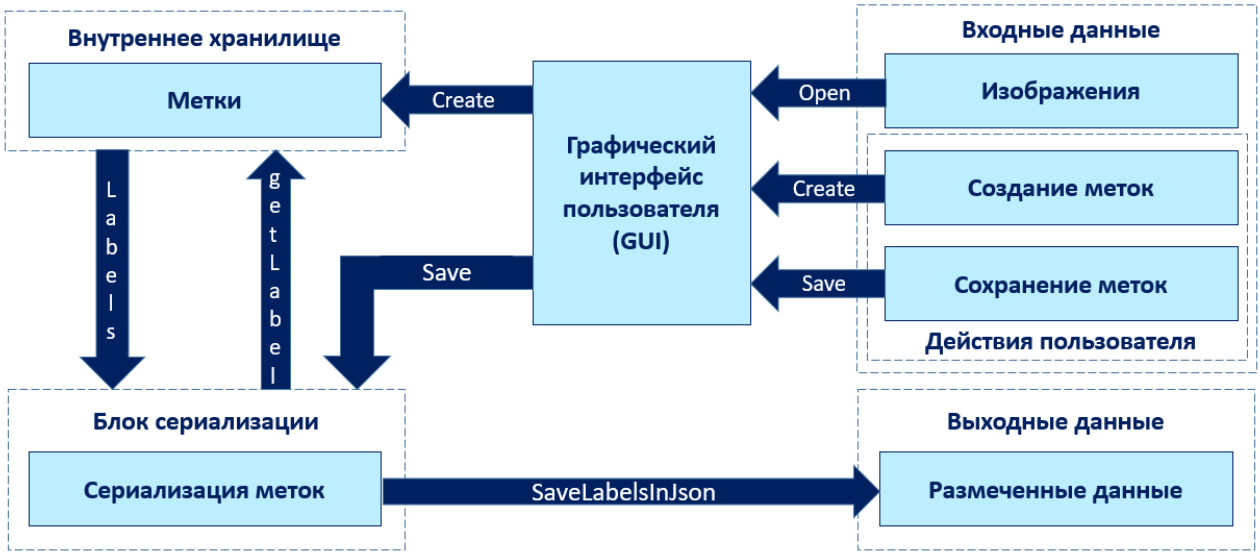

Рисунок 3.3. Структурная схема разрабатываемой программной системы

На вход данному приложению поступают изображения, предназначенные для разметки, а также действия пользователя – такие как рисование прямоугольника, обозначающего область, в которой находится размечаемый объект, а также редактирование названий объектов и их атрибутов в текстовых полях. Все эти действия обрабатываются графическим интерфейсом – на всякое действие пользователя GUI либо отправляет сигналы к нужной внутренней компоненте программной системы (например, создание и редактирование меток), либо же обрабатывает их сам (например, открывает изображение) и отправляет сигнал к внутренней компоненте (например, передает абсолютный путь к файлу).

После обработки действия пользователя графическим интерфейсом и отправки данных к нужной компоненте, вступает в работу внутренние части приложения – такие как «Блок хранения меток» и «Блок сериализации данных».

«Блок хранения меток» после получения получения сигнала о создании метки создает элемент во внутреннем представлении, ключом которого является имя объекта с его уникальным идентификатором, а его значением – введенная в по-

ле аттрибутов строка, преобразованная в JSON. При поступлении же сигнала о редактировании – метка с конкретным ключом изменяется – то есть осуществляется перезапись текущего значения или его ключа. После поступления запроса от компоненты «Блок сериализации данных» на получение данных «Блок хранения меток» возвращает метки во внутреннем представлении.

«Блок сериализации данных» - внутренняя компонента, которая при поступлении сигнала от графического интерфейса делает запрос на получение меток во внутреннем формате от компоненты «Блок хранения меток» и преобразует их строку, которая поддерживает хранение в JSON-формате, после чего записывает ее в файл “labels.json”, который будет сохранен рядом с размеченным изображением. Файл “labels.json” как раз и является выходными данными программной системы.

Для разметки данных прежде всего аналитику (или программисту) необходимо определить некоторый набор правил, отображающих объекты (или фичи), подлежащие разметке, а также формат хранения размеченных данных. Этап определения спецификации объектов включает в себя не только регистрацию самого типа объекта, но и определения его атрибутов и их атрибутов, в том числе фигуры (например, полигон, прямоугольник, точка, параллелепипед и так далее), которой объект на изображении будет размечен, если это разметка данных с целью обучения алгоритмов компьютерного зрения. Поэтому рассмотрим данный этап более детально.

4. Разработка программных средств

4.1. Разработка графического интерфейса

Самой большой частью программной системы для разметки изображений является модуль фронт-энда, который включает в себя не только рисование GUI, но и обработку действий пользователя, чтобы вызвать либо функциональность бек-энда, либо обработать данное действие на уровне фронт-энда.

Языком для написания клиентской части приложения мною был выбран декларативный язык QML, основанный на JavaScript. Данный фреймфорк уже несет в себе множество библиотек, содержащих множество примитивов графического интерфейса приложения - таких как кнопки, меню, вкладки и множество других.

Рассмотрим более подробную разработку некоторых элементов графического интерфейса, обращаясь к документации QT, указанную в списке источников:

- Строка меню (англ. Menubar) – элемент интерфейса пользователя, позволяющий выбрать одну из нескольких перечисленных опций программы. В индустрии принято размещать строку меню в верхней части приложения.

  В QML можно воспользоваться таким типом как MenuBar и добавить элементы MenuItem.

  Для покрытия разметки мне достаточно трех элементов строки меню – File (Основная функциональность, такая как Открыть, Закрыть, Сохранить файл и так далее), Label (редактирование, добавление и удаление меток) и Help (Основная информация о приложении и помощь, содержащая основные

горячие клавиши и некоторые кейсы).

Приведем часть кода, описывающей строку меню, элемент меню File, а также кнопку Open в нем:

```
menuBar: MenuBar {
id: menuBar

Menu {
    id: fileMenuItem
    title: "File"


    MenuItem {
        id: openFileMenuBarItem
        text: "Open file"
        shortcut: "Ctrl+O"
        onTriggered: {
            fileDialog.selectMultiple = true
            fileDialog.title = "Open file"
            fileDialog.selectFolder = false
            fileDialog.nameFilters =
                [ "Image filters (*.jpg *.jpeg *.png)" ]
            fileDialog.visible = true
        }
    }
```

- Панель инструментов (англ. ToolBar) – элемент графического интерфейса пользователя, ориентированный на работу с инструментами для создания и редактирования объектов.

  В случае разработки программной системы, обеспечивающей разметку данных, Панель инструментов будет частично дублировать функциональность, уже представленную в строке меню, например, создание и удаление меток.

  В QML уже существует такой тип как ToolBar, который предоставляет широкие возможности для проработки панели инструментов с точки зрения

фронт-энда. Приведем кусочек кода, которым описан тулбар и его элемент – кнопка создания метки, при нажатии на которую происходит перевод области рисования в режим добавления:

```
toolBar: ToolBar {
    id: toolBar
    RowLayout {
        ToolButton {
            id: addLabelToolBar
            iconName: "edit.svg"
            iconSource: "resources/edit.svg"

            onClicked: {
                canvasImage.state = "adding"
            }
```

- Область изображения – область, где помещается изображение для разметки. (Поверх данной области размещается область рисования.

  В QML есть специальный класс для декодирования и отображения изображений внутри GUI. Поэтому для решения данной задачи решено было воспользоваться им. Обратимся к коду, который описывает изображение на QML:

```
Image {
    id: image
    anchors.fill: parent
    x: parent.x + toolBar.width
    y: parent.y
```

  Таким образом, не придется дополнительно декодировать изображение.

- Область рисования поверх области изображения – область для создания и редактирования прямоугольника, выделяющего область объекта. Стоит от-

метить, что канвас должен переходит в состояние редактирования или добавления в ответ на действие пользователя – нажатие кнопки на панели инструментов или выбор элемента в строке меню. Чтобы не быть голословными, обратимся к коду, описывающему канвас:

```
Canvas {
    id: canvasImage
    x: image.x
    y: image.y
    width: image.width
    height:image.height
    visible: image.visible

    states: [
        State {
            name: "adding"
            when: createLabelButton.isPressed
```

- Вкладки для редактирования и создания объектов – видовое представление панели вкладок для отображения и редактирования меток.

  В QML уже существует такой тип как tabbar, который позволяет написать контроллер для конкретных целей. Для программной системы создания наборов размеченных данных это панель отображения списка меток и панель редактирования меток. Опишем их с использованием скрипта:

```
TabBar {
    id: tabbar
    width: parent.width
    TabButton {
        text: qsTr("Objects")
    }
    TabButton {
        text: qsTr("CreateLabel")
```

```
        }
    }

    StackLayout {
        width: parent.width
        currentIndex: bar.currentIndex
        Item {
            id: homeTab
        }
```

В этой секции были рассмотрены основные элементы графического интерфейса для программной системы создания наборов размеченных данных с целью алгоритмов компьютерного зрения. Однако каждый элемент – это не только набор атрибутов, но и обработка сценариев действий пользователя. Реализованный GUI удовлетворяет требованиям индустрии.

4.2. Разработка блока хранения меток во внутреннем представлении

При разработке данного элемента системы мною было принято решение хранить все метки в хеш-таблице, где ключом элемента будет является пара из имени объекта и его уникального идентификатора. Значением же будет является кортеж из вектора атрибутов, в том числе и прямоугольник, обозначающий область объекта, и имени размечаемого изображения.

STDLIB C++ представляет широкий выбор шаблонов и контейнеров для хранения и работы с данными. Кроме того, стоит также подключить библиотеку QtJSON, которая позволит обработать строку атрибутов правильным образом. Тогда контейнер для хранения меток будет выглядеть так:

```
class LabelStream {
private:
    std::map<std::pair<std::wstring, int>,
             std::tuple<std::vector<QJsonObject>,
```

```
                        std::wstring>> labelStream;

public:
    void setLabelStream(std::wstring objectName,
                        std::wstring attributes,
                        std::map<char, int> boungingBox);
    std::map<std::pair<std::wstring, int>,
             std::tuple<std::vector<QJsonObject>,
                        std::wstring>> getLabelStream();
publicSlots:
    void createLabel(std::wstring objectName,
                     std::wstring attributes,
                     std::map<char, int> boungingBox)
}
```

При такой организации кода у нас получается достаточно гибкое решение для организации хранилища меток, так как для хранения атрибутов используется вектор, а для ключа пара из названия объекта и его уникального номера.

4.3. Разработка блока сериализации данных

Сериализация – это процесс сохранения состояния объекта в последовательность байт, то есть, в нашем случае, это сохранение последовательности меток в выходной файл формата JSON. Для сериализации нам сначала нужно получит внутреннее представление данных, а затем перевести его в формат строки (бинарного представления). Таким образом класс будет выглядеть так:

```
class LabelSerialization {
private:
    QJsonDocument labels;

    std::wstring convertJsonToString();

public:
    void setLabelStream(std::map<std::pair<std::wstring, int>,
                        std::vector<QJsonObject>> labelStream);
    std::void createOutputFile();
```

```
publicSlots:
    void saveLabels()
}
```

В данном классе описано три метода – получения внутреннего представления меток, конвертации внутреннего хранилища в JsonDocument и создания выходного файла с созданными метками, кроме этого принимается еще и сигнал о действии пользователя, захотевшего сохранить размеченные данные. Этих данных достаточно для работы данного модуля.

5. Разработка формата хранения размеченных данных

Для разметки данных программисту (аналитику или разметчику) нужно определить набор правил, отображающих объекты или признаки, подлежащие разметке, а также формат хранения размеченных данных. Этап определения спецификации объектов включает в себя не только регистрации самого типа объекта, но и определения его атрибутов, в том числе фигуры (таких как параллелепипед, полигон, прямоугольник, точка и другие), которой будет размечен объект на изображении (если это разметка данных с целью обучения алгоритмов компьютерного зрения).

5.1. Формат представления размеченных данных

JSON (JavaScript Object Notation) – простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript, определенного в стандарте ECMA-262 3rd Edition – December 1999. JSON – текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными. основан на двух структурах данных:

- Коллекция пар ключ/значение. В разных языках, эта концепция реализована как объект, запись, структура, словарь, хеш, именованный список или ассоциативный массив.

- Упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

Это универсальные структуры данных. Почти все современные языки программирования поддерживают их в какой-либо форме. Логично предположить, что формат данных, независимый от языка программирования, должен быть основан на этих структурах.

- Массив – упорядоченная коллекция значений. Массив начинается с [(открывающей квадратной скобки) и заканчивается] (закрывающей квадратной скобкой). Значения разделены , (запятой). В JSON значения массива должны иметь тип string, number, object, array, boolean или null. для обозначения данного типа в строке используются квадратные скобки, а значения перечисляются через запятую.

  Пример:

```
‘‘Point2dList»: [
    {
        ‘‘x’’: 20,
        ‘‘y’’: 20
    },
    {
        ‘‘x’’: 30,
        ‘‘y’’: 30
    }
]
```

В данном случае был рассмотрен массив точек Point2dList, в котором хранятся всего лишь 2 объекта, значения которых “x”:20, y:”20” и x”:30”, “y”:30

- Строка – коллекция нуля или больше символов Unicode, заключенная в двойные кавычки, используя / (обратную косую черту) в качестве символа экранирования. Символ представляется как односимвольная строка. Похожий синтаксис используется в C и Java.

  Пример:

```
''Name'': ''Aaryn Green''
```

  То есть у атрибута типа ИМЯ есть значение "Aaryn Green", которое сохранено как строка.

- Объект – неупорядоченный набор пар ключ/значение. Объект начинается с (открывающей фигурной скобки) и заканчивается (закрывающей фигурной скобкой). Каждое имя сопровождается : (двоеточием), пары ключ/значение разделяются , (запятой).

  Пример:

```
{
    ''id'': 78,
    ''Name'': ''Alice Spencer'',
    ''Sex'': ''Female'',
    ''Age'': 27,
    ''Profession'': ''Software development engineer'',
    ''isHuman'': TRUE,
    ''BoundingBox'' :  {
        ''x1'': ''20'',
        ''x2'': ''40'',
        ''y1'': ''20'',
        ''y2'': ''40''
    },
    ''Eyes'': [
        {
            ''x'': 30,
```

```
            ‘‘y’’: 30
        },
        {
            ‘‘x’’: 35,
            ‘‘y’’: 30
        }
    ]
}
```

Итак рассмотрим размеченный объект, у которого есть атрибуты уникальный идентификатор со значением, равным целочисленному числу 78, Имя, значение которого равняется строке “Alice Spencer”, Пол и Профессия также являются строками, возраст – целочисленное число, это человек – логический тип, прямоугольник контура – структура (объект), а также Глаза – массив двумерных точек.

- Целочисленный тип – обозначается без кавычек через двоеточие

  Пример:

  ```
  «Age»: 17
  ```

  В данном примере атрибут возраст имеет значение 17

- Числовой нецелочисленный тип – обозначается без кавычек через двоеточие.

  Пример:

  ```
  ‘‘Angle’’: 73.5
  ```

- В данном примере атрибут угол имеет значение 73.5

- Логический тип – имеет значения TRUE или FALSE. Обозначается без кавычек Пример

  ```
  ‘‘isChildren’’: FALSE
  ```

В данном случае атрибут «Это ребенок» равен ЛОЖЬ

Так как на данных, поступающих разметчику для разметки, можно выделить самые различные объекты и их атрибуты (для получения более полной информации, обратитесь к книге Флаха "Машинное обучение указанной в библиографическом списке под номером 1), то на этапе разработки программной системы было решено использовать текстовые поля для ввода имени объектов и их атрибутов в форме JSON-строки. При этом имя объекта будет введено как строка, уникальный идентификатор объекта будет присвоен алгоритмом, а аттрибуты объекта введены как строка, которая может быть преобразована в JSON-формат.

Таким образом пользователь сам определяет атрибуты объекта, но при этом при некорректно введенной строке – которая не удовлетворяет требованиям формата JSON (на этапе заполнения будет произведена валидация введенной строки), пользователь получает предупреждение, после чего корректирует значение. Вызвать такое окно может незакрытая скобка или отсутствие запятой.

Согласно концепции JSON-формата, в выходной файл будет записан некий набор объектов и их атрибутов. Опишем формат выходных данных. В первую очередь в выходном файле

```
‘‘Image’’: ‘‘women_191082.jpeg’’,
‘‘Labels’’: [
{
        ‘‘id’’: 2,
        "Object": "Human",
        ‘‘Attributes’’ : {
            ‘‘Age’’: 18,
            ‘‘BoundingBox’’: {
                ‘‘x’’ : 12,
                ‘‘y’’ : 15,
                ‘‘h’’ : 58,
                ‘‘w’’ : 67
            },
```

```
            ‘‘Sex’’: ‘‘Woman’’
        }
    },
    {
        "id": 3,
        "Object": "Human",
        ‘‘Attributes’’ : {
            ‘‘Age’’: 18,
            ‘‘BoundingBox’’: {
                ‘‘x’’ : 12,
                ‘‘y’’ : 15,
                ‘‘h’’ : 58,
                ‘‘w’’ : 67
            },
            ‘‘Sex’’: ‘‘Woman’’
        }
    }
]
```

5.2. Разработка структуры внутреннего хранения размеченных данных

Для хранения меток внутри программной системы подходит такой тип данных как хеш-таблица, которая реализует интерфейс ассоциативного массива, а именно, позволяет хранить пары (ключ, значение) и выполнять основные операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу. Однако, усовершенствованные хеш таблицы умеют обменивать содержимое, выполнять различные условные поиски, сравнивать ключи и значения.

При хранении меток ключом будет является строка, являющаяся конкантенированной строкой из названия объекта и его уникального идентификационного номера. Значением же в этой паре будет являться набор атрибутов для размечаемого объекта, которые хранятся в JSON-формате.

6. Тестирование системы

Тестирование программной системы создания наборов размеченных данных для обучения алгоритмов компьютерного зрения может производится двумя методами – автоматизированными тестами для тестирования отдельных функций, модулей, всего бек-энда или фронт-энда или же полностью всей системы (так называемые интеграционные тесты), или же вручную (мануальное тестирование). Так как система является небольшой, то автоматизированные тесты использовать не очень выгодно, так как они не особенно сильно сократят время разработки. Поэтому мною было решено вручную тестировать программную систему по мере разработки тех или иных компонент. Разработка системы была начата с написания исходного кода графического интерфейса пользователя. Каждый из элементов GUI тестировался по мере его написания, проверяя исполнение сценариев действия пользователя, таких как нажатие на кнопку, выбор файла, рисование bounding box, внесение атрибутов и так далее. Приведем пример тестирования отдельного элемента графического интерфейса пользователя – пользователь хочет посмотреть информацию об используемом приложении, для этого в строке меню он выбирает поле "Help а потом элемент "About". В ответ на эти действия приложение должно вывести на экран окно с основной информации о приложении, как указано на рисунке 6.1.

А также приведем пример тестирования всего приложения – это основная задача разрабатываемой системы. Для этого создадим некоторую метку, нарисуем для нее bounding box и зададим некоторые атрибуты, как это сделано на 6.2

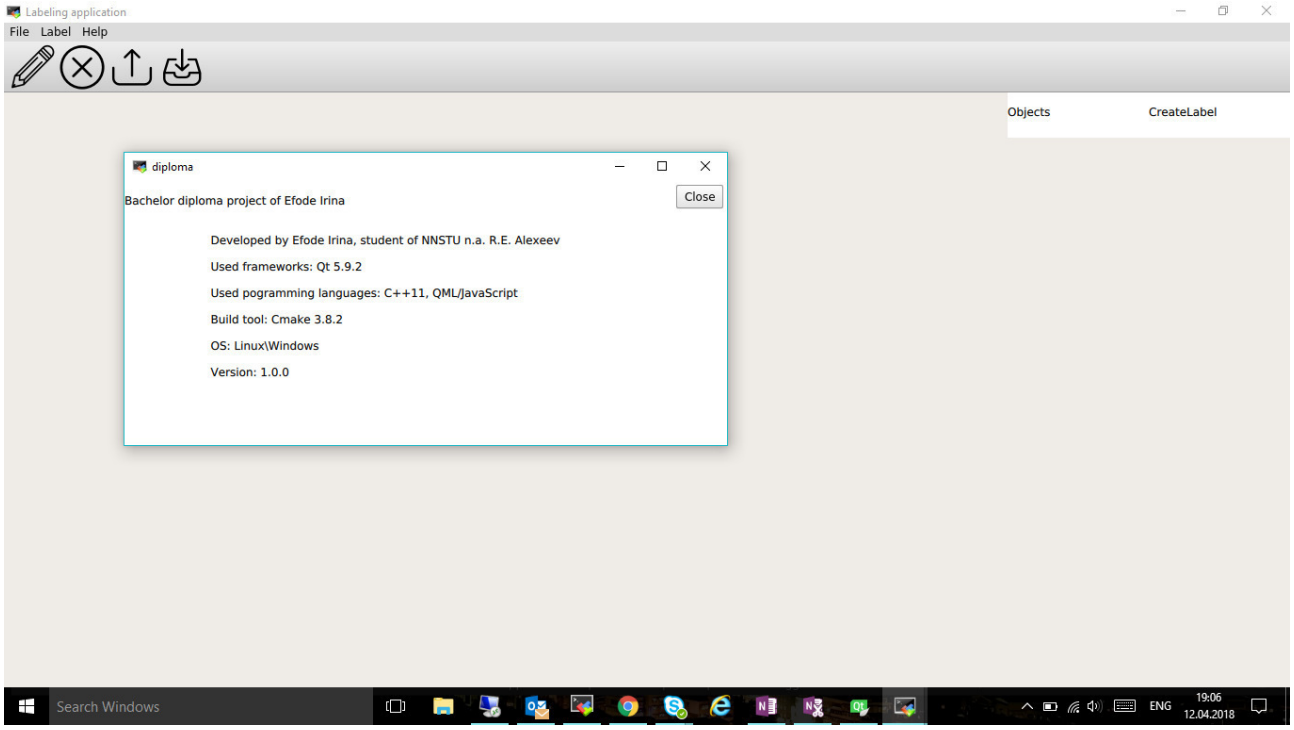

Рисунок 6.1. Скриншот разработанной программной системы, работающей под управлением Windows10

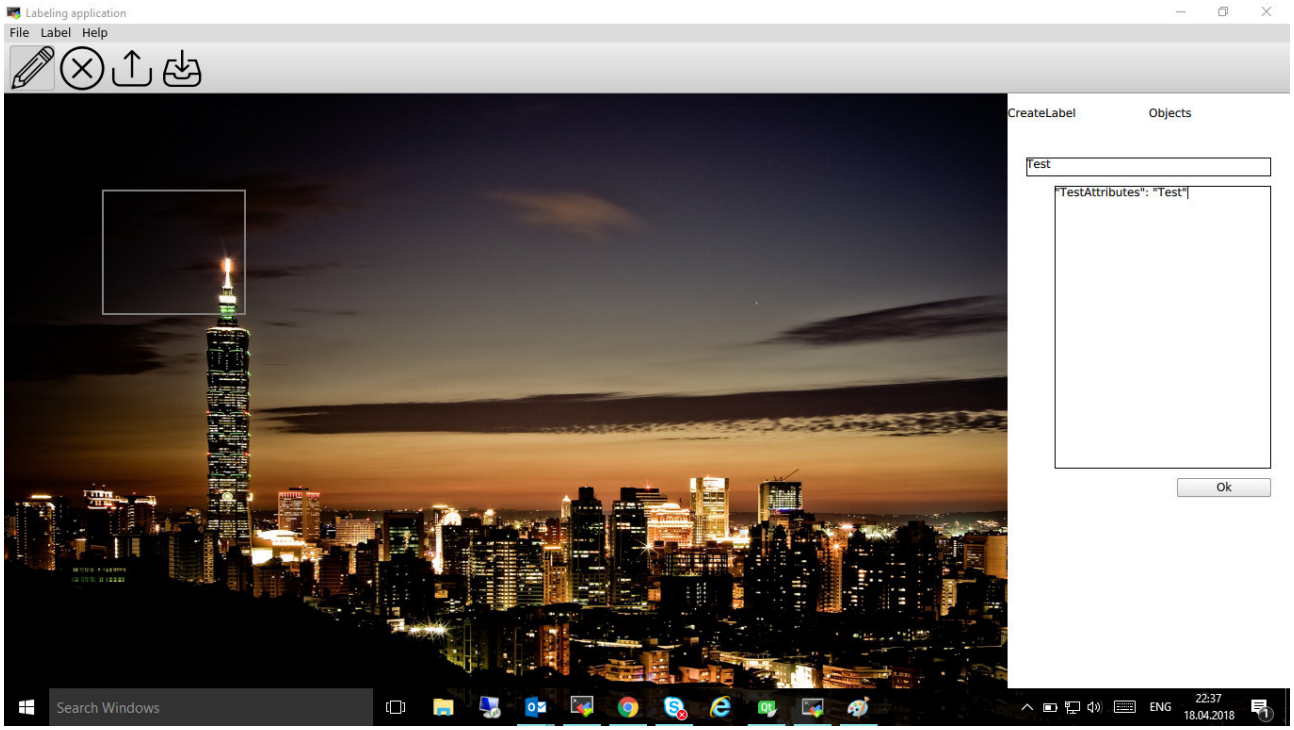

Рисунок 6.2. Скриншот разработанной программной системы, работающей под управлением Windows10

На 6.2 изображено создание метки "Test

с атрибутами: "TestAtt": "Hi!

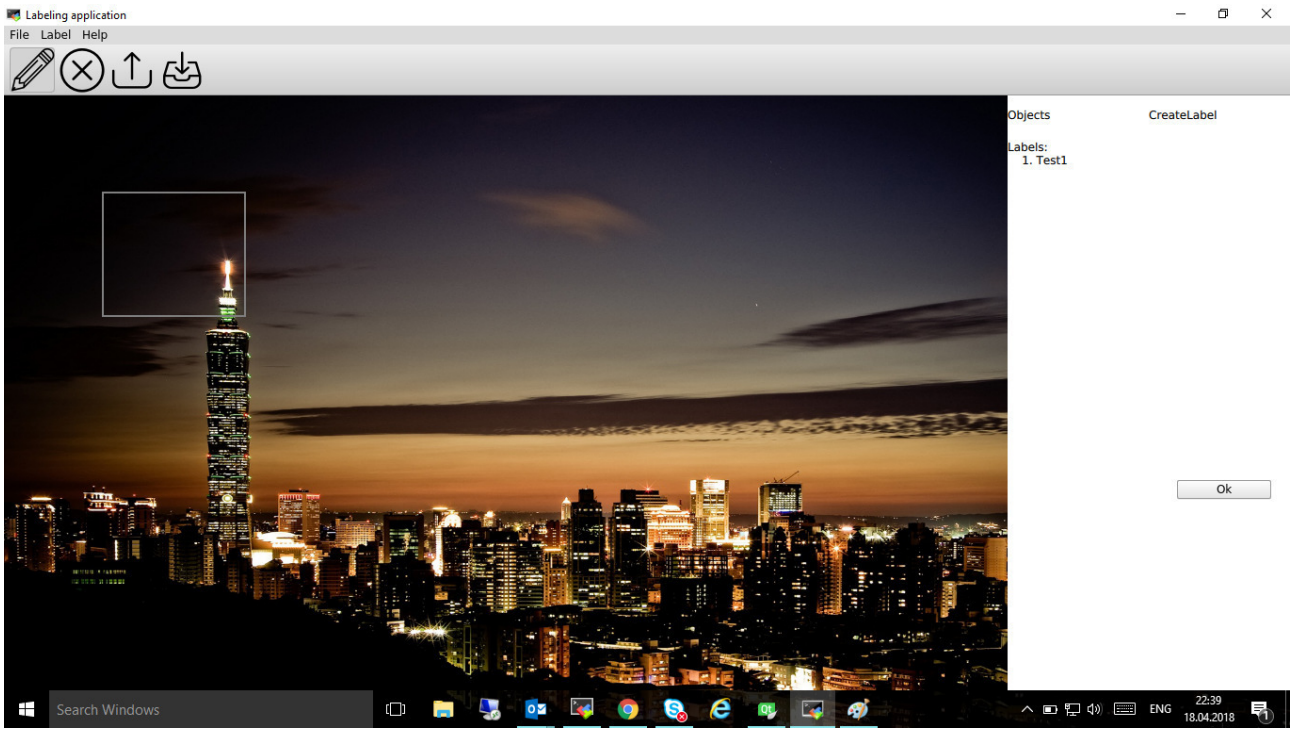

Рисунок 6.3. Скриншот разработанной программной системы, работающей под управлением Windows10

и нарисованный пользователем bounding box. Сразу приведем скриншот полученного файла:

dataset_io_orc.py | "Image": "women_191082.jpeg", | labels.json | dump.txt

```
{
  "Image": "test.jpg",
  "Labels": [
    {
      "id": 1,
      "Object": "Test",
      "Attributes": {
        "BoundingBox": {
          "x": 104,
          "y": 102,
          "h": 106,
          "w": 102
        },
        "TestAttributes": "Test"
      }
    }
  ]
}
```

Рисунок 6.4. Скриншот полученного после разметки файла

Мы сразу можем отметить, что на первый взгляд данные соответствуют реальности.

Внутренние компоненты тестировались двумя способами:

- Тестирование компоненты с использованием отладчика – то есть мы просматриваем, что хранится в переменных при том или иной состоянии системы;
- Тестирование всего приложения и сравнение реальных данных с полученными в файле "labels.json"– это тестирование функциональности приложения без использования дополнительных программных средств.

В заключение можно сказать, что тестирование подразумевает под собой не только валидацию функциональности, но и поиск проблем в программной системе, что никогда не помешает в разработке любого проекта.

Заключение

В процессе выполнения выпускной квалификационной работы было поставлено и решено множество задач, из которых наиболее значимыми являлись:

- Разработка задания на дипломную работу и определение сроков ее выполнения;
- Детальный подбор инструментов разработки программной системы;
- Разработка архитектуры приложения;
- Разработка формата хранения размеченных данных;
- Непосредственно сама разработка программной системы;
- Тестирование системы.

Данные задачи являются каждодневными и для реальных разработчиков программного обеспечения, поэтому, при решении проблемы создания программной системы для выпускной квалификационной работы, был получен ценный опыт.

Однако, стоит отметить, что разработанное приложение далеко от идеала, поэтому, как и многие системы, требует доработки.

Библиографический список

1. Флах П. Машинное обучение. — М.: ДМК Пресс, 2015. — 400 с.
2. I. H. Witten, E. Frank Data Mining: Practical Machine Learning Tools and Techniques (Second Edition). — Morgan Kaufmann, 2005.
3. Qt Documentation [Электронный ресурс]. – Режим доступа: http://doc.qt.io – Заглавие с экрана. – (Дата обращения: 02.12.2017).
4. Хабрахабр [Электронный ресурс]. – https://habrahabr.ru/ – Заглавие с экрана. – (Дата обращения: 04.02.18).
5. Машинное обучение [Электронный ресурс]. – http://www.machinelearning.ru – Заглавие с экрана. – (Дата обращения: 04.02.18).
6. Флах П. Машинное обучение. — М.: ДМК Пресс, 2015. — 400 с.
7. Ефоде И.М., Гай В.Е. Программная система создания наборов размеченных данных для обучения алгоритмов компьютерного зрения. Материалы конференции "ИСТ–2018 2018г.

Приложение А - Листинг приложения

```
# CMakeList.txt

cmake_minimum_required(VERSION 2.8.12)

project(diploma LANGUAGES CXX)

set(CMAKE_INCLUDE_CURRENT_DIR ON)
set(CMAKE_AUTOMOC ON)
set(CMAKE_AUTORCC ON)

find_package(Qt5 COMPONENTS Core Quick REQUIRED)

INCLUDE_DIRECTORIES(src)

add_executable(${PROJECT_NAME} $src "qml.qrc")

target_link_libraries(${PROJECT_NAME} Qt5::Core Qt5::Quick)


// src/main.cpp
#include <QGuiApplication>
#include <QQmlApplicationEngine>

#include "labelstream.h"
#include "labelserialization.h"

int main(int argc, char *argv[])
{
    QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QGuiApplication app(argc, argv);

    QQmlApplicationEngine engine;
    engine.load(QUrl(QLatin1String("qrc:/main.qml")));
```

```
    if (engine.rootObjects().isEmpty())
        return -1;

    LabelStream labels{};
    LabelSerializtion serialization{};

    return app.exec();
}
```

```
\\src/labelstream.h
#ifndef LABELSTREAM_H
#define LABELSTREAM_H

#include <map>
#include <utility>
#include <string>
#include <vector>

#include <QJsonObject>

#include <QObject>

class LabelStream
{
private:
    std::map<std::pair<std::wstring, int>,
             std::vector<QJsonObject>> labelStream;
    int objectId;


    void createNewLabel(std::wstring name,
                        std::wstring attributes,
                        std::map<char, int> label);

public:
    LabelStream();

    std::map<std::pair<std::wstring, int>,
            std::vector<QJsonObject>> getLabelStream();

publicSlots:
    void createLabel(std::wstring name,
```

```
                        std::wstring attributes,
                        std::map<char, int> label);
    }

#endif
```

```
    \\src/labelstream.cpp
    #include "labelstream.h"
    #include <QJsonDocument>
    #include <QJsonArray>
    #include <QJsonValue>

    LabelStream::LabelStream()
    : objectId(0) {}

    std::map<std::pair<std::wstring, int>,
            std::vector<QJsonObject>> LabelStream::getLabelStream()
    {
        return labelStream;
    }
    void LabelStream::createNewLabel(std::wstring name,
                                     std::wstring attributes,
                                     std::map<char, int> boundingBox)
    {
        QJsonDocument doc = QJsonDocument.fromBinaryData(attributes);
        QJsonArray attributes = doc.array();
        QJsonObject boundingBoxObject;
        for (auto item : boundingBox)
        {
            boundingBox.insert(item.key(), QJsonValue(item.value()))
        }

        std::vector<QJsonObject> objectAttributes;
        objectAttributes.push_back(boundingBox);
        objectAttributes.push_back(attributes);
        labelStream.insert(std::make_pair(name, objectId),
                           objectAttributes);

        objectId++;
    }

    void LabelStream::createLabel(std::wstring name,
```

```
                                    std::wstring attributes,
                                    std::map<char, int> label)
    {
        createNewLabel(name, attributes, label)
    }
}


    //src/labelserialization.h
    #ifndef LABELSERIALIZATION_H
    #define LABELSERIALIZATION_H

    #include <map>
    #include <utility>
    #include <string>
    #include <vector>

    #include <QJsonDocument>

    class LabelSerializtion
    {
    private:
        QJsonDocument labels;

        void wstring convertJsonToString(
            td::map<std::pair<std::wstring, int>,
            std::vector<QJsonObject>> labelStream);
        void saveIntoDocument(std::wstring path);

    public:
        LabelSerializtion();

    publicSlots:
        void saveLabels(str::wstring path)
    }

#endif

    //src/labelstream.cpp
    #include "labelserializtion.h"

    #include<QJsonDocument>
    #include <QJsonObject>
```

```
#include <fstream>

void wstring LabelSerializtion::convertJsonToString(
    std::map<std::pair<std::wstring, int>,
    std::vector<QJsonObject>> labelStream)
{
    std::vector<QJsonObject> labels;
    for (auto item : labelStream)
    {
        QJsonObject idObj =
            QJsonObject(std::wstring("id"), item.key.second());
        QJsonObject nameObj =
            QJsonObject(std::wstring("Object"), item.key.first());
        QJsonArray attributes;
        for (const auto& item : labelStream)
        {
            attributes.push_back(item);
        }
        QJsonObject attribObj =
            QJsonObject(str::wstring("Attributes"), attributes);
        labels.push_back (QJsonObject({idObj, nameObj, attribObj}))
    }
    QJsonDocument result = QJsonDocument({std::wstring("Image"), pat
    std::wstring wstringRes = result.toJson();
    return wstringRes;
}

void LabelSerializtion::saveIntoDocument(
                            std::wstring path,
                            std::map<std::pair<std::wstring, int>,
                                     std::vector<QJsonObject>> label
{
    std::ofstream fout(paht+"labels.json");
    fout << convertJsonToString(labelStream);
    fout.close();
}

LabelSerializtion::LabelSerialization();

#endif

//main.qml
```

```
import QtQuick 2.7
import QtQuick.Controls 2.0
import QtQuick.Layouts 1.3
import QtQuick.Controls 1.4

import QtQuick.Controls 1.4         //toolbar and menubar
import QtQuick.Controls.Styles 1.4

import QtQuick.Dialogs 1.0 //open explorer
import QtQuick 2.2

import QtQuick.Window 2.2 //window help


ApplicationWindow {
    id: mainWindow
    visible: true
    width: 640
    height: 480
    title: qsTr("Labeling application")

    FileDialog {
        id: fileDialog
        visible: false
        selectExisting: true

        onAccepted: {
            console.log("You chose: "\
            + fileDialog.fileUrls)
            //multiple
            image.source = fileUrl
            image.visible = true
            }
        onRejected: {
            console.log("Canceled")
            visible = false
        }
    }

    Window {
        id: aboutProjectWindow
        visible: false
```

```
            width: 640
            height: 280
            Text {
                id: aboutProjectText
                text: qsTr("\nBachelor diploma project
                    of Efode Irina\n\n
                    Developed by Efode Irina, student of NNSTU
                    n.a. R.E. Alexeev\n
                    Used frameworks: Qt 5.9.2\n
                    Used pogramming languages: C++11, QML/JavaScript\n
                    Build tool: Cmake 3.8.2\n
                    OS: Linux\\Windows\n
                    Version: 1.0.0")
            }
            Button {
                id: closeHelpDialogButton
                width: 50
                height: 25

                x: parent.x + parent.width - width - 4
                y: parent.y + 4

                text: "Close"
                onClicked: {
                    aboutProjectWindow.visible = false
                }
            }
        }

    Image {
    id: image
    anchors.fill: parent
    x: parent.x
    y: parent.y

    MouseArea {
    id:imadema
    anchors.fill: parent

    onClicked: {
                    if (rectangle.x == -1 && rectangle.y == -1) {
                        rectangle.visible = true
```

```
                    rectangle.x = mouseX
                    rectangle.x = mouseY
                }
if (rectangle.x != -1 && rectangle.y != -1) {
rectangle.visible = true
rectangle.width = mouseX - rectangle.x
rectangle.height = mouseY - rectangle.y
}
}

onPositionChanged: {
if (addLabelToolBar.checked && rectangle.x\
!= -1 && rectangle.y != -1) {
rectangle.width = mouseX - rectangle.x
rectangle.height = mouseY - rectangle.y
}
}
}
        visible: true
        BorderImage {
            border.bottom: 10
            border.left: 10
            border.right: 10
            border.top: 10
        }
    }

    Rectangle {
        id: rectangle
        color: "transparent"
        visible: false
        border.color: "grey"
        x: -1
        y: -1

        x: 104
        y: 102
        border.width: 2
    }


    menuBar: MenuBar {
```

```
id: menuBar

Menu {
    id: fileMenuItem
    title: "File"


    MenuItem {
        id: openFileMenuBarItem
        text: "Open file"
        shortcut: "Ctrl+O"
        onTriggered: {
            fileDialog.selectMultiple = true
            fileDialog.title = "Open file"
            fileDialog.selectFolder = false
            fileDialog.nameFilters =
            [ "Image filters (*.jpg *.jpeg *.png)" ]
            fileDialog.visible = true
        }
    }
    MenuItem {
        id: importStreamMenuBarItem
        text: "Import stream"
        shortcut: "Ctrl+Shift+O"
        onTriggered: {
            fileDialog.selectMultiple = false
            fileDialog.title = "Import stream"
            fileDialog.selectFolder = true
            fileDialog.nameFilters = \
            [ "All file filters (*.*)" ]
            fileDialog.visible = true
        }
    }
    MenuItem {
        id: saveDataMenuBarItem
        text: "Save changed data"
        shortcut: "Ctrl+S"
        onTriggered: {
            fileDialog.selectMultiple = false
            fileDialog.title = "Save stream"
            fileDialog.selectFolder = true
            fileDialog.nameFilters =
```

```
                [ "All file filters (*.*)" ]
                fileDialog.visible = true
            }
        }
        MenuItem {
            id: exportStreamMenuBarItem
            text: "Export stream"
            shortcut: "Ctrl+Shift+S"
            onTriggered: {
                fileDialog.selectMultiple = false
                fileDialog.title = "Export stream"
                fileDialog.selectFolder = true
                fileDialog.nameFilters =
                [ "All file filters (*.*)" ]
                fileDialog.visible = true
            }
        }
        MenuItem {
            id: closeMenuBarItem
            text: "Close"
            shortcut: "Ctrl+ESC"
        }
        MenuItem {
            id: exitMenuBarItem
            text: "Exit"
            shortcut: "Ctrl+Q"
            onTriggered: Qt.quit()
        }
    }

    Menu {
        id: labelMenuItem
        title: "Label"

        MenuItem {
            id: removeAllLabelsMenuBarItem
            text: "Remove all labels"
            shortcut: "Ctrl+D"
        }
        MenuItem {
            id: addNewLabelMenuBarItem
            text: "Add new label"
```

```
                shortcut: "Ctrl+N"
            }
            MenuItem {
                id: undoLabelMenuBarItem
                text: "Undo"
                shortcut: "Ctrl+Z"
            }
            MenuItem {
                id: redoLabelMenuBarItem
                text: "Redo"
                shortcut: "Ctrl+Y"
            }
        }

        Menu {
            id: helpMenuItem
            title: "Help"

            MenuItem {
                id: helpMenuBarItem
                text: "Help"
                shortcut: "F1"
            }
            MenuItem {
                id: aboutMenuBarItem
                text: "About..."

                onTriggered: {
                    aboutProjectWindow.visible = true
                }
            }
        }

    }

    toolBar: ToolBar {
        id: toolBar

        RowLayout {
            ToolButton {
                id: addLabelToolBar
                iconName: "edit.svg"
```

```
                iconSource: "resources/edit.svg"
                checkable: true

                onClicked: {
//                    canvasImage.state = "adding"
                }
            }
            ToolButton {
                id: removeLabelToolBar
                iconName: "Label"
                iconSource: "resources/cancel.svg"
                onClicked: {
                    rectangle.visible = false
                }
            }
            ToolButton {
                id: uploadToolBar
                iconName: "Upload"
                iconSource: "resources/upload.svg"
                onClicked: {
                    rectangle.visible = true
                }
            }
            ToolButton {
                id: downloadToolBar
                iconName: "Download"
                iconSource: "resources/download.svg"
            }
        }
    }

    TabBar {
        id: tabbar
        x: parent.width - 300
        width: parent.width - x
        TabButton {
            height: 20
            Text {
                text: qsTr("Objects")
                anchors.fill: parent
            }
            onClicked: {
```

```
                objerect1.visible = true
                enterrect.visible = false
            }

        }
        TabButton {
            height: 20
            Text {
                text: qsTr("CreateLabel")
                anchors.fill: parent
            }
            onClicked: {
                enterrect.visible = true
                objerect1.visible = false
            }

        }
    }

    Rectangle {
        id: objerect
        visible: true
        x: tabbar.x
        y: tabbar.y + tabbar.height
        width: tabbar.width
        height: 1000
        color: "white"
    }

    Rectangle {
        id: objerect1
        visible: false
        x: tabbar.x
        y: tabbar.y + tabbar.height
        width: tabbar.width
        height: 1000
        color: "white"
        Text {
            id: name
            text: qsTr(emit getLabels())
        }
    }
```

```
Rectangle {
    id: enterrect
    visible: false
    x: tabbar.x
    y: tabbar.y + tabbar.height
    width: tabbar.width
    height: 1000
    color: "white"


}

Rectangle{
    id: rectin
    border.color: "black"
    visible: enterrect.visible
    x: enterrect.x + 20
    y: enterrect.y + 20
    width: tabbar.width - 40
    height: 20
    focus: false
}

Rectangle{
    id: alzksl
    visible: namein.visible
    border.color: "black"
    color: "transparent"
    x: namein.x + 30
    y: namein.y + 30
    width: tabbar.width - 70
    height: 300
    focus: false
}

TextInput{
    id: namein
    text: "Name"
    visible: enterrect.visible
    x: enterrect.x + 20
    y: enterrect.y + 20
```

```
            width: tabbar.width - 40
            height: 20
            focus: true
        }

        TextInput{
            id: textin
            text: "Attributes"
            visible: namein.visible
            x: namein.x + 30
            y: namein.y + 30
            width: tabbar.width - 70
            height: 300
            focus: true
        }
        Button {
            x: textin.x + 130
            y: textin.y + textin.height + 10
            width: 100
            height: 20
            text: "Ok"
            onClicked: {
    emit createNewLabel(namein.text, textin.text,
    {"x": rectangle.x,
     "y": rectangle.y,
     "w": rectangle.width,
     "h": rectangle.height})
                textin.text = "Attributes"
                namein.text = "Name"
            }
        }


        StackLayout {
            width: parent.width
            currentIndex: bar.currentIndex
            Item {
                id: homeTab
                width: tabbar.width
                height: tabbar.height
            }
            Item {
```

```
                id: discoverTab
            }
        }
    }
```