**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Специальность 1-40 05 01-01 Информационные системы и технологии (в проектировании и производстве)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Программирование сетевых приложений»

на тему: «**СЕТЕВОЕ ПРИЛОЖЕНИЕ ДЛЯ ФОРМИРОВАНИЯ ЗАДАЧ ПО ОБРАБОТКЕ ДАННЫХ ЛОКАЛЬНО НА РОБОТЕ ИЛИ УДАЛЕННО НА СЕРВЕРЕ**»

Исполнитель: студент гр. ИТП-41

Расшивалов Н.И.

Руководитель: доцент

Курочка К.С.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2022

СОДЕРЖАНИЕ

Введение 4

1 Средства для реализации нейронных сетей 5

1.1 Особенности обнаружения объектов на изображениях 5

1.2 Задача обнаружения объектов на изображении 10

1.3 Средства разработки программного обеспечения 11

2 Архитектура программного комплекса по формированию задач для обнаружения объектов 14

2.1 Принцип взаимодействия приложения по обнаружению объектов с сервером 14

2.2 Архитектура *YOLOv5* 15

2.3 Набор данных для обучения нейронной сети 17

2.4 Обучение модели *YOLOv5* 18

2.5 Структура программного комплекса по обнаружению объектов 20

3 Структура и верификация приложения по формированию задач для обнаружения объектов 22

3.1 Структура классов и интерфейсов программного комплекса по формированию задач для обнаружения объектов 22

3.2 Нагрузочное тестирование клиентского приложения по обнаружению плодов 25

3.3 Верификация разработанного программного обеспечения 27

Заключение 29

Список использованных источников 30

Приложение А Листинг приложения 31

Приложение Б Схема взаимодействия классов 42

Приложение В Руководство системного программиста 43

Приложение Г Руководство программиста 44

Приложение Д Руководство пользователя 45

# **ВВЕДЕНИЕ**

В настоящее время искусственные нейронные сети широко используются при решении самых разнообразных задач особенно там, где обычные алгоритмические решения оказываются неэффективными или вовсе невозможными. Например, при распознавании текстов, объектов и др. Решения на основе искусственных нейронных сетей становятся все более совершенными и популярными.

Нейронные сети являются одним из направлений научных исследований в области создания искусственного интеллекта, в основе которого лежит стремление подражать нервной системе человека. Нейронные сети основаны на примитивной биологической модели нервной системы.

Преимущества нейронных сетей, обусловлены возможностью распараллеливания обработки информации, а также самообучением, т. е. возможностью обобщать – способность получать обоснованный результат на основании данных, не встречавшихся ранее в процессе обучения. Указанные преимущества позволяют искусственным нейронным сетям решать сложные задачи, считающиеся на сегодняшний день трудноразрешимыми.

Актуальность разработки данного программного обеспечения обусловлена необходимостью автоматизации процессов, которые на данный момент не возможны без активного участия человека. Одним из таких процессов является сбор плодов. В тоже время данная задача обнаружения объектов хорошо поддается распределению за счет разбиения видеопотока на кадры, которые могут быть обработаны удаленно на сервере или локально на роботе.

Целью курсового проекта является разработка программного обеспечения, позволяющего формировать задачи для обработки данных локально на роботе или удаленно на сервере, производить определение объектов в реальном времени с использованием искусственных нейронных сетей.

Задачи курсового проекта можно определить следующим перечнем:

– подготовить набор данных;

– обучить нейронную сеть;

– разработать программное обеспечение, использующее нейронную сеть, позволяющее формировать задачи для обработки данных локально на роботе или удаленно на сервере.

# **1 СРЕДСТВА ДЛЯ РЕАЛИЗАЦИИ НЕЙРОННЫХ СЕТЕЙ**

* 1. **Особенности обнаружения объектов на изображениях**

Обнаружение объектов – это метод [компьютерного зрения](https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b), целью которого является обнаружение таких объектов, как автомобили, здания и люди, и это лишь некоторые из них. Обычно объекты можно идентифицировать [либо по изображениям, либо по видеоканалам](https://heartbeat.fritz.ai/detecting-objects-in-videos-and-camera-feeds-using-keras-opencv-and-imageai-c869fe1ebcdb).

[Обнаружение объектов](https://www.fritz.ai/features/object-detection.html) широко применяется в видеонаблюдении, беспилотных автомобилях и [отслеживании объектов/людей](https://heartbeat.fritz.ai/real-time-person-tracking-on-the-edge-with-a-raspberry-pi-93ae636af9fa). В этой главе рассматриваются основы обнаружения объектов и некоторые из наиболее часто используемых алгоритмов, а также несколько совершенно новых подходов.

Обнаружение объекта идентифицирует присутствие объекта на изображении и рисует ограничивающую рамку вокруг этого объекта. Обычно это включает два процесса:

* классификацию и тип объекта;
* рисование рамки вокруг этого объекта.

Одни из самых распространенных архитектур моделей, используемых для обнаружения объектов [1, c.504]:

* *R-CNN*;
* *Fast R-CNN*;
* *Faster R-CNN*;
* *Mask R-CNN*;
* *YOLO*.

***1.1.1*** Модель*R-CNN* сочетает в себе два основных подхода:

* применение высокопроизводительных сверточных нейронных сетей к восходящим предложениям регионов для локализации и сегментации объектов;
* контролируемая предварительная подготовка к вспомогательным задачам.

Архитектура сети *R-CNN (Regions With CNNs*) была разработана командой из *UC* *Berkley* для применения *Convolution* *Neural* *Networks* к задаче *object* *detection*. Существовавшие на тот момент подходы к решению таких задач приблизились к максимуму своих возможностей и значимо улучшить их показатели не получалось.

*CNN* хорошо показывали себя в классификации изображений, и в данной сети они по сути были применены для того же самого [2, c.202]. Для этого на вход *CNN* подавалось не всё изображение целиком, а предварительно выделенные другим способом регионы, на которых предположительно имеются какие-то объекты.

В качестве *CNN*-сети использовалась так же готовая архитектура – *CaffeNet* (*AlexNet*). Такие нейросети, как и другие для набора изображений *ImageNet*, проводят классификацию на 1000 классов. *R-CNN* разрабатывалась для детектирования объектов меньшего количества классов (*N*= 20 или 200), поэтому последний классификационный слой *CaffeNet* был заменён на слой с *N*+1 выходами (с дополнительным классом для фона).

Несмотря на то, что *CNN* тренировалась на распознавание *N*+1 классов, в итоге она использовалась только для извлечения фиксированного 4096-размерного вектора признаков. Непосредственным определением объекта на изображении занимались *N* линейных *SVM*, каждый из которых проводил бинарную классификацию по своему типу объектов, определяя есть ли такой в переданном регионе или нет. В оригинальном документе вся процедура иллюстрируется схемой, которая указана на рисунке 1.1.

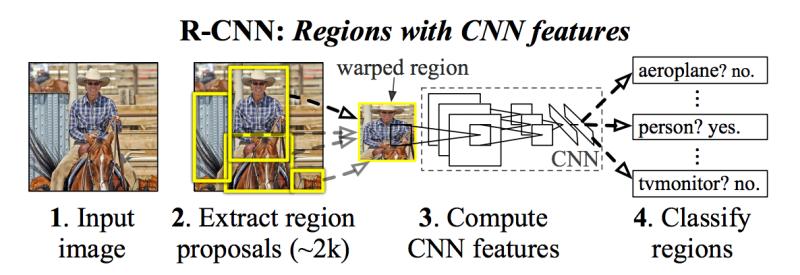


Рисунок 1.1 – Процедура извлечения признаков *CNN*

***1.1.2*** Несмотря на высокие результаты, производительность *R-CNN* была всё же невысока, особенно для более глубоких, чем *CaffeNet* сетей (таких как *VGG16*). Кроме того, обучение *bounding* *box* *regressor* и *SVM* требовало сохранения на диск большого количества признаков, поэтому оно было дорогим с точки зрения размера хранилища.

Авторы *Fast* *R-CNN* предложили ускорить процесс за счёт пары модификаций. Пропускать через *CNN* не каждый из 2000 регионов-кандидатов по отдельности, а всё изображение целиком. Предложенные регионы потом накладываются на полученную общую карту признаков.

Вместо независимого обучения трёх моделей (*CNN*, *SVM*, *bbox* *regressor*) совместить все процедуры тренировки в одну.

Преобразование признаков, попавших в разные регионы, к фиксированному размеру производилось при помощи процедуры *RoIPooling*. Окно региона шириной *w* и высотой *h* делилось на сетку, имеющую *H×W* ячеек размером *h/H × w/W*. По каждой такой ячейке проводился *Max* *Pooling* для выбора только одного значения, давая таким образом результирующую матрицу признаков *H×W*.

Бинарные *SVM* не использовались, вместо этого выбранные признаки передавались на полносвязанный слой, а затем на два параллельных слоя: *softmax* с *K+1* выходами (по одному на каждый класс + 1 для фона) и *bounding* *box* *regressor*. Общая архитектура сети выглядит изображена на рисунке 1.2.

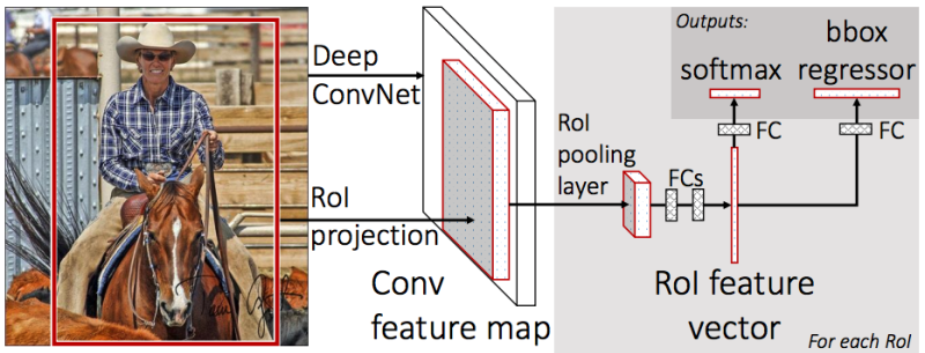


Рисунок 1.2 – Общая архитектура *Fast* *R-CNN*

***1.1.3*** *Faster R-CNN.* После улучшений, сделанных в *Fast R-CNN*, самым узким местом нейросети оказался механизм генерации регионов-кандидатов. В 2015 команда из *Microsoft* *Research* смогла сделать этот этап значительно более быстрым. Они предложили вычислять регионы не по изначальному изображению, а опять же по карте признаков, полученных из CNN. Для этого был добавлен модуль под названием *Region* *Proposal* *Network* (*RPN*). Новая архитектура целиком представлена на рисунке 1.3.

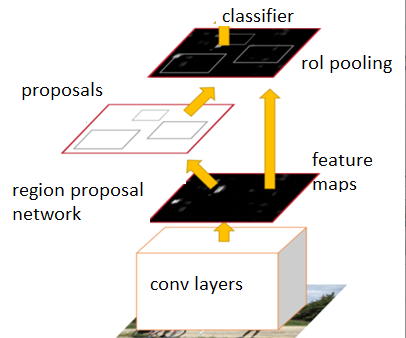


Рисунок 1.3 – Общая архитектура *Faster* *R-CNN*

В рамках *RPN* по извлечённым *CNN* признакам скользят «мини-нейросетью» с небольшим (3*х*3) окном. Полученные с её помощью значения передаются в два параллельных полносвязанных слоя: *box-regression layer (reg)* и *box-classification layer (cls).* Выходы этих слоёв базируются на так называемых *anchor*-ах – *k* рамках для каждого положения скользящего окна, имеющих разные размеры и соотношения сторон. *Reg*-слой для каждого такого *anchor*-а выдаёт по четыре координаты, корректирующие положение охватывающей рамки, *cls*-слой выдаёт по два числа – вероятности того, что рамка содержит хоть какой-то объект или что не содержит.

Для того, чтобы разделять признаки, получаемые в *CNN*, между *RPN* и модулем детектирования, процесс обучения всей сети построен итерационно, с использованием нескольких шагов:

* инициализируется и обучается на определение регионов-кандидатов *RPN*-часть;
* с использованием предлагаемых *RPN* регионов заново обучается *Fast* *R-CNN* часть;
* Обученная сеть детектирования используется, чтобы инициализировать веса для *RPN*. Общие *convolution*-слои, однако, фиксируются и производится донастройка только слоёв, специфичных для *RPN*.
* С зафиксированными *convolution*-слоями окончательно донастраивается *Fast R-CNN.*

***1.1.4***Алгоритм *YOLO* (*You Look Only Once*), предложенный в 2016 году, был первой попыткой сделать возможной детекцию объектов в реальном времени. В рамках алгоритма *YOLO* исходное изображение сначала разбивается на сетку из ячеек. Если центр объекта попадает внутрь координат ячейки, то эта ячейка считается ответственной за определение параметров местонахождения объекта. Каждая ячейка описывает несколько вариантов местоположения ограничивающих рамок для одного и того же объекта. Каждый из этих вариантов характеризуется пятью значениями – координатами центра ограничивающей рамки, его шириной и высотой, а также степени уверенности в том, что ограничивающая рамка содержит в себе объект. Также необходимо для каждой пары класса объектов и ячейки определить вероятность того, что ячейка содержит в себе объект этого класса.

Алгоритм *YOLO* работает быстрее алгоритмов семейства *R-CNN* за счёт того, что поддерживает дробление на константное количество ячеек вместо того, чтобы предлагать регионы и рассчитывать решение для каждого региона отдельно, однако, в качестве проблем *YOLO* указывается плохое качество распознавания объектов сложной формы или группы небольших объектов из-за ограниченного числа кандидатов для ограничивающих рамок.

Принцип работы *YOLO* представлен на рисунке 1.4.

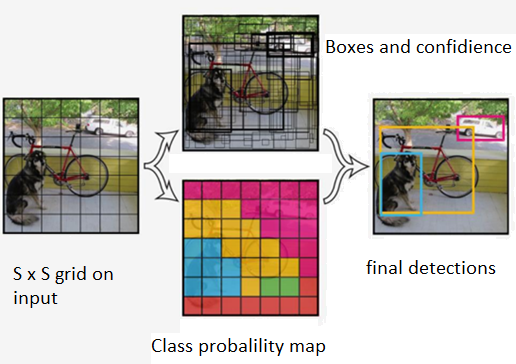


Рисунок 1.4 – Принцип работы *YOLO*

В качестве алгоритма нейронной сети выбран алгоритм *YOLO* так он имеет существенные преимущества перед *R-CNN* и *Fast R-CNN*, учитывая, что данная задача не требует работы в реальном времени, алгоритм *YOLO* позволяет более точно обнаруживать объекты в сравнении с *Fast R-CNN* и в реальном времени.

## **1.2 Задача обнаружения объектов на изображении**

Задача нахождения объектов на изображении – задача машинного обучения, в рамках которой выполняется определение наличия или отсутствия объекта определённого домена на изображении, нахождение границ этого объекта в системе координат пикселей исходного изображения. В зависимости от алгоритма обучения, объект может характеризоваться координатами ограничивающей рамки, ключевыми точками, контуром объекта [3, c.328].

Задача нахождения объектов на изображении может быть поставлена различным образом и включает в себя класс других задач, помогающих определить, какие объекты находятся на изображении и где они расположены в сетке пикселей исходного изображения.

Задача семантической сегментации – задача, в которой на вход модели подаётся изображение, а на выходе для каждого пикселя является метка принадлежности этого пикселя к определённой категории. Например, если в исходном изображении человек переходит дорогу, то для каждого пикселя необходимо вывести, является ли этот пиксель частью человеческого тела, профиля дороги, знака дорожного движения, неба, или какого-то другого типа. Существенный недостаток применения одной лишь семантической сегментации относительно задач, связанных с распознаванием объектов – маркировка пикселей по принадлежности только к типу объекта, что не создаёт различия между объектами как таковыми. Например, если назвать «объектом» связную область пикселей, характеризующих одинаковый тип, то два объекта, перегораживающих друг друга на исходном изображении, будут определены как один объект, что в корне неверно. Задача семантической сегментации изображения с дифференцированием объектов называется задачей сегментации экземпляров. Модели, решающие задачу сегментации экземпляров, применяются, в том числе, для подсчёта людей в массовых скоплениях, для автомобилей с автоматическим управлением.

Задача классификации с локализацией – задача, в которой в дополнение к предсказанию метки категории класса определяется рамка, ограничивающая местоположение экземпляра одиночного объекта на картинке. Как правило, рамка имеет прямоугольную форму, её стороны ориентированы параллельно осям исходного изображения, а площадь является минимальной при условии полного нахождения экземпляра объекта внутри этой рамки. Такую прямоугольную рамку называют термином «ограничивающая рамка». Ограничивающую рамку можно задать как при помощи центра, ширины и высоты, так и при помощи четырёх сторон. Модель в данном случается одновременно обучается как верной классификации, так и максимально точному определению границ рамки.

Задача детекции объектов – задача, в рамках которой необходимо выделить несколько объектов на изображении посредством нахождения координат их ограничивающих рамок и классификации этих ограничивающих рамок из множества заранее известных классов. В отличие от классификации с локализацией, число объектов, которые находятся на изображении, заведомо неизвестно.

Для уменьшения количества рассматриваемых ограничивающих рамок выделяют два основных параллельно развивающихся подхода.

Двухэтапные методы, они же «методы, основанные на регионах» – подход, разделённый на два этапа. На первом этапе селективным поиском или с помощью специального слоя нейронной сети выделяются регионы интереса – области, с высокой вероятностью содержащие внутри себя объекты. На втором этапе выбранные регионы рассматриваются классификатором для определения принадлежности исходным классам и регрессором, уточняющим местоположение ограничивающих рамок;

Одноэтапные методы – подход, не использующий отдельный алгоритм для генерации регионов, вместо этого предсказывая координаты определённого количества ограничивающих рамок с различными характеристиками, такими, как результаты классификации и степень уверенности и в дальнейшем корректируя местоположение рамок.

## **1.3 Средства разработки программного обеспечения**

*Python* и *C*# являются высокоуровневыми объектно-ориентированными языками программирования. Их легко изучать, они обеспечивают быструю разработку и отличную производительность. *Python* широко используется для веб-разработки, разработки программного обеспечения, анализа данных, визуализации данных и автоматизации задач. С другой стороны, *C*# – популярный выбор для создания веб-приложений, настольных приложений и веб-сервисов [4].

Свой синтаксис *C#* во многом унаследовал от *C*++ и *Java*. Разработчики, имеющие опыт написания приложений на этих языках, найдут в C# много знакомых черт. Но в то же время он в значительной степени инновационный, благодаря атрибутам, делегатам, событиям и другим возможностям, которые позволили использовать новые подходы в программировании. Принципиально важным отличием от своих предшественников является изначальная ориентация на безопасность кода за счет наличия среды *CLR*, при неуправляемом программировании программисты зачастую забывают освободить память, ставшую ненужной, или пытаются использовать уже освобожденную, что приводит к самым опасным и непредвиденным ошибкам в приложении [5].

Переняв многое от своих предшественников, опираясь на практику их использования, C# исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++, не поддерживает множественное наследование классов (между тем допускается множественная реализация интерфейсов) [6, c.502].

Преимущества C#:

– *C*# – простой, надежный и масштабируемый язык программирования;

– динамически типизированный характер *C*# облегчает разработчикам поиск ошибок в коде.

– *C*# устраняет проблему утечки памяти.

Недостатки *C*#:

– существование ряда ограничений при разработке для операционных систем отличных от *Windows*;

– *C*# не такой гибкий, как другие языки программирования, так как он зависит от платформы .*NET*.

*Python* – это универсальный интерпретируемый язык высокого уровня. Стиль *Python* – значительные отступы, которые подчеркивают удобочитаемость кода. Он следует нескольким принципам программирования, таким как объектно-ориентированное, функциональное, структурированное, рефлексивное и процедурное. *Python* включает обширную стандартную библиотеку, которая значительно упрощает разработку программ.

*Python* хорошо известен своей читабельностью кода. Кроме того, *Python* легко изучить и понять, поскольку в его синтаксисе используются простые английские ключевые слова и не используются фигурные скобки для разделения блоков.

Преимущества *Python*:

– *Python* – это язык с динамической типизацией. Это означает, что нет необходимости определять тип данных переменной, поскольку она автоматически присваивает типы данных переменным во время выполнения;

– *Python* легко читать и изучать благодаря синтаксису, похожему на английский. Кроме того, исключается использование точки с запятой после конца оператора и разделителей для начала и конца блока;

– *Python* является интерпретируемым языком, он выполняет код построчно, останавливает выполнение в случае ошибки и сообщает об этом.

– *Python* совместим и переносится в системы *Windows*, *macOS* и *Unix* / *Linux*;

* Имеет широкий спектр возможностей по работе с нейронными сетями.

Недостатки *Python*:

– *Python* имеет низкую скорость, потому что это интерпретируемый язык, и он выполняет код построчно;

– Это не идеальный выбор для задач с интенсивным использованием памяти, поскольку он потребляет большой объем памяти из-за гибкости типов данных;

– Поскольку *Python* неэффективен с точки зрения памяти и имеет медленную вычислительную мощность, он не используется при разработке клиентских или мобильных приложений.

Так как фреймворки *С*# и *Python* для работы с нейронными сетями сопоставимы по производительности ипредоставляют широкий спектр возможностей по их реализации, язык программирования C# и будет использоваться для создания приложения, способного обрабатывать данные в реальном времени локально или отправлять их на сервер.

# **2 АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА ПО ФОРМИРОВАНИЮ ЗАДАЧ ДЛЯ ОБНАРУЖЕНИЯ ОБЪЕКТОВ**

## **2.1 Принцип взаимодействия приложения по обнаружению объектов с сервером**

В качестве механизма сетевого взаимодействия выбрана архитектура «Клиент-Сервер» (в качестве клиента может выступать как клиентское приложение конечного пользователя, так и другой сервер позволяя, например, производить обнаружение объектов в распределенном виде).

Такая модель выбрана ввиду своей простоты и того факта, что в данной задаче инициатором любого обмена данными выступает клиент, следовательно, серверу нет необходимости хранить отдельный поток для взаимодействия с клиентом, в тоже время для оптимизации взаимодействия клиента с сервером предполагается использование постоянных *HTTP*-соединений (с помощью заголовка *Connection* и значения *Keep-Alive*).

Клиент и сервер обмениваются сообщениями в шаблоне запрос-ответ. Клиент отправляет запрос по необходимому пути, в свою очередь на сервере имеется система путей, отвечающая за выбор конечной точки которая будет заниматься обработкой запроса и формированием ответа. Чтобы формализовать обмен данными, сервер реализует интерфейс прикладного программирования (*API*). *API* – это уровень абстракции для доступа к сервису. Ограничивая связь определённым форматом контента, данный подход облегчает межплатформенный обмен данными.

Архитектура программного обеспечения по формированию задач для обработки данных локально или на сервере представлена на рисунке 2.1.

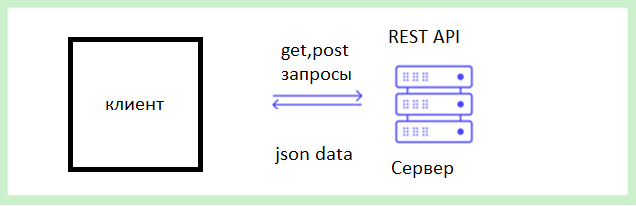


Рисунок 2.1 – Архитектура программного обеспечения по формированию задач для обработки данных локально или на сервере

Преимущества, которые предлагает данная архитектура:

– системы, реализующие *REST API*, могут эффективно масштабироваться благодаря оптимизации взаимодействия между сервером и клиентом по *REST*, отсутствие сохранения состояния снимает нагрузку с сервера: серверу не нужно сохранять информацию о предыдущих запросах клиента;

– веб-службы *REST* поддерживают полное разделение клиента и сервера, упрощают и разделяют различные серверные компоненты, чтобы каждая часть могла развиваться независимо, изменения платформы или технологии в серверном приложении не влияют на клиентское приложение;

– *REST API* не зависит от используемой технологии, можно создавать как клиентские, так и серверные приложения на разных языках программирования, не затрагивая структуру *API*. Также можно изменить базовую технологию на любой стороне, не влияя на обмен данными.

Взаимодействие клиента с сервером представлено следующими этапами:

– клиент связывается с сервером с помощью протокола *HTTP*, руководствуясь документацией *API*, клиент форматирует запрос таким образом, чтобы его понимал сервер;

– сервер аутентифицирует клиента и подтверждает, что клиент имеет право сделать этот запрос;

– сервер получает запрос и обрабатывает его;

– сервер возвращает ответ клиенту, ответ содержит информацию, которая сообщает клиенту, был ли запрос успешным, также ответ включает сведения, запрошенные клиентом.

## **2.2 Архитектура *YOLOv5***

*YOLO*  – архитектура нейронных сетей, предназначенная для детекции объектов на изображении. Отличительной особенностью *YOLO* является подход к решению задачи детекции.

Один из способов решения задачи детекции заключается в разбиении изображения на квадратные области, затем классификация этих областей на наличие объекта и классификация самого объекта. Таким образом, изображение просматривается дважды (один раз для определения областей, где есть объект, второй – для классификации этого объекта.) Этот способ работает долго и требует больших затрат вычислительных мощностей.

*YOLO* же использует другой принцип. Исходное изображение сжимается таким образом, чтобы получить квадратную матрицу размером 13 на 13, в каждой клетке которой записана информация о наличии объекта и классе этого объекта на соответствующей части картинки. Таким образом, *YOLO* просматривает картинку один раз, что существенно увеличивает скорость обработки.

*YOLOv5* – усовершенствованная пятая версия *YOLO*, реализованная на фреймворке *PyTorch*.

Архитектура *YOLOv5* предствалена на рисунке 2.2.

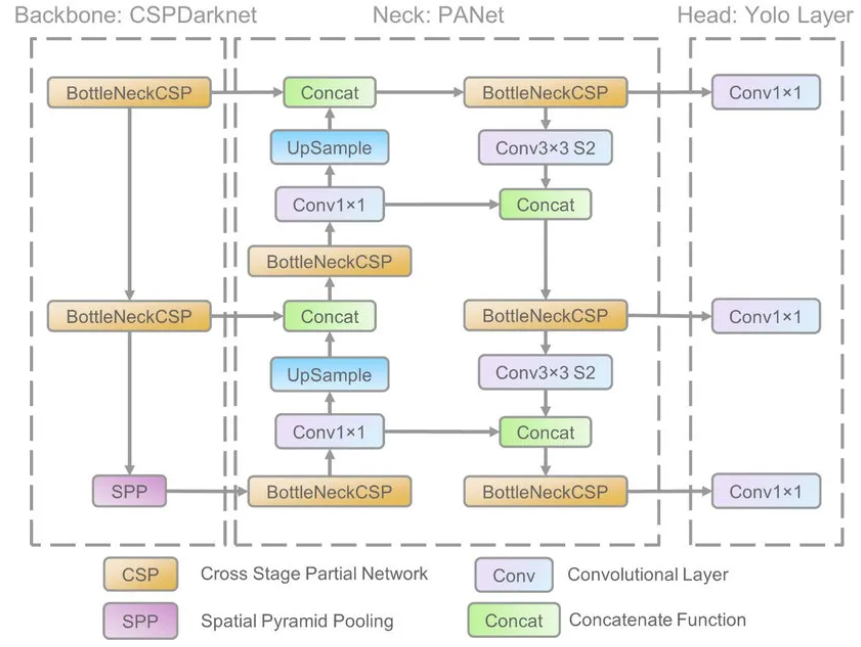


Рисунок 2.2 – Архитектура *YOLOv5*

Семейство моделей YOLO состоит из трех основных архитектурных блоков:

1. *Backbone* – использует *CSPDarknet* в качестве основы для извлечения объектов из изображений, состоящих из многоступенчатых частичных сетей;

2. *Neck* – использует *PANet* для создания сети пирамид объектов для выполнения агрегации объектов и нее передачи в *Head* для прогнозирования;

3. *Head* – cлои, которые генерируют прогнозы по блокам, полученным с помощью *Neck*.

*YOLOv5* имеет множество разновидностей предварительно обученных моделей. Разница между ними заключается в компромиссе между размером модели и временем распознавания. Облегченная версия модели *YOLOv5s* имеет размер всего 14 МБ, но не очень точна. С другой стороны, есть *YOLOv5x*, размер которого составляет 168 МБ, но данный вариант является наиболее точной версией из всех разновидностей.

Существует пять разновидностей:

*– n –* модель очень маленького размера;

*– s –* модель небольшого размера;

*– m* – модель среднего размера;

*– l –* модель крупного размера;

*– x –* модель очень большого размера по сравнению с другими разновидностями.

Между пятью моделями нет разницы с точки зрения используемых операций, за исключением количества слоев и параметров, как показано на рисунке 2.3.

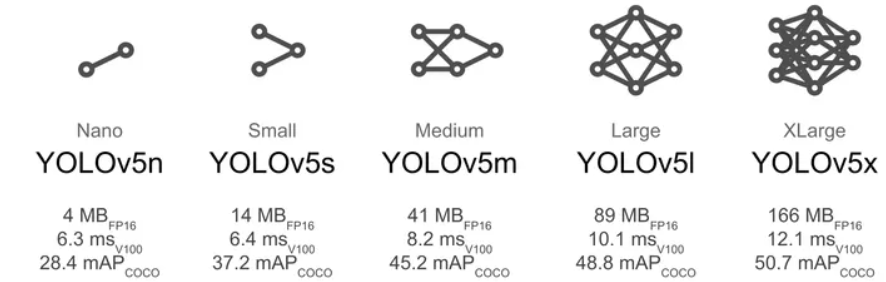


Рисунок 2.3 – Разновидности *YOLOv5*

Выходным результатом *YOLOv5* является двумерный массив размер которого зависит входного массива. Например, при вводимом по умолчанию размере изображения 640 и количестве классов 2 получается двумерный массив размером 25200 на 7 (строки и столбцы). Строки представляют количество обнаружений. Таким образом, каждый раз, когда сеть запускается, получается 25200 ограничивающих рамок. Каждая ограничивающая рамка содержит одномерный массив из семи записей в случае двух классов (количество классов, прибавленное к элементам, хранящим информацию об обнаруженном объекте), который определяет качество обнаружения. Этой информации достаточно, чтобы отфильтровать желаемые обнаружения [7].

Первые два элемента – это нормализованные координаты центра обнаруженной ограничивающей рамки. Затем идут нормализованные ширина и высота. Индекс четыре имеет показывает вероятность того, что обнаруженный объект является объектом. Следующие две записи показывают оценки для каждого класса набора данных.

* 1. **Набор данных для обучения нейронной сети**

Предметной областью являются огурцы. Для реализации приложения подготовлен набор изображений огурцов включая другие овощи и фрукты.

Набор данных представлен двумя классами:

– *cucumber* – представляет огурцы;

– *trash* – представляет любой другой овощ либо фрукт чтобы позволить сети узнавать объекты отличные от огурца и, следовательно, уменьшить вероятность того что неизвестный объект будет принят за огурец.

Пример размеченных данных представлен на рисунке 2.4.

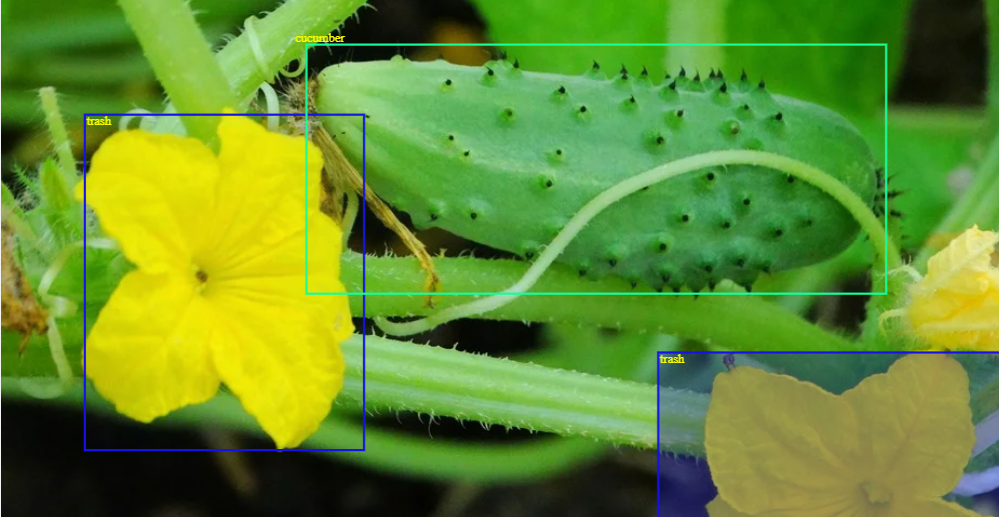
**

Рисунок 2.4 – Пример размеченных данных

Формат набора данных: *YOLO txt* и представляет собой список текстовых файлов, имеющих название изображения и содержащих список отмеченных объектов с указанием номера класса. Формат текстовых файлов представлен на рисунке 2.5.

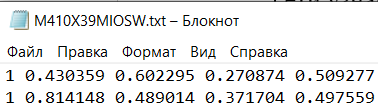


Рисунок 2.5 – Формат текстовых файлов набора данных

* 1. **Обучение модели *YOLOv5***

Для обучения *YOLOv5* выбраны разновидности *nano* и *small* которые показывают наибольшую производительность в нахождении объектов.

Для обучения *YOLOv5* необходимо:

– библиотека *YOLOv5* и вспомогательные *Python* пакеты необходимые для работы с данной библиотекой*;*

– предобученные веса для необходимой разновидности *YOLOv5*;

– файл *yaml* (в нем указываются пути до тренировочной и валидационной выборок, а также количество классов и их метки);

– набор данных.

Основные *Python* пакеты необходимые для работы с библиотекой *YOLOv5*:

– *matplotlib* – используется для создания графиков с информацией о процессе обучения;

– *numpy* – используется для работы с матрицами;

– *opencv*-*python* – используется для работы с изображениями;

– *torch* и *torchvision* – используются для реализации модели.

*yaml* файл необходимый для обучения модели представлен на рисунке 2.6.

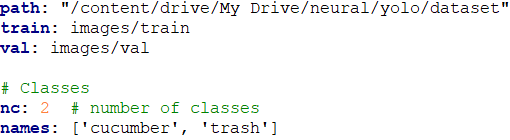


Рисунок 2.6 – *yaml* файл необходимый для обучения модели

В таблице 2.1 находятся результаты обучения модели при разных настройках.

Таблица 2.1 – Результаты обучения модели при разных настройках

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Разновидность | Количество эпох | Размер  подвыборки | *Precision* | *Mean average*  *precision(mAP)* |
| *nano* | 200 | 32 | 0.889 | 0.931 |
| *nano* | 200 | 64 | 0.866 | 0.91 |
| *small* | 200 | 16 | 0.914 | 0.933 |
| *small* | 200 | 32 | 0.95 | 0.956 |

В результате обучения выявлены оптимальные значения гиперпараметров обучения таких как: скорость обучения, *shear*, *scale*, *momentum* и другие. Обучение состояло из 200 эпох с 11 шагами в рамках эпохи. Набор данных был разделен в пропорциях 80 к 20%. В результате обучения модели *YOLOv5* получена точность в 95% на валидационной выборке, а значение ошибки поиска – 5% при использовании разновидности сети *small*. При обучении разновидности сети *nano* получена точность в пределах 88% что значительно хуже в сравнении с разновидностью сети *small*. В качестве функции потерь использовалась бинарная кросс-энтропия.

Графики зависимости точности и функции потерь от количества эпох изображен на рисунке 2.7

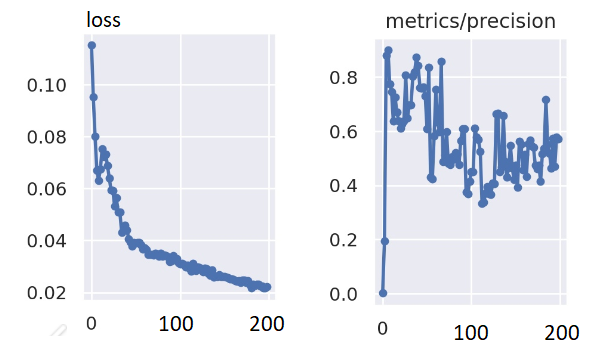


Рисунок 2.7 – Графики зависимости точности и функции потерь от количества эпох

В результате обучения наиболее хорошие результаты получены с использованием разновидности *small* и размера подвыборки 32. Веса из данного обучения и в следствие и использованы в приложении.

## **2.5 Структура программного комплекса по обнаружению объектов**

Клиентское приложение программного комплекса состоит консольного приложения, а также вспомогательных сервисов, отвечающих за взаимодействие с сервером и уровня бизнес-логики.

Структура клиентского приложения представлена на рисунке 2.8.

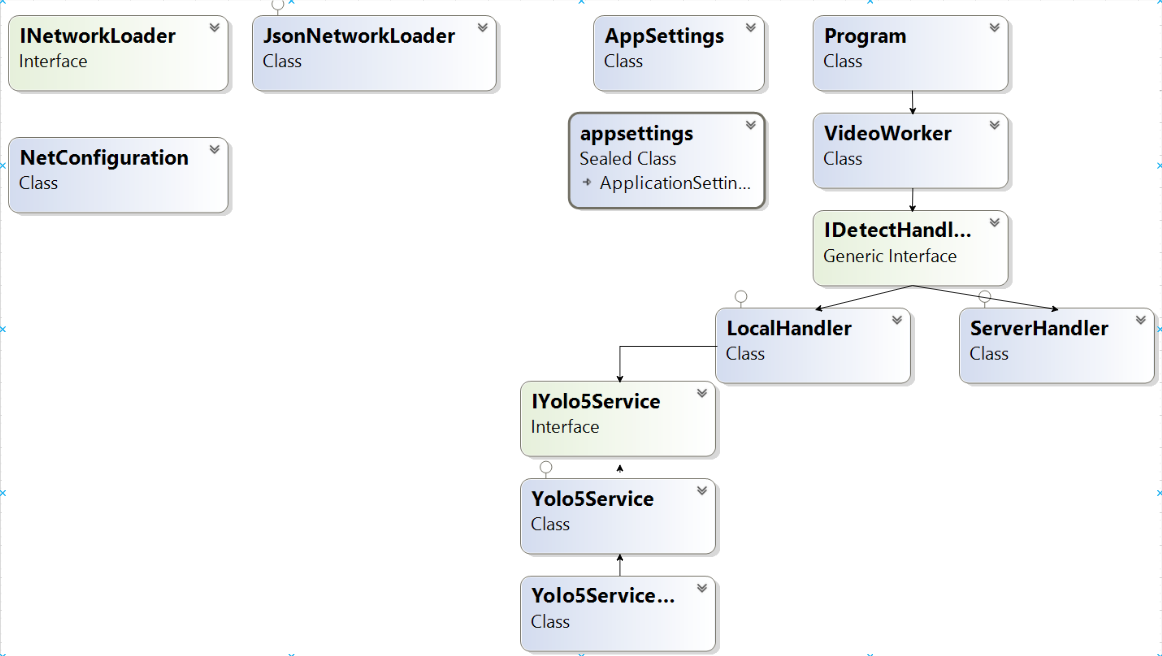


Рисунок 2.8 – Структура клиентского приложения

Главным компонентом клиентского приложения является веб-камера*,* на которой происходит обнаружение объектов. Листинг кода представлен в приложении А.

Уровень бизнес-логики – отвечает за работу по обнаружению объектов локально, если сервер не отвечает или если время ответа от сервера больше заданного промежутка времени, на входных изображениях с использованием нейронной сети.

Главным компонентом уровня бизнес-логики является *YoloDetector* с которым и происходит взаимодействие с верхнего уровня посредством реализации интерфейса *IYoloDetector*, данный компонент использует предоставленную модель с заданными параметрами, реализующую интерфейс *IYoloModel* для произведения обнаружения объектов и создания моделей *YoloDetection*.

Основные компоненты уровня бизнес-логики и их взаимосвязь показаны на рисунке 2.9.

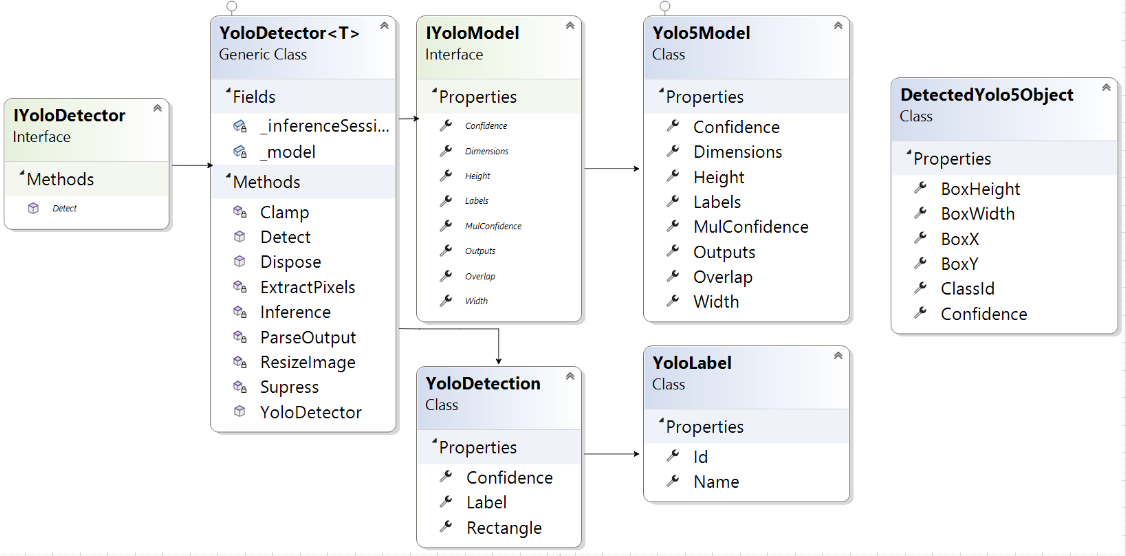


Рисунок 2.9 – Основные компоненты уровня бизнес-логики и их взаимосвязь

Полная схема взаимодействия классов программного комплекса представлена в приложении Б.

Для передачи результатов обнаружения из набора моделей *YoloDetection* клиентам используется *DetectedYolo5Object,* который имеет более упрощенный вид по сравнению с *YoloDetection*. Это объект, определяющий способ отправки данных по сети, который является *DTO (data transfer object)*.

.

**3 СТРУКТУРА И ВЕРИФИКАЦИЯ ПРИЛОЖЕНИЯ ПО ФОРМИРОВАНИЮ ЗАДАЧ ДЛЯ ОБНАРУЖЕНИЯ ОБЪЕКТОВ**

**3.1 Структура классов и интерфейсов программного комплекса по формированию задач для обнаружения объектов**

Программный комплекс решающий поставленную задачу представлен одним приложением: клиент, реализованный с использованием языка программирования *С#*.

При реализации приложения использовались классы:

* *OpenCvSharp –* для обработки кадров с веб-камеры;
* *HttpClient –* для отправки полученных кадров с веб-камеры на сервер в реальном времени;
* *ML.NET –* для работы нейронной сети локально, если присутствуют проблемы на стороне сервера.

Клиент подключается к серверу по заданному адресу: *https://*{*ip adress*}*/v1/yolo5/{variation}/json*

Для работы с нейронными сетями используется библиотека *ML.NET* которая позволяет использовать модель, обученную на *Python* и экспортированную в *ONNX (Open Neural Network Exchange)* формате.

В качестве опции клиент имеет параметр строки запрос *threshold* позволяющий игнорировать объекты, которые имеют точность определения объекта ниже заданного значения.

Объект возвращаемый при обращении по пути */v1/yolo5/{variation}/json* представлен классом *DetectedYolo5Object* и содержит следующие свойства:

– *ClassId* – номер класса к которому принадлежит объект;

– *Confidence* – вероятность принадлежности объекта к указанному в поле *ClassId* классу;

– *BoxX* – минимальный *x* ограничивающей рамки, содержащей объект;

– *BoxY* – минимальный *y* ограничивающей рамки, содержащей объект;

– *BoxWidth* – ширина ограничивающей рамки, содержащей объект;

– *BoxHeight* – высота ограничивающей рамки, содержащей объект.

Класс *DetectedYolo5Object* является трансфертным объектомданных содержащихся в классе *YoloDetection*, данный класс используется на уровне бизнес-логики и создается классом *YoloDetector* при обнаружении объектов на изображении и имеет следующие свойства:

– *Label* – относится к классу *YoloLabel* и хранит информацию о классе объекта такую как название и номер;

– *Confidence* – вероятность принадлежности объекта к указанному в поле *ClassId* классу;

– Rectangle – относится к встроенному классу *ReactangleF* и хранит подробную информацию об ограничивающей рамке объекта.

Пример *JSON* объекта, получаемого клиентом при обращении к серверу по пути */v1/yolo5/{variation}/json* представлен на рисунке 3.1.

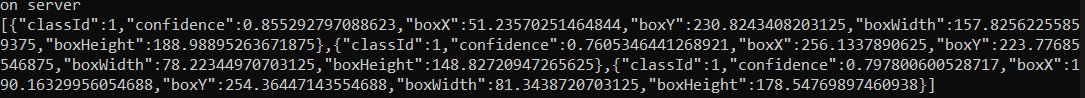


Рисунок 3.1 – Пример получаемого *JSON* объекта при обращении к серверу

Пример изображения, полученного с веб-камеры после обработки ответа с сервера, представлен на рисунке 3.2.



Рисунок 3.2 – Пример изображения, полученного с веб-камеры после обработки ответа с сервера

Для конфигурирования параметров модели таких как размер, количество классов, используется загрузка конфигурации моделей из *JSON* файла который читается во время запуска с помощью класса *JsonNetworkLoader* который реализует интерфейс *INetworkLoader*,что позволит в будущем иметь возможность загрузки конфигурации, например, по сети либо из файла другого формата, данный *JSON* файл содержит следующие поля:

* *Name* – название модели, по этому названию модель идентифицируется при обращении клиента;
* *Path* – путь файлу с весами модели, данный файл читается также при запуске для инициализации модели и при обращении клиента происходит только операция обнаружения;
* *Model –* конфигурация, отражающая обученную модель, для представления данного поля имеется класс *Yolo5Model* который содержит следующие свойства:

1. *Width* – ширина входного изображения;
2. *Height* – высота входного изображения;
3. *Dimensions* – количество элементов в строке выходного двумерного массива (количество классов, прибавленное к элементам, хранящим информацию об обнаруженном объекте);
4. *Confidence* – обнаруженные объекты игнорируются если имеют уверенность ниже данного значения;
5. *MulConfidence* – обнаруженные объекты игнорируются если имеют уверенность, умноженную на уверенность того, что представленный объект является объектом ниже данного значения;
6. *Overlap* – допустимое значение пересечения обнаруженных объектов, в случае пересечения объекты с наименьшим значение уверенности игнорируются;
7. *Outputs* – названия выходных слоев (для *YOLOv5* один выходной слой с названием *output0*);
8. *Labels* – массив класса *YoloLabel* представляющего информацию о классе набора данных.

Пример *JSON*-файла конфигурации моделей нейронных сетей показан на рисунке 3.3.



Рисунок 3.3 – Пример *JSON*-файла конфигурации моделей нейронных сетей

Для локальной обработки данных с веб-камеры в приложении имеется сервис *Yolo5Service*,который отвечает за выбор нужной модели и вызов *YoloDetector,* класса бизнес-логики, чтобы получить список объектов, находящихся на изображении локально и в реальном времени.

## **3.2 Нагрузочное тестирование клиентского приложения по обнаружению плодов**

При тестировании приложения были рассмотрены следующие проблемы:

* сервер не отвечает;
* время ответа от сервера больше заданного значения в приложении.

Если сервер не отвечает в течении определенного количества обращений к нему или время ответа от сервера больше заданного, то изображения, полученные с веб-камеры, обрабатываются локально при помощи нейронной сети, развернутой на клиенте.

Уведомление о локальной обработке данных показано на рисунке 3.4.

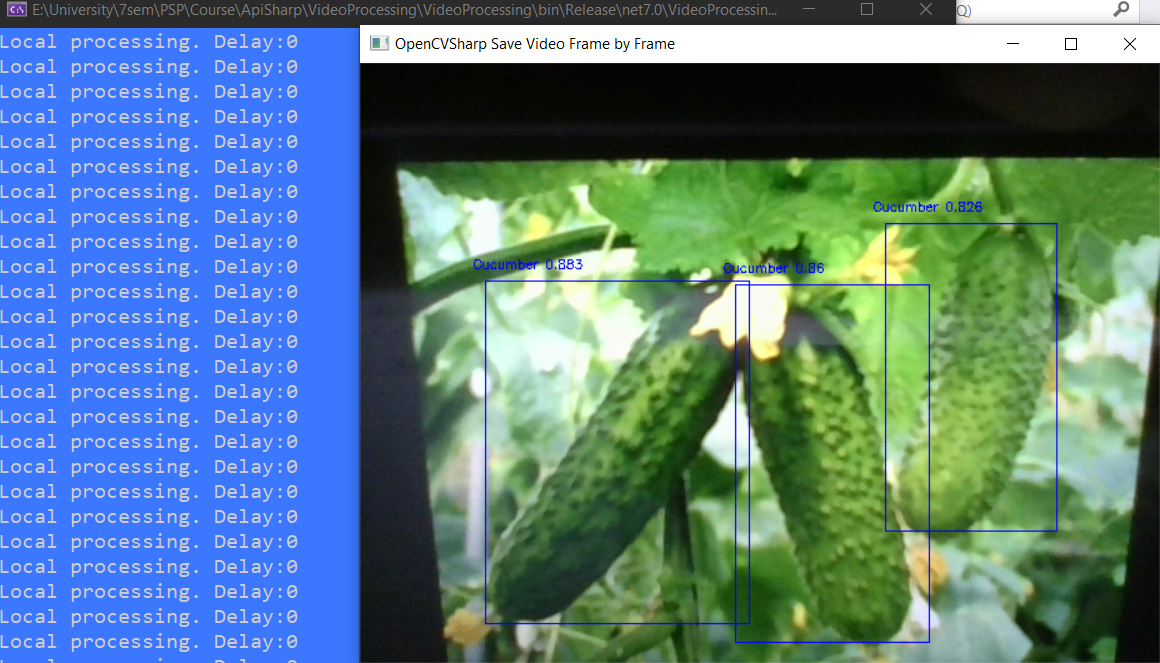


Рисунок 3.4 – Уведомление о локальной обработке данных

Если сервер в рабочем состоянии и время ответа от сервера меньше заданного количества времени, то данные, полученные с веб-камеры, отправляются на сервер и там обрабатываются.

Уведомление о серверной обработке данных показано на рисунке 3.5.

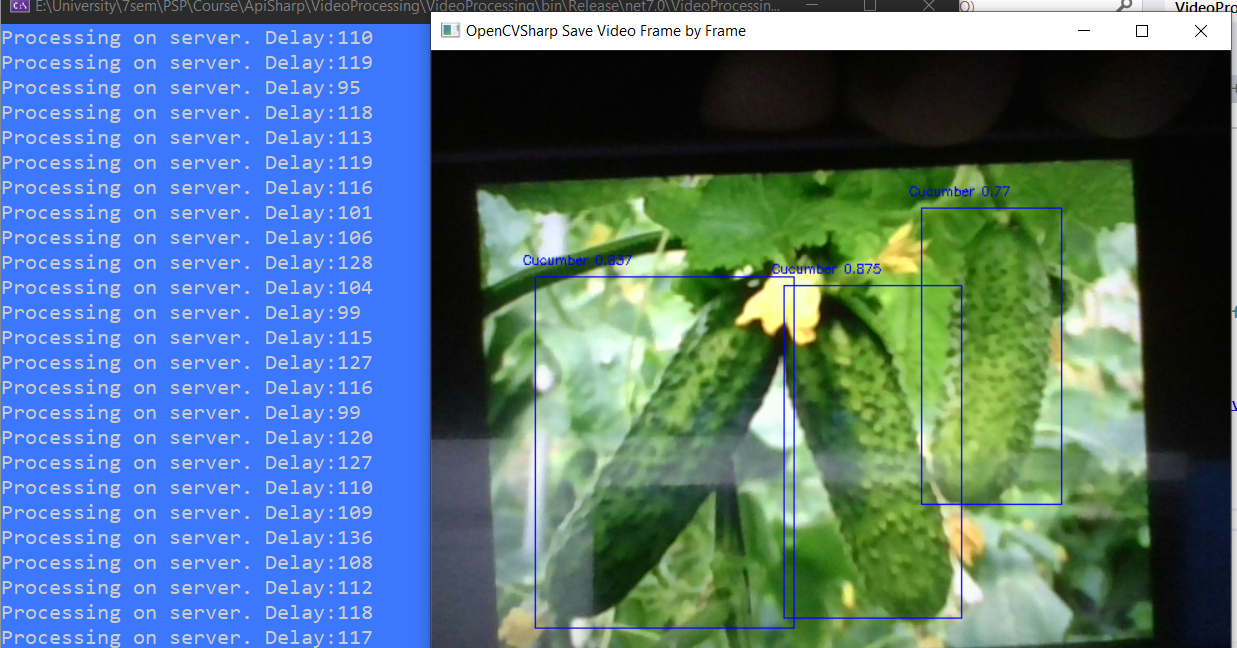


Рисунок 3.5 – Уведомление о серверной обработке данных

В приложении обработаны критические ситуации:

* если во время работы приложения «упал сервер», то происходит переход на локальную обработку данных и через определенный промежуток времени будет отправлен новый запрос на сервер и если сервер доступен, то вычисления будут происходить не локально, а на нем.
* Если во время работы приложения, сервер отвечает больше заданного промежутка времени, то вычисления буду происходить локально до тех пор, пока время получения ответа не будет меньше заданного промежутка времени.

Уведомление о восстановленной обработке данных на сервере показано на рисунке 3.6.

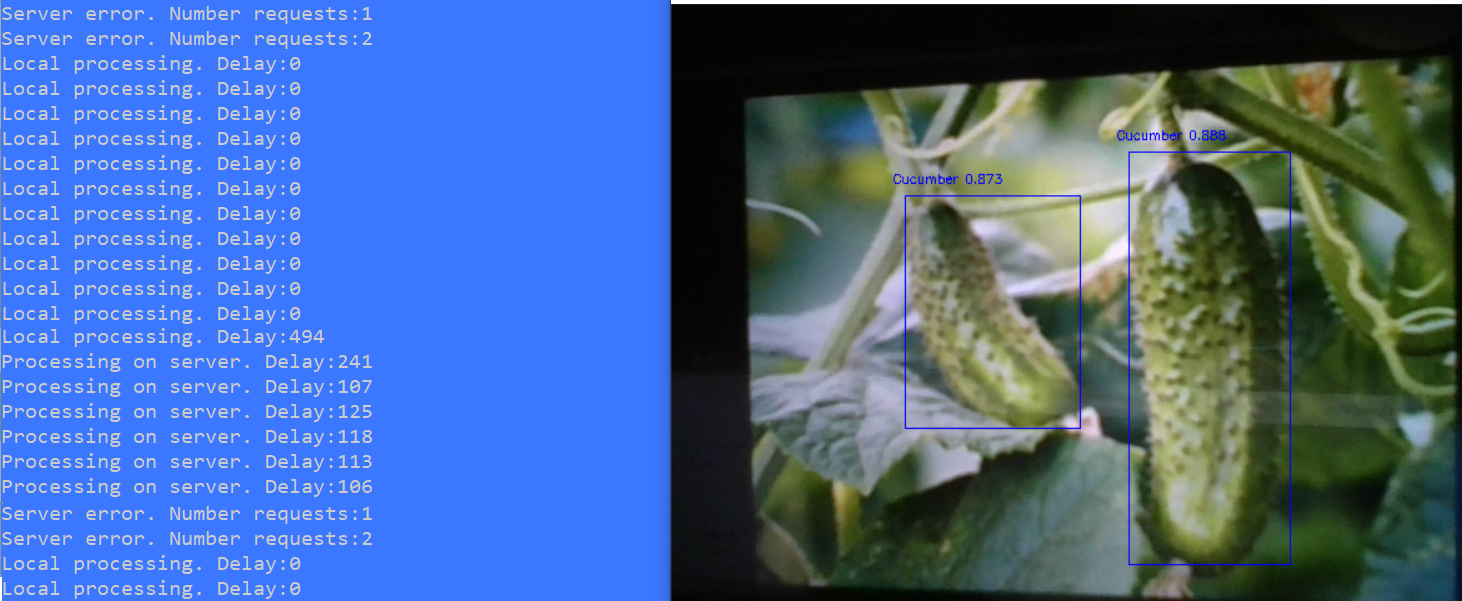


Рисунок 3.6 – Уведомление о восстановленной обработке данных на сервере

Были проведены сравнения обработки данных локально на устройстве и удаленно на сервере.

Результаты, полученные при обработке данных локально на устройстве и удаленно на сервере, изображены на рисунке 3.7

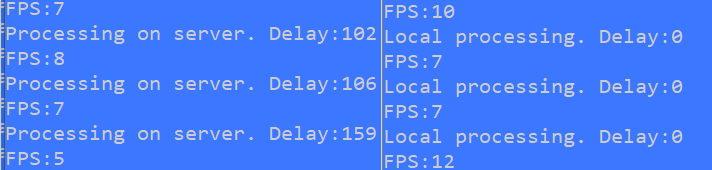


Рисунок 3.7 – Результат сравнения обработки данных локально на устройстве и удаленно на сервере

Среднее значение *fps (frame per second),* при обработке данных локально показало результат в 10-11 *fps,* при использовании программно-аппаратной архитектуры параллельных вычислений *CUDA (Compute Unified Device Architecture).*

Среднее значение *fps,* при обработке данных на сервере показало результат в шесть-семь *fps.*

Графики сравнения *fps,* при обработке данных на сервере и локально показаны на рисунке 3.8.

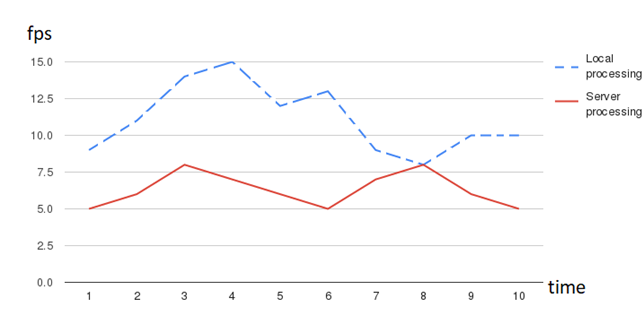


Рисунок 3.8 – Графики сравнения *fps,* при обработке данных на сервере и локально

Результат обработки данных показал, что вычисление, проходящее локально, не на много отличается от вычислений, проходящих на сервере. Данный результат вполне приемлемым, для обработки данных.

Для локальной обработки данных не требуются большие вычислительные мощности, но использование сервера с более мощным процессором или видеокартой поможет ускорить обработку данных, тем самым увеличив производительность локального устройства.

## **3.4 Верификация разработанного программного обеспечения**

Для верификации разработанного программного был выбран ноутбук с веб-камерой.

Результат работы приложения показан на рисунке 3.9.

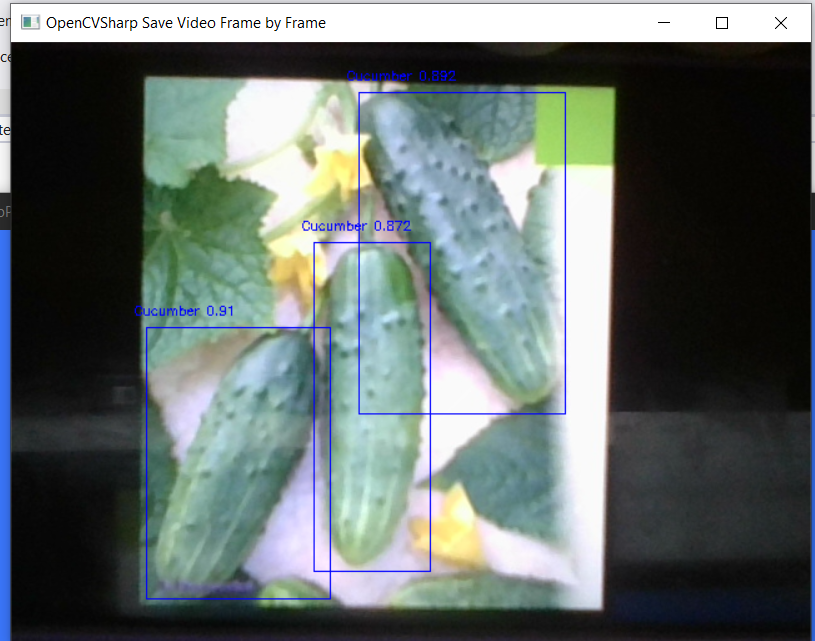


Рисунок 3.9 – Результат, полученный в ходе работы приложения

В качестве исходных данных были выбраны фотографии огурцов, которые показывались веб-камере.

В результате работы приложения на видеоизображении, получаемого с веб-камеры, отображались рамки объектов и вероятность принадлежность к классу «*cucumber*».

В ходе работы приложения была проверена обработка данных, как локально на клиенте, так и удаленно на сервере. В результате приложение корректно работает с данными, обработанными на сервере и локально, отображая рамки объектов, принадлежащих к классу «*cucumber*».

Руководства системного программиста, программиста и пользователя с более подробной информацией о работе с программным комплексом представлены в приложениях В, Г и Д соответственно.

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта изучена и проанализирована информация в области разработки программного обеспечения для решения задачи формирования задач по обнаружению объектов с использованием искусственных нейронных сетей в реальном времени.

В ходе разработки программного обеспечения решены следующие задачи:

– обучение нейронной сети в *Google* *colaab* и импорт полученных весов в приложение;

– разработка программного кода, производящего формировать задачи для обнаружения объектов с использованием обученной модели нейронной сети локально на устройстве или удаленно на сервере;

– проведение верификации обученной модели нейронной сети с использованием метрик качества работы нейронной сети*.*

Программный комплекс разработан на языках программирования *C#*.

Проводилась верификация обученной нейронной сети с использованием метрик *precision* и *mAP*, а также сравнение производительности работы фреймворков *PyTorch* и *ML.NET*, установлено, что *ML.NET* имеет производительность сопоставимую с *PyTorch*, в тоже время позволяя сделать делать локальные вычисления более производительным за счет использования языка программирования *C#*.

Также проводилось тестирование производительности приложения, по результатам которого можно сделать вывод, что приложение обеспечивает обработку изображений в реальном времени и показывает высокую производительность при обработке данных, как локально, так и удаленно на сервере.

Разработанный программный комплекс позволяет формировать задачи по обнаружению объектов в реальном времени и обрабатывать их, если есть соединение с интернетом, на сервере или если соединение отсутствует, производить обработку данных локально на роботе практически без потери производительности.

# **Список использованных источников**

1. Хайкин, С. Нейронные сети: Полный курс: Пер. с англ. / С. Хайкин. – М.: Вильямс, 2008. – 1103 с.

2. Саймон, Х. Нейронные сети. Полный курс: Пер. с англ. / Х. Саймон – СПб: Питер, 2018. – 1104 с.

3. Шолле, Ф. Глубоко обучение на *Python*: Пер. с англ. / Ф. Шолле – СПб: Питер, 2018. – 400 с.

4. *C*# против *Python*. – Электрон. данные. – Режим доступа: https://devdelphi.ru/blog/c-protiv-python-kakoi-variant-luchshe. – Дата доступа: 11.11.2022.

5. Сравнение *C Sharp и Java*: Свободная энциклопедия. – Электрон. данные. – Режим доступа: https://ru.wikipedia.org/wiki/Сравнение\_C\_Sharp\_и\_Java. – Дата доступа: 10.11.2022.

6. Рихтер, Дж. *CLR VIA C#*. Программирование на платформе *Microsoft* *.NET Framework* 4.5 на языке *C#* / Рихтер Дж. – 4-е изд., пер. с англ. – СПб. : Питер, 2019. – 896 с.

*7. YOLOv5* [Электронный ресурс]. – 2022. – Режим доступа: https://github.com/ultralytics/yolov5. – Дата доступа: 15.11.2022.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

using System;

using System.Diagnostics;

using VideoProcessing.Services;

namespace VideoProcessing

{

**internal class Program**

{

static void Main(string[] args)

{

const int allowedError = 2;

const int allowedDelay = 350;

const int globalRefreshTime = 30000;

var netLoader = new JsonNetworkLoader($"{AppContext.BaseDirectory}/Networks/networks.json");

var availableYoloNets = netLoader.LoadNets();

var globalTimer = new Stopwatch();

globalTimer.Start();

new VideoWorker(availableYoloNets).Work(allowedError,allowedDelay,globalRefreshTime, globalTimer);

}

}

}

using OpenCvSharp;

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Threading;

using System.Threading.Tasks;

using VideoProcessing.Handlers;

using Yolov5;

namespace VideoProcessing

{

**internal class VideoWorker**

{

public Dictionary<string, IYoloDetector> AvailableYoloNets { get; set; }

public VideoWorker(Dictionary<string, IYoloDetector> availableYoloNets)

{

this.AvailableYoloNets= availableYoloNets;

}

public Task Work(int allowedError, int allowedDelay, int globalRefreshTime, Stopwatch globalTimer)

{

var window = new Window("OpenCVSharp Save Video Frame by Frame");

var capture = new VideoCapture(0);

var image = new Mat();

int countError = 0;

var delay = 0;

List<DetectedYolo5Object> data = new();

var timer = new Stopwatch();

timer.Start();

while (capture.IsOpened())

{

capture.Read(image);

if (image.Empty())

break;

if (Cv2.WaitKey(1) == 113) //Q

break;

window.ShowImage(image);

try

{

if (countError < allowedError)

{

data = GetLocalOrServerData(delay, allowedDelay, image);

}

else

{

Console.WriteLine($"Local processing. Delay:{delay}");

data = new LocalHandler(AvailableYoloNets).GetData(image);

}

}

catch (Exception)

{

countError++;

Console.WriteLine($"Server error. Number requests:{countError}");

}

Console.WriteLine($"FPS:{1000 / (timer.ElapsedMilliseconds)}");

timer.Restart();

image = GetBoxes(data, image);

window.ShowImage(image);

if (globalTimer.ElapsedMilliseconds >= globalRefreshTime)

{

countError = 0;

globalTimer.Restart();

}

}

return Task.CompletedTask;

}

public static Mat GetBoxes(List<DetectedYolo5Object> data, Mat image)

{

foreach (var i in data)

{

HersheyFonts fontFace = HersheyFonts.HersheySimplex;

double scale = 1.0 / 3;

var confidence = Math.Round(i.Confidence, 3);

Cv2.Rectangle(image, new Point(i.BoxX, i.BoxY), new Point(i.BoxX + i.BoxWidth, i.BoxY + i.BoxHeight), 255, 1);

Cv2.PutText(image, "Cucumber " + confidence, new Point(i.BoxX - 10, i.BoxY - 10), fontFace, scale, 255);

}

return image;

}

public List<DetectedYolo5Object> GetLocalOrServerData(int delay, int allowedDelay, Mat image)

{

List<DetectedYolo5Object> result;

var sw = new Stopwatch();

sw.Start();

var checkResult = new ServerHandler().GetData(image);

sw.Stop();

delay = (int)sw.ElapsedMilliseconds;

if (delay < allowedDelay)

{

result = checkResult;

Console.WriteLine($"Processing on server. Delay:{delay}");

}

else

{

result = new LocalHandler(AvailableYoloNets).GetData(image);

Console.WriteLine($"Local processing. Delay:{delay}") ;

}

return result;

}

}

}

using System.Collections.Generic;

using Yolov5;

namespace VideoProcessing.Services

{

**public class Yolo5ServiceOptions**

{

public required Dictionary<string, IYoloDetector> AvailableNets { get; set; }

public Yolo5Service Yolo5Service

{

get => default;

set

{

}

}

}

}

using System.Collections.Generic;

using System.Drawing;

using Yolov5;

namespace VideoProcessing.Services

{

**public class Yolo5Service : IYolo5Service**

{

private readonly Yolo5ServiceOptions \_options;

public Yolo5Service(Yolo5ServiceOptions options)

{

\_options = options;

}

public IEnumerable<YoloDetection> Detect(string netName, Image image, float? threshold)

{

var net = \_options.AvailableNets[netName];

return net.Detect(image, threshold);

}

}

}

using Yolov5.Models;

namespace VideoProcessing.Services

{

**public class NetConfiguration**

{

public string Name { get; set; }

public string Path { get; set; }

public Yolo5Model Model { get; set; }

}

}

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using Yolov5.Models;

using Yolov5;

namespace VideoProcessing.Services

{

**public class JsonNetworkLoader : INetLoader**

{

private readonly string \_jsonPath;

public JsonNetworkLoader(string jsonPath)

{

\_jsonPath = jsonPath;

}

public Dictionary<string, IYoloDetector> LoadNets()

{

List<NetConfiguration>? nets = null;

try

{

nets = JsonConvert.DeserializeObject<List<NetConfiguration>>(File.ReadAllText(\_jsonPath));

if (nets == null)

{

new InvalidOperationException("Couldn't load nets configuration from JSON file");

}

}

catch (Exception ex)

{

throw new InvalidOperationException("Couldn't load nets configuration from JSON file", ex);

}

var availableYoloNets = nets.Select(nc =>

new KeyValuePair<string, IYoloDetector>(nc.Name, new YoloDetector<IYoloModel>(

nc.Model,

Path.IsPathFullyQualified(nc.Path) ? nc.Path : $"{AppContext.BaseDirectory}/{nc.Path}"))).ToDictionary(p => p.Key, p => p.Value);

return availableYoloNets;

}

}

}

using System.Collections.Generic;

using Yolov5;

namespace VideoProcessing.Services

{

public interface INetworkLoader

{

Dictionary<string, IYoloDetector> LoadNets();

}

}

using Newtonsoft.Json;

using OpenCvSharp;

using System;

using System.Collections.Generic;

using System.Globalization;

using System.IO;

using System.Net.Http;

using System.Threading.Tasks;

using Yolov5;

namespace VideoProcessing.Handlers

{

**internal class ServerHandler : IDetectHandler<DetectedYolo5Object>**

{

private readonly string connectionString = "http://192.168.0.103:5059/v1/yolo5/nano/json";

public ServerHandler()

{

}

public List<DetectedYolo5Object> GetData(Mat image)

{

var result = SendData(image).Result;

return JsonConvert.DeserializeObject<List<DetectedYolo5Object>>(result);

}

public async Task<string> SendData(Mat image)

{

var client = new HttpClient();

var content = new MultipartFormDataContent("Upload----" + DateTime.Now.ToString(CultureInfo.InvariantCulture))

{

{ new StreamContent(new MemoryStream(image.ToBytes())), "file", "my.jpg" }

};

var requestMessage = new HttpRequestMessage(HttpMethod.Post, $"{connectionString}")

{

Content = content

};

var responseMessage = await client.SendAsync(requestMessage);

var output = await responseMessage.Content.ReadAsStringAsync();

return output;

}

}

}

using OpenCvSharp;

using OpenCvSharp.Extensions;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using VideoProcessing.Services;

using Yolov5;

namespace VideoProcessing.Handlers

{

**internal class LocalHandler : IDetectHandler<DetectedYolo5Object>**

{

public Dictionary<string, IYoloDetector> AvailableYoloNets { get; set; }

public LocalHandler(Dictionary<string, IYoloDetector> availableYoloNets)

{

this.AvailableYoloNets = availableYoloNets;

}

public List<DetectedYolo5Object> GetData(Mat image)

{

return SendLocalData(image, AvailableYoloNets);

}

public List<DetectedYolo5Object> SendLocalData(Mat image, Dictionary<string, IYoloDetector> availableYoloNets)

{

float threshold = 0.7f;

string variation = "nano";

Yolo5Service yolo5Service = new(new Yolo5ServiceOptions()

{

AvailableNets = availableYoloNets

});

var localResult = GetDetections(variation, threshold, image.ToBitmap(), yolo5Service);

var data = localResult.Select(o => new DetectedYolo5Object

{

ClassId = o.Label.Id,

Confidence = o.Confidence,

BoxX = o.Rectangle.X,

BoxY = o.Rectangle.Y,

BoxWidth = o.Rectangle.Width,

BoxHeight = o.Rectangle.Height,

}).ToList();

return data;

}

public IEnumerable<YoloDetection> GetDetections(string variation, float? threshold, Image image, IYolo5Service yolo5Service)

{

var detections = yolo5Service.Detect(variation, image, threshold);

return detections;

}

}

}

using OpenCvSharp;

using System.Collections.Generic;

namespace VideoProcessing.Handlers

{

internal interface IDetectHandler<T>

{

public List<T> GetData(Mat image);

}

}

namespace Yolov5.Models

{

/// <summary>

/// Model descriptor.

/// </summary>

public interface IYoloModel

{

int Width { get; set; }

int Height { get; set; }

int Dimensions { get; set; }

float Confidence { get; set; }

float MulConfidence { get; set; }

float Overlap { get; set; }

string[] Outputs { get; set; }

List<YoloLabel> Labels { get; set; }

}

}

namespace Yolov5.Models

{

**public class Yolo5Model : IYoloModel**

{

public int Width { get; set; } = 640;

public int Height { get; set; } = 640;

public int Dimensions { get; set; } = 7;

public float Confidence { get; set; } = 0.75f;

public float MulConfidence { get; set; } = 0.25f;

public float Overlap { get; set; } = 0.45f;

public string[] Outputs { get; set; } = new[] { "output0" };

public List<YoloLabel> Labels { get; set; } = new List<YoloLabel>()

{

new YoloLabel { Id = 1, Name = "cucumber" },

new YoloLabel { Id = 2, Name = "trash" },

};

}

}

namespace Yolov5

{

**public class DetectedYolo5Object**

{

public required int ClassId { get; set; }

public required double Confidence { get; set; }

public required double BoxX { get; set; }

public required double BoxY { get; set; }

public required double BoxWidth { get; set; }

public required double BoxHeight { get; set; }

}

}

using System.Drawing;

namespace Yolov5

{

public interface IYoloDetector

{

List<YoloDetection> Detect(Image image, float? threshold);

}

}

using System.Drawing;

namespace Yolov5

{

/// <summary>

/// Object prediction.

/// </summary>

**public class YoloDetection**

{

public required YoloLabel Label { get; set; }

public required RectangleF Rectangle { get; set; }

public required float Confidence { get; set; }

}

}

using Microsoft.ML.OnnxRuntime;

using Microsoft.ML.OnnxRuntime.Tensors;

using System.Collections.Concurrent;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Drawing.Imaging;

using Yolov5.Models;

namespace Yolov5

{

/// <summary>

/// Yolov5 prodictor.

/// </summary>

**public class YoloDetector<T> : IDisposable, IYoloDetector where T : IYoloModel**

{

private readonly T \_model;

private readonly InferenceSession \_inferenceSession;

/// <summary>

/// Creates new instance of YoloScorer with weights path and options.

/// </summary>

public YoloDetector(T model, string weightsPath, SessionOptions? opts = null)

{

\_model = model;

\_inferenceSession = new InferenceSession(File.ReadAllBytes(weightsPath), opts ?? SessionOptions.MakeSessionOptionWithCudaProvider(0));

}

/// <summary>

/// Runs object detection.

/// </summary>

public List<YoloDetection> Detect(Image image, float? threshold)

{

threshold ??= \_model.Confidence;

return Supress(ParseOutput(Inference(image), image, threshold.Value));

}

/// <summary>

/// Disposes YoloScorer instance.

/// </summary>

public void Dispose()

{

\_inferenceSession.Dispose();

}

/// <summary>

/// Resizes image keeping ratio to fit model input size.

/// </summary>

private Bitmap ResizeImage(Image image)

{

PixelFormat format = image.PixelFormat;

var output = new Bitmap(\_model.Width, \_model.Height, format);

var (w, h) = (image.Width, image.Height); // image width and height

var (xRatio, yRatio) = (\_model.Width / (float)w, \_model.Height / (float)h); // x, y ratios

var ratio = Math.Min(xRatio, yRatio); // ratio = resized / original

var (width, height) = ((int)(w \* ratio), (int)(h \* ratio)); // roi width and height

var (x, y) = ((\_model.Width / 2) - (width / 2), (\_model.Height / 2) - (height / 2)); // roi x and y coordinates

var roi = new Rectangle(x, y, width, height); // region of interest

using (var graphics = Graphics.FromImage(output))

{

graphics.Clear(Color.FromArgb(0, 0, 0, 0)); // clear canvas

graphics.SmoothingMode = SmoothingMode.None; // no smoothing

graphics.InterpolationMode = InterpolationMode.Bilinear; // bilinear interpolation

graphics.PixelOffsetMode = PixelOffsetMode.Half; // half pixel offset

graphics.DrawImage(image, roi); // draw scaled

}

return output;

}

/// <summary>

/// Extracts pixels into tensor for net input.

/// </summary>

private Tensor<float> ExtractPixels(Image image)

{

var bitmap = (Bitmap)image;

var rectangle = new Rectangle(0, 0, bitmap.Width, bitmap.Height);

BitmapData bitmapData = bitmap.LockBits(rectangle, ImageLockMode.ReadOnly, bitmap.PixelFormat);

int bytesPerPixel = Image.GetPixelFormatSize(bitmap.PixelFormat) / 8;

var tensor = new DenseTensor<float>(new[] { 1, 3, \_model.Height, \_model.Width });

unsafe // speed up conversion by direct work with memory

{

Parallel.For(0, bitmapData.Height, (y) =>

{

byte\* row = (byte\*)bitmapData.Scan0 + (y \* bitmapData.Stride);

Parallel.For(0, bitmapData.Width, (x) =>

{

tensor[0, 0, y, x] = row[x \* bytesPerPixel + 2] / 255.0F; // r

tensor[0, 1, y, x] = row[x \* bytesPerPixel + 1] / 255.0F; // g

tensor[0, 2, y, x] = row[x \* bytesPerPixel + 0] / 255.0F; // b

});

});

bitmap.UnlockBits(bitmapData);

}

return tensor;

}

/// <summary>

/// Runs inference session.

/// </summary>

private DenseTensor<float>[] Inference(Image image)

{

Bitmap? resized = null;

if (image.Width != \_model.Width || image.Height != \_model.Height)

{

resized = ResizeImage(image); // fit image size to specified input size

}

var inputs = new List<NamedOnnxValue> // add image as onnx input

{

NamedOnnxValue.CreateFromTensor("images", ExtractPixels(resized ?? image))

};

IDisposableReadOnlyCollection<DisposableNamedOnnxValue> result = \_inferenceSession.Run(inputs); // run inference

var output = new List<DenseTensor<float>>();

foreach (var item in \_model.Outputs) // add outputs for processing

{

output.Add(result.First(x => x.Name == item).Value as DenseTensor<float>);

};

return output.ToArray();

}

/// <summary>

/// Parses net outputs (sigmoid or detect layer) to predictions.

/// </summary>

private List<YoloDetection> ParseOutput(DenseTensor<float>[] output2, Image image, float threshold)

{

var output = output2[0];

var result = new ConcurrentBag<YoloDetection>();

var (w, h) = (image.Width, image.Height); // image w and h

var (xGain, yGain) = (\_model.Width / (float)w, \_model.Height / (float)h); // x, y gains

var gain = Math.Min(xGain, yGain); // gain = resized / original

var (xPad, yPad) = ((\_model.Width - w \* gain) / 2, (\_model.Height - h \* gain) / 2); // left, right pads

Parallel.For(0, (int)output.Length / \_model.Dimensions, (i) =>

{

if (output[0, i, 4] <= threshold) return;

Parallel.For(5, \_model.Dimensions, (j) =>

{

output[0, i, j] = output[0, i, j] \* output[0, i, 4]; // class confidence \* object confidence

});

var max = output[0, i, 5];

var maxClassJ = 5;

for (var j = 5; j < \_model.Dimensions; j++)

{

if (output[0, i, j] > max)

{

max = output[0, i, j];

maxClassJ = j;

}

}

if (output[0, i, maxClassJ] <= \_model.MulConfidence)

{

return;

}

float xMin = ((output[0, i, 0] - output[0, i, 2] / 2) - xPad) / gain; // unpad bbox tlx to original

float yMin = ((output[0, i, 1] - output[0, i, 3] / 2) - yPad) / gain; // unpad bbox tly to original

float xMax = ((output[0, i, 0] + output[0, i, 2] / 2) - xPad) / gain; // unpad bbox brx to original

float yMax = ((output[0, i, 1] + output[0, i, 3] / 2) - yPad) / gain; // unpad bbox bry to original

xMin = Clamp(xMin, 0, w - 0); // clip bbox tlx to boundaries

yMin = Clamp(yMin, 0, h - 0); // clip bbox tly to boundaries

xMax = Clamp(xMax, 0, w - 1); // clip bbox brx to boundaries

yMax = Clamp(yMax, 0, h - 1); // clip bbox bry to boundaries

YoloLabel label = \_model.Labels[maxClassJ - 5];

var prediction = new YoloDetection

{

Label = label,

Confidence = output[0, i, maxClassJ],

Rectangle = new RectangleF(xMin, yMin, xMax - xMin, yMax - yMin),

};

result.Add(prediction);

});

return result.ToList();

}

/// <summary>

/// Removes overlaped duplicates (nms).

/// </summary>

private List<YoloDetection> Supress(List<YoloDetection> items)

{

var result = new List<YoloDetection>(items);

foreach (var item in items) // iterate every prediction

{

foreach (var current in result.ToList()) // make a copy for each iteration

{

if (current == item) continue;

var (rect1, rect2) = (item.Rectangle, current.Rectangle);

RectangleF intersection = RectangleF.Intersect(rect1, rect2);

float intArea = intersection.Width \* intersection.Height; // intersection area

float unionArea = rect1.Width \* rect1.Height + rect2.Width \* rect2.Height - intArea; // union area

float overlap = intArea / unionArea; // overlap ratio

if (overlap >= \_model.Overlap)

{

if (item.Confidence >= current.Confidence)

{

result.Remove(current);

}

}

}

}

return result;

}

/// <summary>

/// Returns value clamped to the inclusive range of min and max.

/// </summary>

private float Clamp(float value, float min, float max)

{

return (value < min) ? min : (value > max) ? max : value;

}

}

}

namespace Yolov5

{

/// <summary>

/// Detected object label.

/// </summary>

**public class YoloLabel**

{

public required int Id { get; set; }

public required string Name { get; set; }

}

}

**ПРИЛОЖЕНИЕ Б**

(Обязательное)

**Схема взаимодействия классов**

**ПРИЛОЖЕНИЕ В**

(обязательное)

**Руководство системного программиста**

1. Общие сведения о программном комплексе по обнаружению объектов.

Разработанный программный комплекс предназначен для решения задачи обнаружения объектов на изображениях. Для работы клиента необходимо наличие*.NET Core 7.0.0* и выше.

Для корректной работы приложения необходимо соблюдение минимально допустимых системных требований:

– 2 ГБ оперативной памяти (рекомендуется 4 ГБ и выше);

– процессор с тактовой частотой не ниже 1,8 ГГц. Рекомендуется использовать как минимум двухъядерный процессор.

2. Структура программного комплекса по обнаружению объектов.

Программное обеспечение состоит из одного приложения:

– клиент – использует веб-камеру для обнаружения объектов на заданных изображениях в реальном времени.

3. Настройка программного комплекса по обнаружению объектов.

Для запуска приложения необходимо указать адрес, про которому будет доступен сервер в файле *appsettings.json* запустить исполняемый файл.

Код подлежащий редактированию представлен на рисунке В.1.

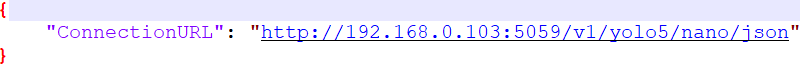


Рисунок В.1 – Файл содержащий строку подключения к серверу

4. Проверка программного комплекса по обнаружению объектов. После проведения всех необходимых настроек приложение готово к запуску, для этого достаточно запустить исполняемый файл.

Открытие консоли и видеоизображения с веб-камеры свидетельствует о корректной работе приложения.

5. Дополнительные возможности.

Дополнительные возможности в приложении отсутствуют.

6. Сообщения системному программисту.

При возникновении ошибок запуска приложения необходимо убедиться, что указанный адрес не занят другим приложением.

**ПРИЛОЖЕНИЕ Г**

(обязательное)

**Руководство программиста**

1. Назначение и условия применения программного комплекса по обнаружению объектов.

Разработанный программный комплекс предназначен для решения задачи обнаружения объектов на изображениях.

Условия необходимые для выполнения программы:

– объем оперативной памяти устройства не менее 2 ГБ;

– процессор с тактовой частотой не ниже 1,8 ГГц. Рекомендуется использовать как минимум двухъядерный процессор;

– наличие *.NET Runtime 7.0.0*.

2. Характеристики программного комплекса по обнаружению объектов.

Работа с программным комплексом происходит с помощью клиентского приложения.

3. Обращение к программному комплексу по обнаружению объектов.

Для просмотра и редактирования исходного кода программного необходимо открыть файл *VideoProcessing.sln* с помощью среды разработки, поддерживающей *.NET*, например, *Visual Studio*.

4. Входные и выходные данные.

Входными данными программы являются видеопоток в любом формате.

Выходными данными является видеопоток с отмеченными найденными объектами.

5. Сообщения.

При возникновении ошибок работы приложения необходимо проверить информацию, выводимую в терминал для принятия дальнейших действий по решению проблемы.

**ПРИЛОЖЕНИЕ Д**

(обязательное)

**Руководство пользователя**

1. Введение.

Разработанный программный комплекс предназначен для решения задачи обнаружения объектов на изображениях.

Для работы с программным обеспечением пользователь должен иметь начальные сведения и навыки работы с персональным компьютером в рамках любой операционной системы.

2. Назначение и условия применения.

Программное обеспечение предназначено для осуществления обнаружения объектов на видеоизображении, не требуя от пользователей углубленных знаний предметной области.

Условия необходимые для выполнения программы:

– объем оперативной памяти устройства не менее 2 ГБ;

– процессор с тактовой частотой не ниже 1,8 ГГц. Рекомендуется использовать как минимум двухъядерный процессор.

3. Подготовка к работе.

Для работы необходимо открыть исполняемый файл, в случае отсутствия проблем в работоспособности приложения должен открыться терминал и окно с изображением, получаемым с веб-камеры.

4. Описание операций.

Операция 1: Обнаружение объектов на изображении;

Условия необходимые для выполнения: запущенное приложение;

Подготовительные действия: отсутствуют.

Основные действия: отсутствуют.

Заключительные действия: отсутствуют.

5. Аварийные ситуации.

Для избегания возникновения аварийных ситуаций необходимо использовать приложение на устройстве с веб-камерой.

В случае отказа работы приложения обратиться к системному программисту.

6. Рекомендации по освоению.

Для освоения в приложении необходимо изучить информацию ознакомиться с руководством пользователя.