МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра информационных систем управления

ЛУШАКОВА Светлана Сергеевна

ПОВЫШЕНИЕ ТОЧНОСТИ ОБНАРУЖЕНИЯ ДРОНОВ НА ИЗОБРАЖЕНИЯХ

Дипломная работа

Научный руководитель:
доктор технических наук,
доцент
А.М. Недзьведь

Допущена к защите

«__» _____ 2025 г.

Заведующий кафедрой информационных систем управления
доктор технических наук, доцент
А.М. Недзьведь

Минск, 2025

РЕФЕРАТ

Дипломная работа, 45 страниц, 11 рисунков, 5 таблиц, 4 формулы, 2 приложения, 11 источников.

Ключевые слова: ТОЧНОСТЬ, ОБНАРУЖЕНИЕ, НЕЙРОННЫЕ СЕТИ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ.

Объект исследования – обнаружение объектов с помощью нейронных сетей.

Предмет исследования – точность обнаружения дронов на кадрах видео нейросетевыми детекторами.

Цель работы – повысить точность обнаружения дронов нейросетевыми детекторами.

Методы исследования – сравнительный анализ точности различных нейронных сетей-детекторов объектов; выбор оптимальной архитектуры нейронной сети и ее параметров; применение для детекции одновременно двух нейронных сетей с различными параметрами; выбор методов аугментации датасетов.

Полученные результаты и их новизна: достигнуто повышение точности обнаружения дронов на изображениях с помощью трех разработанных алгоритмов путем обучения и тестирования различных архитектур нейросетевого детектора YOLO.

Достоверность материалов и результатов дипломной работы: использованные материалы и результаты дипломной работы являются достоверными. Работа выполнена самостоятельно.

Область возможного практического применения: результаты могут быть использованы в работах различных предприятий, чьи интересы связаны с отслеживанием дронов и повышением точности их обнаружения, а также в дальнейших исследованиях применения нейронных сетей для детекции объектов.

РЭФЕРАТ

Дыпломная праца, 45 старонак, 5 табліц, 11 малюнкаў, 4 формулы, 2 прыкладанні, 11 крыніц.

Ключавыя словы: ДАКЛАДНАСЦЬ, ВЫЯЎЛЕННЕ, НЕЙРОННЫЯ СЕТКІ, КАМП'ЮТАРНЫ ЗРОК.

Аб’ект даследавання – выяўленне аб’ектаў з дапамогай нейронных сетак.

Прадмет даследавання – дакладнасць выяўлення дронаў на кадрах відэа нейрасеткавымі дэтэктарамі.

Мэта працы – павысіць дакладнасць выяўлення дронаў нейрасеткавымі дэтэктарамі.

Метады даследавання – параўнальны аналіз дакладнасці розных нейронавых сетак-дэтэктараў аб’ектаў; выбар аптымальнай архітэктуры нейронавай сеткі і яе параметраў; ужыванне для дэтэкцыі адначасова двух нейронавых сетак з рознымі параметрамі; выбар метадаў аўгментацыі датасэтаў.

Атрыманыя вынікі і іх навізна: дасягнута павышэнне дакладнасці выяўлення дронаў на выявах з дапамогай трох распрацаваных алгарытмаў шляхам навучання і тэставання розных архітэктур нейрасеткавага дэтэктара YOLO.

Дакладнасць матэрыялаў і вынікаў дыпломнай працы: выкарыстаныя матэрыялы і вынікі дыпломнай працы з’яўляюцца дакладнымі. Праца выканана самастойна.

Вобласць магчымага практычнага прымянення: вынікі могуць быць выкарыстаны ў работах розных прадпрыемстваў, чые інтарэсы звязаны з адсочваннем дронаў і павышэннем дакладнасці іх выяўлення, а таксама ў далейшых даследаваннях прымянення нейронавых сетак для дэтэкцыі аб’ектаў.

ABSTRACT

Diploma work, 45 pages, 5 tables, 11 illustrations, 4 formulas, 2 appendices, 11 sources.

Keywords: ACCURACY, DETECTION, NEURAL NETWORKS, COMPUTER VISION.

The object of research is object detection using neural networks.

The subject of the study is accuracy of detection of drones on video frames by neural network detectors.

The purpose of the research is to improve the accuracy of drone detection by neural network detectors.

Methods of research: comparative analysis of the accuracy of various object-detecting neural networks; selection of the optimal neural network architecture and its parameters; application of two neural networks with different parameters simultaneously for detection; selection of dataset augmentation techniques.

The results of the work and their novelty: improved the accuracy of drone detection on images using three developed algorithms by training and testing different architectures of the YOLO neural network detector.

Authenticity of the materials and results of the diploma work: the materials used and the results of the diploma work are authentic. The work has been put through independently.

Recommendations on the usage: the results can be used by companies whose activities are related to tracking drones and improving the accuracy of their detection, as well as in further research on the application of neural networks for object detection.

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

ИИ	– искусственный интеллект;
Датасет	– это обработанный и структурированный набор данных;
Large-модель	– модель нейронной сети-детектора, обученная на всех изображениях датасета;
Small-модель	– модель нейронной сети-детектора, обученная только на части изображений датасета с объектами, имеющими малый размер.

ВВЕДЕНИЕ

Будь то личная или социальная жизнь, домашнее или рабочее пространство – во многих сферах деятельности человека появляются дополнительные инструменты, которые облегчают обработку огромного количества информации и решение сложных задач, требующих наличия аналитических навыков и способности к обучению [10].

В настоящее время особое распространение получило понятие «искусственный интеллект». Искусственный интеллект (англ. artificial intelligence; AI) в широком смысле – это способность компьютера имитировать или выполнять действия, свойственные человеческому интеллекту. ИИ – современная дисциплина на стыке мира математики, информатики, биологии и других наук, которая изучает технологии, позволяющие человеку создавать «интеллектуальные» программы и «обучать» компьютеры решать задачи самостоятельно [2].

Современное развитие технологий искусственного интеллекта и, в частности, нейронных сетей, открывает новые возможности в области компьютерного зрения [5]. Обнаружение и идентификация объектов на изображениях являются ключевыми задачами, которые имеют широкий спектр применения, из чего вытекает актуальность и значимость их исследования [9]. Важной задачей в данной области остается повышение точности обнаружения, особенно когда речь идет о малоразмерных объектах. В качестве примера таких объектов могут выступать дроны [11].

Дроны, как правило, трудно обнаружимы не только из-за их размеров и низкого контраста на фоне, но и из-за расстояния до них, что приводит к снижению точности работы алгоритмов, и, как следствие, влияет на практические результаты [6]. В связи с этим появляется необходимость в исследовании существующих нейросетевых алгоритмов, а также в разработке новых подходов к процессам обучения и тестирования [4].

Данная работа посвящена повышению точности обнаружения дронов на изображениях с применением нейронных сетей. В рамках исследования:

– проведен анализ текущего состояния проблемы и современных подходов к обнаружению объектов на изображениях;

– предложены алгоритмы, повышающие точность обнаружения за счет задания различных параметров нейросетевых детекторов;

– предложенные алгоритмы программно реализованы, протестированы и обучены.

ГЛАВА 1. АНАЛИЗ ПРОБЛЕМЫ И ПОСТАНОВКА ЗАДАЧИ

1.1 Современные алгоритмы компьютерного зрения

Компьютерное зрение – область информационных технологий, посвященная развитию технических средств, технологий, методов и алгоритмов, программного обеспечения для получения и применения знаний, содержащихся в видеоинформации (изображениях, видеопоследовательностях, 3D- сценах и других видеопредставлениях окружающей нас реальности).

Одно из значимых направлений компьютерного зрения – обработка и распознавание изображений. К основным задачам распознавания изображений относятся:

1. Классификация объектов. Задача классификации заключается в определении, к какому классу относится изображение или его часть. Например, модель может определить, содержит ли изображение собаку, кошку или автомобиль, однако при этом не указывается местоположение объекта на изображении – только его категория. Для решения проблем классификации объектов используют алгоритмы глубокого обучения с целью уменьшения вычислительной нагрузки, качественного извлечения признаков и принятия решения.

2. Сегментация объектов. Существует два типа сегментации объектов: семантическая сегментация и сегментация экземпляров. Семантическая сегментация – задача разделения изображения на различные сегменты, каждый из которых представляет отдельный класс объектов, например, класс людей, класс автомобилей, класс котов и т.д. Сегментация экземпляров – задача компьютерного зрения, которая заключается в идентификации и выделении каждого объекта на изображении уникальным сегментом, включая присвоение каждому объекту уникальной метки. Ключевое отличие описанных типов сегментации заключается в том, что семантическая работает на уровне классов, а сегментация экземпляров – на уровне конкретных объектов, причем последняя является более детализированной и сложной проблемой, требующей большего объема вычислений и данных для обучения моделей.

3. Обнаружение объектов. Заключается в обнаружении и определении местоположения объектов на изображении или видео. Обычно результатом является набор рамок, ограничивающих прямоугольников (bounding boxes), вокруг объектов и меток классов, к которым они относятся. Например, модель

может обнаружить на изображении несколько автомобилей и пешеходов и указать их точное расположение на изображении.

Наглядные примеры для основных задач распознавания изображений представлены на рисунке 1.1:

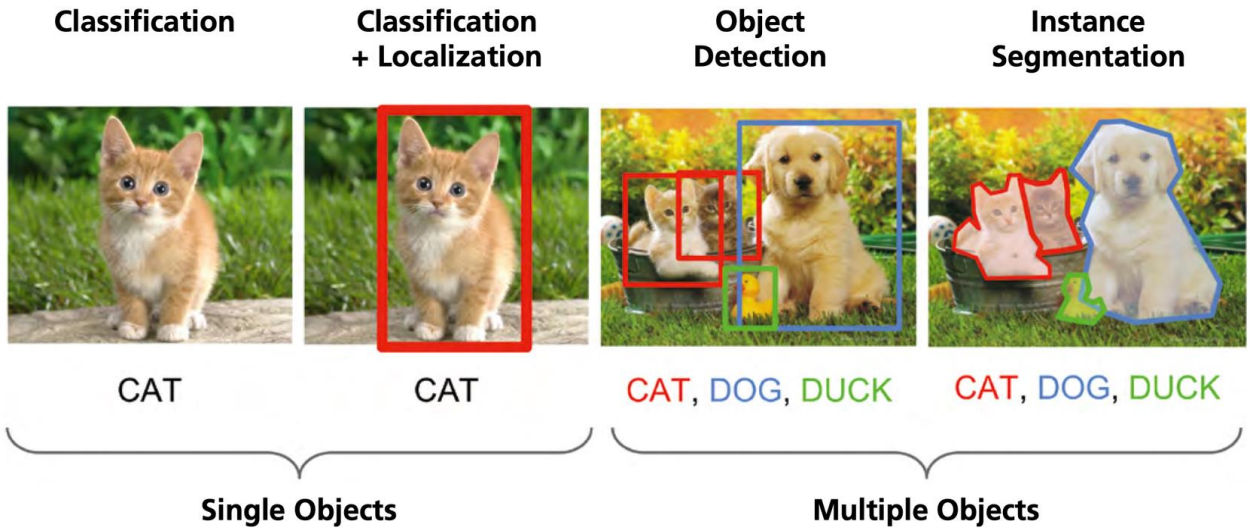

Рисунок 1.1 – Основные задачи распознавания

В последние годы для решения упомянутых задач стали успешно применяться методы глубокого обучения.

Глубокое обучение произвело революцию в области компьютерного зрения, позволив разрабатывать высокоэффективные методы и архитектуры нейронных сетей, способные распознавать сложные закономерности в визуальных данных. В основе большинства методов глубокого обучения лежат нейронные сети.

Далее остановимся на архитектурах, основанных на глубоком обучении, и нейросетевых моделях обнаружения объектов, рассмотрим эти области более подробно.

1.1.1 Архитектуры, основанные на глубоком обучении

Сверточные нейронные сети – вид нейронных сетей прямого распространения, в которых слои представляют собой 2d-массивы, каждый из которых получается из нескольких предыдущих путем их свертки с локальными фильтрами небольшого размера – матрицами, в подавляющем большинстве случаев размера меньшего 7x7 элементов.

CNN состоит из одного или нескольких сверточных слоев (часто с этапом предварительной обработки – нормализацией), слоев объединения,

полносвязных слоев (также известных как плотные слои) и слоев нормализации.

Ниже приведены известные архитектуры CNN [1]:

– LeNet (1998), разработанная Яном Лекуном и др., была предназначена для распознавания рукописных цифр и почтовых индексов. Это одна из первых сверточных сетей, которая использовалась в основном для распознавания символов.

– AlexNet (2012), разработанная Алексом Крижевским, Ильей Суцкевером и Джеффри Хинтоном, значительно превзошла другие модели в соревновании ImageNet (ILSVRC-2012), благодаря чему сверточные нейронные сети стали более популярны. AlexNet отличалась более глубокими слоями и линейными активационными функциями (ReLU) для ускорения обучения.

– VGG (2014), разработанная группой Visual Graphics из Оксфорда (отсюда VGG), продемонстрировала важность глубины в архитектурах CNN. В ней использовались очень маленькие (3x3) сверхточные фильтры, и она была углублена до 16-19 слоёв.

– GoogLeNet (2014), в которой был представлен модуль Inception, значительно сокративший количество параметров в сети (4 миллиона по сравнению с 60 миллионами в AlexNet). В этой архитектуре для улучшения обучения использовались пакетная (batch) нормализация, искажения изображений и RMSprop.

– ResNet (2015), разработанная Каймингом Хэ и др., представила остаточное обучение для упрощения обучения сетей, которые значительно глубже, чем те, что использовались ранее. Она использовала «пропускные соединения», чтобы градиенты проходили через сеть без ухудшения качества, и выиграла ILSRC 2015 с глубиной до 152 слоев.

– DenseNet (2017) усовершенствовал идею повторного использования признаков в ResNet. Каждый слой соединяется с каждым другим слоем по принципу прямой связи. Эта архитектура обеспечивает максимальный поток информации между слоями сети.

– MobileNet (2017) основаны на оптимизированной архитектуре, которая использует свёрточные нейронные сети для построения лёгких глубоких нейронных сетей. Они предназначены для мобильных и периферийных устройств и отличаются эффективностью с точки зрения вычислений и энергопотребления.

1.1.2 Нейросетевые детекторы

Ниже перечислим широко используемые архитектуры нейросетевых детекторов [1]:

– RCNN (расшифровывается как Regions with CNN features, или «регионы с функциями CNN»), представленный Россом Гиршиком и др., был одной из первых систем обнаружения объектов на основе глубокого обучения. В данном детекторе используется выборочный поиск для создания предложений по областям, которые затем передаются в CNN для извлечения признаков, которые в конечном итоге классифицируются с помощью SVM. Несмотря на свою мощность, RCNN работает довольно медленно из-за высокой вычислительной нагрузки при обработке каждого предложения по области отдельно.

– Fast R-CNN, улучшенная версия RCNN, также разработанная Россом Гиршиком, устраняет неэффективность за счет совместного использования вычислений. Она обрабатывает все изображение с помощью CNN для создания сверточной карты признаков, а затем применяет слой объединения областей интереса для извлечения признаков из карты признаков для каждого предложения по области. Такой подход значительно ускоряет обработку и повышает точность за счёт использования многозадачной потери, которая объединяет классификацию и регрессию ограничивающих рамок.

– Faster R-CNN, созданный Шаоцином Ренном и др., улучшает Fast R-CNN, добавляя сеть предложений регионов. Данная сеть заменяет алгоритм выборочного поиска, использовавшийся в предыдущих версиях, и одновременно прогнозирует границы объектов и баллы в каждой позиции карты признаков. Подобная интеграция повышает скорость и точность создания предложений регионов.

– Cascade R-CNN, разработанный Чжаовей Цай и Нуно Васконселос, является расширением Faster R-CNN, которое повышает эффективность обнаружения за счёт использования каскада детекторов R-CNN, каждый из которых обучается с увеличивающимся порогом пересечения и объединения. В итоге полученный многоэтапный подход постепенно уточняет прогнозы, что приводит к более точному обнаружению объектов.

– SSD (расшифровывается как Single Shot MultiBox), твердотельный накопитель, разработанный Вэй Лю и др., который упрощает процесс обнаружения, устраняя необходимость в отдельной сети для определения границ области. В детекторе используется одна нейронная сеть для прогнозирования координат ограничивающей рамки и вероятностей классов непосредственно по полным изображениям, чем достигается баланс между

скоростью и точностью. SSD разработан для обеспечения высокой эффективности, что одновременно делает его подходящим для задач обработки в реальном времени.

– YOLO (расшифровывается как You Only Look Once) – популярная модель для обнаружения объектов, разработанная Джозефом Редмоном и др. В данном алгоритме процесс обнаружения рассматривается как задача регрессии. Модель делит изображение на сетку и прогнозирует ограничивающие прямоугольники с вероятностями для каждой ячейки сетки. YOLO работает довольно быстро и достаточно точно даже во время обработки изображений в реальном времени, что делает её подходящей для задач, требующих высокой скорости.

– RT-DETR, трансформерный детектор, разработанный компанией Baidu. Real-Time DETR представляет собой легковесную и высокопроизводительную версию DETR, оптимизированную для работы в реальном времени. Данный детектор сочетает в себе эффективные архитектурные решения, гибридный сборщик данных и асинхронный параллелизм, позволяя достичь высокой скорости обработки без значительной потери точности. Модель особенно подходит для встраиваемых систем и мобильных устройств, где важны как производительность, так и энергоэффективность.

1.2 Текущее состояние проблемы в выбранной области

В последние десятилетия область компьютерного зрения значительно развилась благодаря достижениям в области машинного обучения и нейросетевых технологий. Современные нейросетевые алгоритмы, такие как YOLO (You Only Look Once) [8], Faster R-CNN [9], SSD (Single Shot MultiBox Detector) [7] и другие, описанные в предыдущем пункте, демонстрируют высокую эффективность в задачах обнаружения объектов. Однако, проблемы, связанные с обнаружением малоразмерных объектов, по-прежнему остаются актуальными [11].

Как было упомянуто во введении, обнаружение малоразмерных объектов усложняет не только их размер, но и потерянность на фоне, наличие других более заметных объектов. Например, в системах видеонаблюдения, безопасности, медицине или автономных транспортных средствах важно не только обнаружить объект, но и сделать это быстро, с высокой точностью и надежностью. Тем не менее, существующие алгоритмы демонстрируют значительное снижение точности при работе с малоразмерными объектами [6]. Исследования показывают, что большинство традиционных нейросетевых

моделей сосредоточены на крупных и средних объектах [3], тем самым оставляя малоразмерные объекты вне поля их зрения. Из всего вышеперечисленного вытекает вывод о необходимости проведения дополнительного анализа и улучшения существующих методов, изучении более специализированных или адаптированных подходов.

1.3 Постановка задачи

На основе проведенного анализа текущего состояния проблемы в области обнаружения малоразмерных объектов на изображениях можно выделить ключевые задачи, которые необходимо решить для повышения эффективности существующих методов. Основной целью настоящей работы является разработка подходов к улучшению точности обнаружения дронов с использованием нейросетевых алгоритмов.

Для достижения этой цели были поставлены следующие задачи:

– рассмотреть существующие алгоритмы распознавания объектов и их особенности при работе с малоразмерными объектами;

– обозначить причины снижения точности при обнаружении малоразмерных объектов, таких как дроны;

– проанализировать факторы, влияющие на качество детекции (размер объекта, контрастность фона, расстояние до объекта);

– предложить и протестировать другие подходы на базе алгоритма YOLO;

– сравнить точности полученных алгоритмов, выделить наиболее эффективные.

1.4. Выводы

В первой главе были получены следующие основные результаты:

– изучена и проанализирована литература по теме;

– приведены наиболее эффективные и широко используемые методы компьютерного зрения в области обработки изображений;

– выявлены проблемы обнаружения малоразмерных объектов с использованием нейронных сетей;

– на основе анализа проблем сформулирована задача повышения точности распознавания дронов на изображениях.

ГЛАВА 2. РАЗРАБОТАННЫЙ АЛГОРИТМ

2.1 Алгоритмы

В рамках работы были реализованы алгоритмы, направленные на улучшение точности обнаружения объектов небольшого размера с использованием модели YOLO от Ultralytics. Основная идея заключается в адаптации процесса обучения и постобработки результатов детекции для повышения эффективности работы модели при работе с объектами разной площади.

В качестве первого этапа для всех алгоритмов выступает обучение моделей на полученных датасетах, подробнее о котором будет рассказано в Главе 3. После получения моделей, обученных на изображениях с объектами различного размера, появляется возможность тестировать их как по отдельности, так и в различных комбинациях. Подробно процесс тестирования будет описан в Главе 4.

Стоит упомянуть, что модель нейронной сети-детектора, обученная на всех изображениях датасета, далее будет указываться как Large-модель, а модель нейронной сети-детектора, обученная только на части изображений датасета с объектами, имеющими малый размер, далее будет иметь название Small-модель.

Ниже будут представлены полученные в ходе работы алгоритмы для более точного обнаружения дронов на изображениях.

2.1.1 Алгоритм А

В первом алгоритме предлагается использовать все рамки (ограничивающие объекты прямоугольники), полученные от Large-модели, а после добавлять рамки, полученные от Small-модели, не пересекающиеся с рамками от Large-модели.

Данный алгоритм состоит из следующей последовательности шагов:

Шаг 1. Обучение модели Large на всех изображениях.

Шаг 2. Обучение модели Small на изображениях с малыми объектами.

Шаг 3. Задание границы площади для дальнейшей сортировки рамок.

Шаг 4. Задание границы показателя достоверности (confidence score) для дальнейшей сортировки рамок.

Шаг 5. Получение всех рамок объектов на изображениях датасета от модели Large.

Шаг 6. Получение всех рамок объектов на изображениях датасета от модели Small.

Шаг 7. Сортировка рамок от модели Small: использование в дальнейшей работе только тех, чья площадь не превышает заданной границы и чей показатель достоверности не ниже заданного.

Шаг 8. Добавление к рамкам Large определенных рамок Small: в случае, когда рамки, полученные от Small-модели, являются уникальными, то есть не пересекаются (метрика IoU(Intersection over Union) равна 0) с рамками от Large-модели, осуществляется добавление этой рамки к рамкам модели Large как результат распознавания объекта на изображении.

Шаг 9. Тестирование: подсчет метрик accuracy, precision, recall и F1-score.

Первый алгоритм описан ниже на рисунке 2.1 в виде блок-схемы:

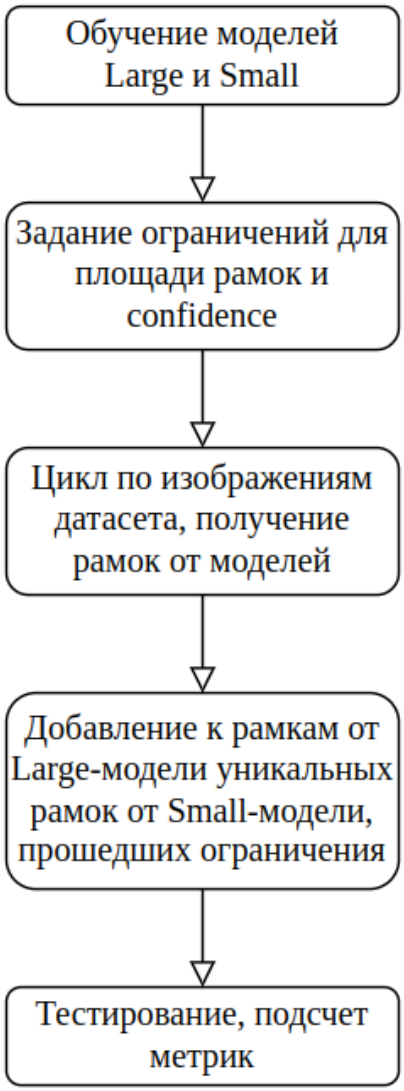

Рисунок 2.1 – Блок-схема первого алгоритма

Такой алгоритм позволяет повысить количество обнаруживаемых объектов, что выражено в снижении показателя False Negative. Однако, как следствие, повышаются и True Positive как количество правильно распознанных дронов среди всех реально существующих, и False Positive как количество обнаруженных объектов там, где их на самом деле не было.

2.1.2 Алгоритм B

Во втором алгоритме предлагается использовать рамки Small-модели и Large-модели, которые будут отсортированы по некоторому заданному заранее ограничению площади рамок и показателю достоверности (confidence score).

Данный алгоритм состоит из следующей последовательности шагов:

Шаг 1. Обучение модели Large на всех изображениях.

Шаг 2. Обучение модели Small на изображениях с малыми объектами.

Шаг 3. Задание границы площади для дальнейшей сортировки рамок.

Шаг 4. Задание границы показателя достоверности для дальнейшей сортировки рамок.

Шаг 5. Получение всех рамок объектов на изображениях датасета от модели Large.

Шаг 6. Получение всех рамок объектов на изображениях датасета от модели Small.

Шаг 7. Сортировка рамок от модели Small: использование в дальнейшей работе только тех, чья площадь не превышает заданной границы и чей показатель достоверности не ниже заданного.

Шаг 8. Сортировка рамок от модели Large: использование в дальнейшей работе только тех, чья площадь больше заданной границы и чей показатель достоверности не ниже заданного.

Шаг 9. Тестирование: подсчет метрик accuracy, precision, recall и F1-score.

Такой алгоритм позволяет сортировать полученные рамки объектов, используя в качестве критерия их размер и, как следствие, специализирующийся на нем тип модели. Так как задача состоит в том, чтобы лучше находить объекты маленького размера, будет предпочтительнее рамки с маленькой площадью брать от Small-модели, которая была обучена на малых объектах и показывает на них результат лучший, чем Large-модель.

Второй алгоритм описан ниже в виде блок-схемы на рисунке 2.2.

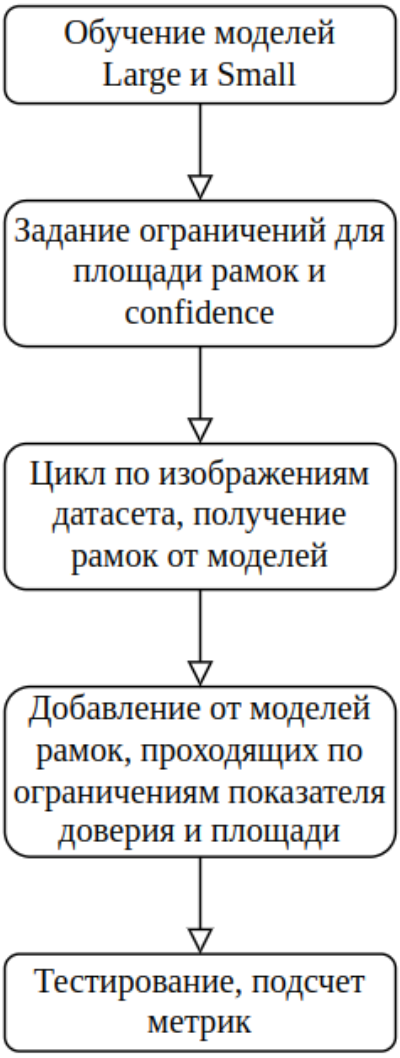

Рисунок 2.2 – Блок-схема второго алгоритма

2.1.3 Алгоритм C

В третьем алгоритме предлагается использовать рамки от Large-модели только с определенным показателем достоверности, что включает следующие шаги:

Шаг 1. Обучение модели Large на всех изображениях.

Шаг 2. Задание границы показателя достоверности (confidence score) для дальнейшей сортировки рамок.

Шаг 3. Получение всех рамок объектов на изображениях датасета от модели Large.

Шаг 4. Сортировка рамок от модели Large: использование в дальнейшей работе только тех, чей показатель достоверности не ниже заданного.

Шаг 5. Тестирование: подсчет метрик accuracy, precision, recall и F1-score.

Третий алгоритм описан ниже на рисунке 2.3 в виде блок-схемы:

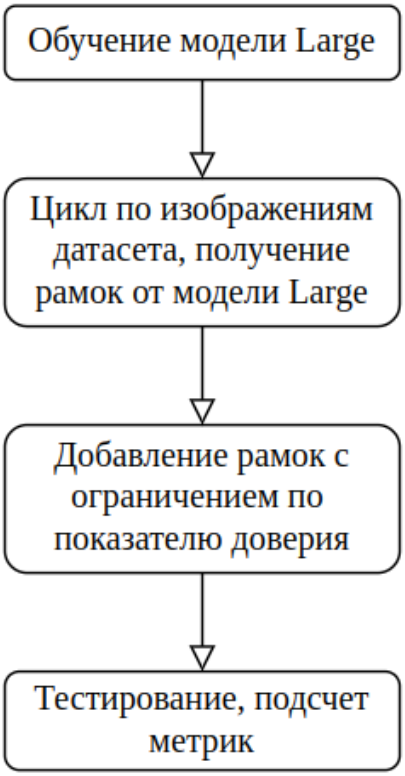

Рисунок 2.3 – Блок-схема третьего алгоритма

Таким образом, при использовании алгоритма повышается точность распознавания, но и модель реагирует на объекты реже, что будет видно при тестировании и практическом использовании данного варианта.

2.2 Выводы

Во второй главе были получены следующие основные результаты:

– были разработаны три алгоритма с целью повышения точности распознавания дронов на изображениях;

– каждый алгоритм был подробно описан;

– были приведены блок-схемы алгоритмов для наглядного описания последовательности действий;

– были приведены краткие выводы по каждому алгоритму.

ГЛАВА 3. ОПИСАНИЕ НАБОРА ДАННЫХ И ПРОЦЕСС ОБУЧЕНИЯ

3.1 Используемый набор данных

Набор данных получен в лаборатории №211 обработки и распознавания изображений в ГНУ «Объединенный институт проблем информатики Национальной академии наук Беларуси» и имеет название Large Dataset.

Датасет состоит из трех выборок: обучающей, тестовой и валидационной. В каждой из них находятся файлы изображений и текстовые файлы с метками (номером класса объекта и координатами его рамки).

Пример изображения из обучающей выборки датасета Large представлен ниже на рисунке 3.1:

Рисунок 3.1 – Пример изображения с объектом

Текстовый файл меток к объекту на изображении состоит из строк, каждая из которых, на примере меток объекта на рисунке 3.1, имеет вид:

1) 0 – номер класса объекта, классификационная метка (в нашем случае «дрон»),

2) 0.3034413406879498 – нормализованная относительно ширины изображения координата центра рамки (bounding box) по оси X,

3) 0.3458032562308371 – нормализованная относительно высоты изображения координата центра рамки по оси Y,

4) 0.21929823761543404 – нормализованная относительно ширины изображения ширина рамки,

5) 0.23876824692246326 – нормализованная относительно высоты изображения высота рамки.

Рамка объекта на изображении выглядит следующим образом:

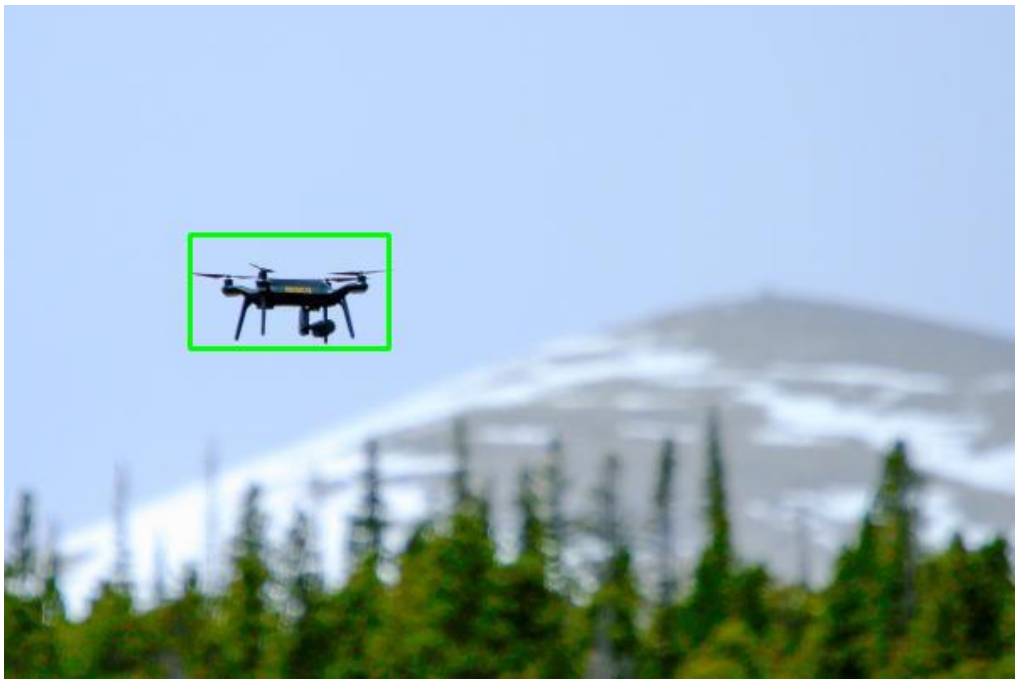

Рисунок 3.2 – Пример изображения с рамкой

Исходный датасет Large Dataset аугментированный, то есть увеличенный за счет применения к изображениям следующих преобразований: изменения угла наклона изображений, обрезка изображений, изменение масштаба. Пример преобразованных изображений представлен на рисунке 3.3:

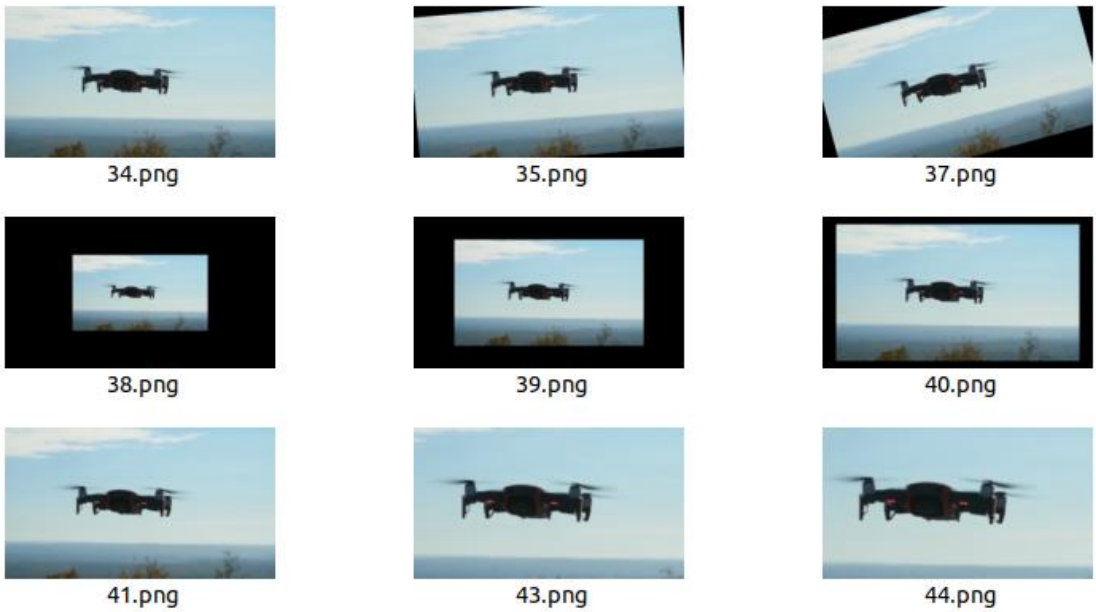

Рисунок 3.3 – Пример преобразованных изображений

В процессе работы Large Dataset был разбит с помощью кода из приложения А на два датасета по признаку площади рамки объекта на изображениях. Суть алгоритма разбиения заключалась в следующем: если площадь рамки объекта на изображении была меньше или равна заранее заданному порогу, например, 8000, то изображение и метки, которые относятся к этому объекту, копировались в новый набор данных, Small Objects Dataset, в противном случае – в другой набор данных, Big Objects Dataset.

Ввиду того, что датасет объектов с маленькими рамками выходил значительно меньше исходного Large, было принято решение увеличить его за счет добавления зашумленных изображений, так как модели, обученные на данных с нанесенным шумом, улучшают способность к обобщению и выявлению закономерности в данных, что может предотвратить переобучение и повысить производительность модели во время тестирования и практического применения.

Зашумленное изображение выглядит следующим образом:

Рисунок 3.3 – Пример зашумленного изображения

Для осуществления зашумления изображений была использована следующая функция:

```
def add_gaussian_noise(image):
   row, col, ch = image.shape
   gauss = np.random.normal(0, 15.0, (row, col, ch))
```

```
    noisy = image + gauss
    noisy = np.clip(noisy, 0, 255).astype(np.uint8)
    return noisy
```

В данном коде 15.0 – стандартное отклонение для гауссовского шума, добавляемого на изображение.

Для дополнительного повышения точности работы Small-модели в процессе обучения была использована дополнительная техника аугментации поворот слева направо (fliplr), встроенная в библиотеку Ultralytics:

```
sm9000_noisy_model = YOLO('yolo12n.pt')
sm9000_noisy_results = sm9000_noisy_model.train(
    data='sm9000_noisy.yaml',
    epochs=1000,
    imgsz=1200,
    batch=4,
    device = 'cuda',
    optimizer = 'AdamW',
    lr0 = 0.0001,
    save_period = 5,
    patience = 200,
    name='yolov12_sm9000_dataset',
    project='sm9000_noisy_1000E',
    fliplr = 0.4
)
```

Такая техника аугментации зеркально отражает изображение вдоль оси x с заданной вероятностью. Данная операция применялась к изображениям датасета с вероятностью 0,4.

Благодаря зашумлению изображений, датасет Small Object Dataset стал в два раза больше, а модель Small, код обучения которой представлен выше, благодаря аугментации и увеличению размера изображений с 640 до 1200 пикселей, показала лучшие результаты по сравнению с моделями, обученными на изначальном Small датасете.

3.2 Описание процесса обучения

За основу для обучения новых моделей были взяты предобученные на COCO модели YOLO от Ultralytics.

Фрагмент кода для обучения модели на Large Dataset:

```
Large_dataset_model = YOLO('yolo12n.pt')
Large_results = Large_dataset_model.train(
  data='Large_dataset.yaml',
  epochs=1000,
  imgsz=640,
  batch=32,
  device = 'cuda',
  optimizer = 'AdamW',
  lr0 = 0.0001,
  save_period = 1,
  patience = 500,
  name='yolov12_Large_dataset',
  project='/runs/Large_dataset_1000E'
)
```

На момент написания работы последней вышедшей версией YOLO являлась модель yolo12. Были обучены модели nano 10-й, 11-й и 12-й версий на 300, 600 и 1000 эпох на описанных в пункте 3.1 наборах данных.

Основные параметры обучаемой модели задавались следующим образом:

– **`data='dataset.yaml'`**, выбор файла конфигурации датасета (буден представлен ниже), содержащего информацию о путях к изображениям, аннотациям, количеству классов и их названиях; является стандартным способом указания данных для обучения в YOLO, так как файл .yaml структурирован и удобен для управления датасетами.

– **`epochs=1000`**, количество эпох (полных проходов по всем данным) для обучения модели, обосновано сложностью задачи и размером датасета; было выбрано количество эпох 1000, так как преимущественно 300 и 600 эпох было недостаточно.

– **`imgsz=640`**, размер входного изображения (в пикселях), до которого будут масштабироваться все изображения перед подачей в модель; увеличение размера до 1200 повысило точность, но замедлило обучение и вынудило уменьшить размер пакета (параметр batch).

– **`batch=32`**, размер пакета (батча, количества изображений, обрабатываемых одновременно); для использованной в работе GPU были выбраны значения 4 с размером входного изображения 1200 пикселей и 32 с размером входного изображения 640 пикселей, так как больший батч (например, 64) требует больше памяти.

– **device='cuda'**, указание устройства, на котором будет выполняться обучение (GPU с CUDA); использование GPU значительно ускоряет обучение по сравнению с CPU, что критически важно при работе в глубоком обучении.

– **optimizer='AdamW'**, ортимизатор для обновления весов модели во время обучения; AdamW – это модификация Adam, которая лучше справляется с регуляризацией (weight decay), обеспечивает хорошую сходимость.

– **lr0=0.0001**, начальная скорость обучения (learning rate); lr0=0,0001 – типичная начальная скорость обучения: она достаточно мала, чтобы избежать резких изменений весов, но достаточно велика для стабильной сходимости.

– **save_period=1** показывает, как часто (в эпохах) сохраняются контрольные точки (checkpoints) модели; частое сохранение, в частности после каждой эпохи, как в примере, гарантирует, что прогресс в случае аварийного завершения обучения не будет потерян, и если же обучение было экстренно завершено, для его возобновления необходимо указать последние сохраненные значения (last.pt) и поставить resume=True.

– **patience=200**, количество эпох без улучшения метрики (например, mAP), после которого обучение будет остановлено (early stopping); значение 200 предполагает, что модель будет обучаться еще 200 эпох, даже если прогресс минимальный, с целью убедиться в отсутствии возможных улучшений.

Параметры save_period и patience позволяют контролировать процесс обучения, избежать потери данных и прогресса. Результаты разных экспериментов хранятся в отдельных папках для предотвращения перезаписи файлов, а также для удобства поиска и анализа.

Файл конфигурации создавался автоматически с помощью написанной функции:

```
def create_yaml(data_path, output_path, object_name):
    data = {
        'path': data_path,
        'train': os.path.join(data_path, 'train'),
        'val': os.path.join(data_path, 'valid'),
        'test': os.path.join(data_path, 'test'),
        'nc': 1,
        'names': [object_name]
    }
    with open(output_path, 'w') as f:
        yaml.dump(data, f, default_flow_style=False)
    print(f"YAML file created at: {output_path}")
```

Пример файла конфигурации .yaml для обучения на наборе данных Large Dataset:

```
names:
- drone
nc: 1
path: /Sviatlana/Large_dataset
test: /Sviatlana/Large_dataset/test
train:/Sviatlana/Large_dataset/train
val:/Sviatlana/Large_dataset/valid
```

В файле указываются названия классов и их количество, пути к датасету, к тестовой, к обучающей и к валидационной выборкам датасета.

Для создания конфигурационного файла, в зависимости от модели, указывались путь к датасету, название .yaml файла и класса объекта:

```
Large_dataset_path = '/Large_dataset'
create_yaml(Large_dataset_path,'Large_dataset.yaml','drone')
```

Для обучения модели на всех объектах использовался датасет Large Dataset, такая модель будет называться large. Для обучения модели только на тех объектах, чья площадь рамки не превышает некоторый порог, использовался датасет Small Objects Dataset, и модель, соответственно, будет называться small. Аналогично, для обучения модели на объектах с рамками, большими порога, использовался Big Objects Dataset, и такая модель будет называться big. Например, модель, прошедшая обучение 1000 эпох на изображениях, площадь рамок объектов которых не превышает (меньше или равна) 8000, будет называться small_8000_1000E.

3.3 Выводы

В третьей главе были получены следующие основные результаты:

– представлен используемый набор данных;

– описан процесс создания датасетов и их аугментации;

– показан и описан процесс обучения моделей;

– перечислены основные параметры и приведено обоснование их использования.

ГЛАВА 4. ТЕСТИРОВАНИЕ АЛГОРИТМОВ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

4.1 Описание процесса тестирования

Процесс тестирования заключался в подсчете метрик качества детекции объектов, получаемых из соотношений количества рамок, найденных YOLO, к реальным. Код функций и процесса тестирования одной модели указан в приложении Б. На рисунке 4.1 представлен результат работы Large-модели, где зеленая рамка – истинная рамка (ground truth bounding box), а синяя – предсказанная моделью (predicted bounding box):

Рисунок 4.1 – Пример полученных рамок

4.1.1 Метрики оценки качества детекции

В качестве ключевых метрик были выбраны accuracy, precision, recall и f1-score, которые вычисляются исходя из полученных значений TruePositive, TrueNegative, FalsePositive, FalseNegative (рисунок 4.2).

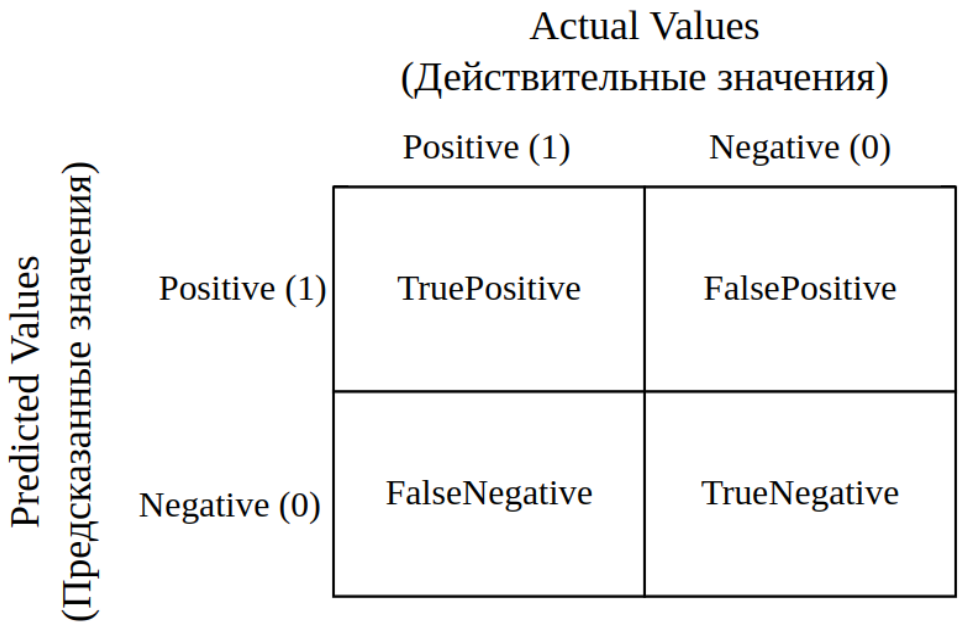

Рисунок 4.2 – Матрица ошибок

В случае распознавания объектов на изображениях показателем того, что рамка, полученная YOLO, соответствует рамке истинной, будет метрика Intersection over Union (рисунок 4.3):

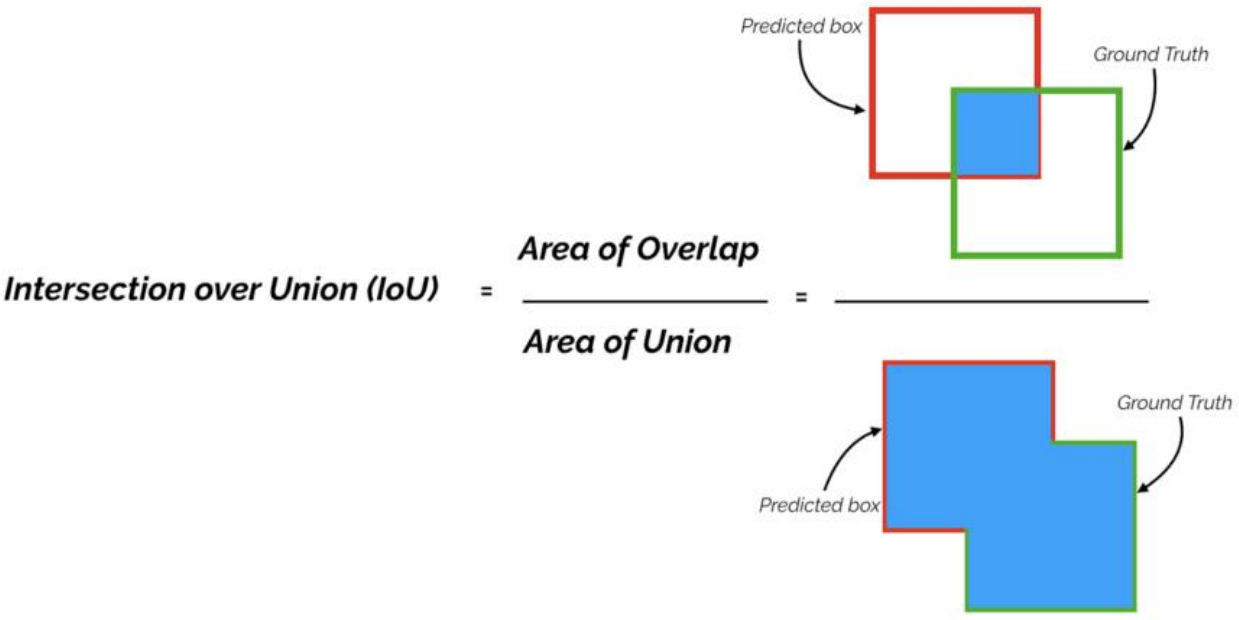

Рисунок 4.3 – Метрика IoU

Тогда матрицу ошибок можно интерпретировать следующим образом:

1) TruePositive увеличивается на единицу в том случае, когда пересечение истинной рамки и полученной от модели будет более 50%, то есть IoU ≥ 0.50.

2) FalsePositive увеличивается на единицу в том случае, когда модель находит рамку, которой на самом деле на изображении нет.

3) FalseNegative увеличивается на единицу в том случае, когда пересечение истинной рамки и полученной от модели будет меньше 50%, то есть IoU < 0,50, либо модель не находит рамку вообще.

4) TrueNegative = 0, так как в задачах детекции обнаруживаются только объекты, но не фон.

После подсчета вышеперечисленных характеристик находятся соответствующие метрики по формулам (4.1) – (4.4):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{TP}{TP + FP + FN}, \quad (4.1)$$

$$Precision = \frac{TP}{TP + FP}, \quad (4.2)$$

$$Recall = \frac{TP}{TP + FN}, \quad (4.3)$$

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \quad (4.4)$$

На основе полученных результатов можно оценить эффективность работы алгоритма.

4.1.2 Результаты тестирования

Так как задача заключалась в повышении точности распознавания небольших объектов на изображениях, начальной точкой считались результаты, полученные путем тестирования моделей Large, то есть моделей, которые были обучены на изображениях с дронами любого размера: и крупными, и средними, и малыми. Таким образом, для лучшего распознавания малоразмерных дронов необходимо к рамкам, получаемым от Large-модели, добавлять рамки от модели, специально обученной на маленьких объектах, то есть Small-модели.

Во всех алгоритмах A, B и C использовались результаты, полученные на датасете Large Dataset, поэтому вначале было проведено тестирование моделей Large на нем (таблица 4.1). На основе сравнения результатами, полученными на тестовой выборке этого датасета, оценивалась эффективность разработанных алгоритмов и делались выводы о прогрессе работы.

Таблица 4.1 – Результаты тестирования моделей Large на Large Dataset с confidence = 0,25 (default)

Кол-во пройденных эпох	**300**	**600**	**1000**
TruePositive	22 843	23 002	**23 024**
FalsePositive	1 594	1 138	**945**
FalseNegative	1 615	1 456	**1 434**
Accuracy	87,68%	89,87%	**90,63%**
Precision	93,48%	95,29%	**96,06%**
Recall	93,40%	94,05%	**94,14%**
F1-score	93,44%	94,66%	**95,09%**

Далее было проведено тестирование алгоритма C, который подразумевает использование модели Large, обученной 1000 эпох, давшей лучший результат на датасете Large (таблица 4.1), с повышением параметра confidence score с 0,25, до 0,625 (таблица 4.2).

Таблица 4.2 – Результаты тестирования алгоритма C

Confidence	**0,25**	**0,375**	**0,50**	**0,625**
TruePositive	**23 024**	22 974	22 928	22 828
FalsePositive	945	785	706	**642**
FalseNegative	**1 434**	1 484	1 530	1 630
Accuracy	90,63%	91,01%	**91,11%**	90,95%
Precision	96,06%	96,70%	97,01%	**97,26%**
Recall	**94,14%**	93,93%	93,74%	93,34%
F1-score	95,09%	95,29%	**95,35%**	95,26%

Затем был протестирован алгоритм A, результаты выполнения теста представлены в таблице 4.3. При тестировании использовались модели Large, давшая лучший результат на датасете Large (таблица 4.1), и Small, обученная на аугментированном Small Objects датасете, а также задавались ограничения для площади получаемых от Small-модели рамок и confidence score.

Таблица 4.3 – Результаты тестирования алгоритма A

Граница	**9000**			**13000**		
Confidence	**0,800**	**0,825**	**0,850**	**0,800**	**0,825**	**0,850**
TruePositive	23 049	23 046	23 037	**23 050**	23 047	23 037
FalsePositive	959	956	**953**	959	956	**953**
FalseNegative	1 409	1 412	1 421	**1 408**	1 411	1 421
Accuracy	90,68%	90,68%	90,66%	**90,69%**	**90,69%**	90,66%
Precision	96,01%	96,02%	**96,03%**	96,01%	96,02%	**96,03%**
Recall	**94,24%**	94,23%	94,19%	**94,24%**	94,23%	94,19%
F1-score	95,11%	95,11%	95,10%	**95,12%**	**95,12%**	95,10%

Далее был протестирован алгоритм B, в котором было реализовано разделение рамок по границе от двух моделей, результат приведен в таблице 4.4. В данном тесте использовались модели Large, давшая лучший результат на датасете Large (таблица 4.1), и Small, обученная на исходном Small Objects датасете, а также задавались ограничения для площади получаемых от обеих моделей рамок и confidence score.

Таблица 4.4 – Результаты тестирования алгоритма B

Confidence	**0,375**		**0,50**		**0,625**	
Граница	**2000**	**1000**	**2000**	**1000**	**2000**	**1000**
TruePositive	22 899	**22 902**	22 862	22 863	22 778	22 777
FalsePositive	917	834	834	752	727	**669**
FalseNegative	1 559	**1 556**	1 596	1 595	1 680	1 681
Accuracy	90,24%	90,55%	90,39%	**90,69%**	90,44%	90,65%
Precision	96,15%	96,49%	96,48%	96,82%	96,91%	**97,15%**
Recall	93,63%	**93,64%**	93,47%	93,48%	93,13%	93,13%
F1-score	94,87%	95,04%	94,95%	**95,12%**	94,98%	95,09%

Таким образом, были показаны ключевые варианты тестирования моделей.

4.2 Анализ полученных результатов

Исходя из результатов, полученных в предыдущем пункте при тестировании на Large Dataset моделей, обученных на 1000 эпох, для простоты выведем таблицу 4.5. Во второй колонке таблицы приведены результаты тестирования детектора YOLO12 на всем датасете Large (Large-модель, обученная 1000 эпох, давшая лучший результат в таблице 4.1) для сравнения с результатами, полученными разработанными алгоритмами A, B и C.

Таблица 4.5 – Лучшие результаты, полученные алгоритмами A, B и C.

	Детектор YOLO12	**Алгоритм A**	**Алгоритм B**	**Алгоритм C**
TruePositive	23 024	**23 050**	22 863	22 928
FalsePositive	945	959	752	**706**
FalseNegative	1 434	**1 408**	1 595	1 530
Accuracy	90,63%	90,69%	90,69%	**91,11%**
Precision	96,06%	96,01%	96,82%	**97,01%**
Recall	94,14%	**94,24%**	93,48%	93,74%
F1-score	95,09%	95,12%	95,12%	**95,35%**

При сравнении полученных результатов приходим к следующим выводам:

– наилучшие показатели accuracy, precision, F1-score получены при использовании алгоритма C;

– наилучший показатель recall получен при использовании алгоритма A;

– использование алгоритма A увеличило показатели модели Large из второй колонки таблицы 4.5: accuracy повысился на 0,06%, recall – на 0,1%, F1-score – на 0,03%;

– использование алгоритма A, по сравнению с моделью Large из второй колонки таблицы 4.5, увеличило количество правильно распознанных дронов (TruePositive) на 26 дронов, понизив тем самым количество нераспознанных объектов (FalseNegative) на 26 дронов соответственно, одновременно с этим повысив количество ложно обнаруженных (FalsePositive) объектов на 14 единиц;

– использование алгоритма B увеличило показатели модели Large из второй колонки таблицы 4.5: accuracy повысился на 0,06%, precision – на 0,76%, F1-score – на 0,03%;

– использование алгоритма B, по сравнению с моделью Large из второй колонки таблицы 4.5, уменьшило количество ложно обнаруженных (FalsePositive) дронов на 190, одновременно уменьшило количество правильно распознанных дронов (TruePositive) на 161 дрон, повысив тем самым количество нераспознанных объектов (FalseNegative) объектов на 161 единицу.

– использование алгоритма C увеличило показатели модели Large из второй колонки таблицы 4.5: accuracy повысился на 0,48%, precision – на 0,95%, F1-score – на 0,26%;

– использование алгоритма C, по сравнению с моделью Large из второй колонки таблицы 4.5, уменьшило количество ложно обнаруженных (FalsePositive) дронов на 239, одновременно уменьшило количество правильно распознанных дронов (TruePositive) на 96 дронов, повысив тем самым количество нераспознанных объектов (FalseNegative) объектов на 96 единиц.

Следует заметить, что, несмотря на наилучшие показатели применения алгоритма C, для реального обнаружения дронов предпочтительно использовать алгоритм A, который позволяет обнаруживать большее количество дронов.

4.3 Выводы

В четвертой главе были получены следующие основные результаты:

– описаны используемые метрики качества обнаружения объектов и их получение;

– проведен и описан процесс тестирования разработанных алгоритмов A, B и C;

– получены лучшие результаты тестирования разработанных алгоритмов;

– проведен анализ полученных результатов, подтверждающий улучшение точности распознавания объектов с помощью разработанных алгоритмов, в отличие от классического применения архитектуры YOLO.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы были выявлены основные проблемы обнаружения малоразмерных дронов на изображениях, а также повышения точности их распознавания. Основные проблемы связаны с размером объектов, их слабой отличимостью от фона, наличием других более заметных объектов, расстоянием, с которого проводится обнаружение.

На основе проблем была сформулированы следующие задачи: рассмотреть существующие алгоритмы распознавания объектов, обозначить причины снижения точности при обнаружении малоразмерных дронов, предложить и протестировать подходы на базе алгоритма YOLO, позволяющие улучшить точность детекции малоразмерных дронов, сравнить точности применения полученных алгоритмов, выделить наиболее эффективные.

В ходе работ был использован датасет изображений с дронами, предоставленный лабораторией обработки и распознавания изображений Объединенного института проблем информатики Национальной академии наук Беларуси.

Были разработаны три алгоритма, предназначенные для повышения точности обнаружения дронов, проведено обучение и тестирование различных архитектур нейросетевого детектора YOLO с целью выявления наиболее эффективного подхода к решению задачи.

Было выявлено, что для практического обнаружения дронов предпочтительно использовать разработанный алгоритм А, позволяющий обнаруживать большее количество дронов.

Полученные в ходе работы результаты предполагают дальнейшее улучшение алгоритма и последующее его использование в научных целях, а также в рабочих процессах различных организаций, область которых – компьютерное зрение, робототехника, системы безопасности или автоматизация процессов в промышленности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Computer Vision Algorithms / [Электронный ресурс] // GeeksforGeeks : [сайт]. – URL: https://www.geeksforgeeks.org/computer-vision-algorithms (дата обращения: 14.03.2025).

2. Goodfellow, I., Bengio, Y., Courville, A. Deep Learning [Текст] / I. Goodfellow, Y. Bengio, A. Courville // MIT Press. – 2016.

3. Huang, Z., Huang, L., Gong, Y., Huang, C., Wang, X. Mask Scoring R-CNN [Текст] / Z. Huang [и др.] // Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). – 2019.

4. Khan, S., Naseer, M., Hayat, M., Zhu, F., Porikli, F. Transformers in Vision: A Survey / [Электронный ресурс] S. Khan [и др.] // arXiv : [сайт]. – 2021. – URL: https://arxiv.org/abs/2101.01169 (дата обращения: 14.03.2025).

5. LeCun, Y., Bengio, Y., Hinton, G. Deep learning [Текст] / Y. LeCun, Y. Bengio, G. Hinton // Nature. – 2015. – Vol. 521, №7553. – С. 436–444.

6. Li, J., Liang, X., Shen, S., Xu, T., Feng, J., Yan, S. Scale-Aware Trident Networks for Object Detection / [Электронный ресурс] J. Li [и др.] // arXiv : [сайт]. – 2019. – URL: https://arxiv.org/abs/1901.01892 (дата обращения: 14.03.2025).

7. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A. C. SSD: Single Shot MultiBox Detector / [Электронный ресурс] W. Liu [и др.] // arXiv : [сайт]. – 2015. – URL: https://arxiv.org/abs/1512.02325 (дата обращения: 14.03.2025).

8. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection / [Электронный ресурс] J. Redmon [и др.] // arXiv : [сайт]. – 2015. – URL: https://arxiv.org/abs/1506.02640 (дата обращения: 14.03.2025).

9. Ren, S., He, K., Girshick, R., Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks / [Электронный ресурс] S. Ren [и др.] // arXiv : [сайт]. – 2015. – URL: https://arxiv.org/abs/1506.01497 (дата обращения: 14.03.2025).

10. Russell, S., Norvig, P. Artificial Intelligence: A Modern Approach [Текст] / S. Russell, P. Norvig. – 4th ed. – Pearson. – 2020.

11. Zhang, Y., Li, K., Li, K., Wang, L., Zhong, B., Fu, Y. Tiny Object Detection in Aerial Images / [Электронный ресурс] Y. Zhang [и др.] // arXiv : [сайт]. – 2020. – URL: https://arxiv.org/abs/2007.10603 (дата обращения: 14.03.2025).

ПРИЛОЖЕНИЕ А

ЛИСТИНГ ФАЙЛА IMAGES_SORT.IPYNB

```
import cv2
import os
import numpy as np
import shutil
def absolute_bbox_coords(width, height, values):
    _, x_center, y_center, bbox_width, bbox_height = values
    x_min = int((x_center - bbox_width / 2) * width)
    y_min = int((y_center - bbox_height / 2) * height)
    x_max = int((x_center + bbox_width / 2) * width)
    y_max = int((y_center + bbox_height / 2) * height)
    return x_min, y_min, x_max, y_max

def show_image(image_name, images_path, labels_path):
    image_path = os.path.join(images_path, image_name)
label_path = os.path.join(labels_path, image_name.replace('.png',
'.txt'))
    image = cv2.imread(image_path)
    if image is None:
      print(f"Ошибка загрузки изображения: {image_path}")
      return
    height, width = image.shape[:2]
    print(f"Размер изображения: {width}x{height}")
    with open(label_path, 'r') as f:
     for line in f.readlines():
      values = list(map(float, line.split()))
       if len(values) != 5:
        print("Неправильный формат строки:", line)
        continue
     x_min, y_min, x_max, y_max = absolute_bbox_coords(width,
height, values)
      if x_min < 0 or y_min < 0 or x_max > width or y_max > height:
      print(f"Некорректные координаты: {x_min}, {y_min}, {x_max},
{y_max}")
         continue
        a = x_max-x_min
        b = y_max-y_min
        print(f"Размер рамки: {a}x{b}. Площадь: {a * b}")
       cv2.rectangle(image, (x_min, y_min), (x_max, y_max), (0,
255, 0), 2)
    cv2.imshow('Image with Bounding Boxes', image)
```

```
    cv2.waitKey(0)
    cv2.destroyAllWindows()

def find_Small_bounding_boxes(images_path, labels_path, area_lim):
    Small_bounding_boxes = set()
    big_bounding_boxes = set()
    for label_file in os.listdir(labels_path):
      if label_file.endswith('.txt'):
        image_name = label_file.replace('.txt', '.png')
        image_path = os.path.join(images_path, image_name)
        image = cv2.imread(image_path)
        if image is None:
         print(f"Ошибка загрузки изображения: {image_path}")
         continue
        height, width = image.shape[:2]
        label_path = os.path.join(labels_path, label_file)
        with open(label_path, 'r') as f:  # Откр. файл меток
         for line in f.readlines():
          values = list(map(float, line.split()))
          if len(values) != 5:
           print("Неправильный формат строки:", line)
           continue
        x_min, y_min, x_max, y_max = absolute_bbox_coords (width,
height, values)
      if x_min < 0 or y_min < 0 or x_max > width or y_max > height:
       print(f"Некорректные координаты: {x_min}, {y_min}, {x_max},
{y_max}")
            continue
          s = (x_max - x_min) * (y_max - y_min)
          if s <= area_lim:
           small_bounding_boxes.add(image_name)
          else:
           big_bounding_boxes.add(image_name)
    return list(Small_bounding_boxes), list(big_bounding_boxes)

def create_dataset(new_dataset, images_path, labels_path, files,
sample_type):
  output_path = os.path.join(new_dataset, sample_type)
  if not os.path.exists(output_path):
    os.makedirs(output_path)
  images_output_path = os.path.join(output_path, 'images/')
  labels_output_path = os.path.join(output_path, 'labels/')
  os.makedirs(images_output_path, exist_ok=True)
  os.makedirs(labels_output_path, exist_ok=True)
```

```
  for image_name in files:
   source_image_path = os.path.join(images_path, image_name)
 destination_image_path     =     os.path.join(images_output_path,
image_name)
   if os.path.exists(source_image_path):
     if not os.path.exists(destination_image_path):
      shutil.copy(source_image_path, destination_image_path)
     else:
       print(f"Изобр. уже существует: {destination_image_path}")
   else:
     print(f"Изобр. не найдено: {source_image_path}")
   label_name = image_name.replace('.png', '.txt')
   source_label_path = os.path.join(labels_path, label_name)
 destination_label_path     =     os.path.join(labels_output_path,
label_name)
    if os.path.exists(source_label_path):
      if not os.path.exists(destination_label_path):
        shutil.copy(source_label_path, destination_label_path)
      else:
        print(f"Метка уже существует: {destination_label_path}")
    else:
       print(f"Метка не найдена: {source_label_path}")
print('Копирование в ', output_path, ' завершено')

def has_duplicate_filenames(directory):
    seen_filenames = set()
    for root, dirs, files in os.walk(directory):
        for filename in files:
            if filename in seen_filenames:
                return True
            seen_filenames.add(filename)
    return False

sample_types = ['train/', 'test/', 'valid/']
s_size = 9000
orig_dataset = 'Large_dataset/'
Small_obj_dataset = ('Small_obj_' + str(s_size) + '_dataset/')
big_obj_dataset = ('Big_obj_' + str(s_size) + '_dataset/')
path = '/Sviatlana/GraduationWork/datasets/'
orig_dataset_path = (path + orig_dataset)
Small_obj_dataset_path = (path + Small_obj_dataset)
big_obj_dataset_path = (path + big_obj_dataset)
for sample_type in sample_types:
 set_type = os.path.join(orig_dt_path, sample_type)
```

```
 images_path = os.path.join(set_type, 'images/')
 labels_path = os.path.join(set_type, 'labels/')
 Small_boxes, big_boxes = find_small_bounding_boxes (images_path,
labels_path, s_size)
create_dataset(Small_obj_dataset_path,  images_path,  labels_path,
Small_boxes, sample_type)
create_dataset(big_obj_dataset_path,   images_path,   labels_path,
big_boxes, sample_type)
```

ПРИЛОЖЕНИЕ Б

ЛИСТИНГ ФАЙЛА METRICS_TEST.IPYNB

```
import cv2
import numpy as np
import glob
import sys
import torch
import torchvision
import os
from ultralytics import YOLO

os.environ['YOLO_VERBOSE'] = 'False'
device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
print(device)

def ImageResizeAndTensor(ImagePath, New_width, New_Height, Nchan):
    H = New_Height
    W = New_width
    C = Nchan
    I = cv2.imread(ImagePath)
    height, width, chan = I.shape
    if C != chan:
        return(print('Incorrect Channel Number'))
    frackx = W/width
    fracky = H/height
    if frackx < fracky:
        w = int(width*fracky + 0.5)
        h = H
        J = cv2.resize(I, (w, h))
        JJ = J[0:H, 0:W]
    else:
        w = W
        h = int(height*frackx + 0.5)
        J = cv2.resize(I, (w, h))
        JJ = J[0:H, 0:W]
    JJJ = JJ.astype('float32')/127.5 - 1.
    h,w,c = JJ.shape
    if h != H or w != W or c!= 3:
        print('Image has strange shape')
    transform = torchvision.transforms.ToTensor()
    tens3d = transform(JJJ)
```

```
    tens4d = tens3d.unsqueeze(0)
    tens = tens4d.to('cuda')
    return JJ, tens, height, width

def GroundTruthFrameRead(FramePath):
    file = open(FramePath,'r')
    lines = file.readlines()
    file.close()
    a = []
    for line in lines:
        line = line.strip()
        info = line.split()
        a.extend(info)
    a = list(map(float,a))
    return a

def  GroundTruthFrameResize(bbrel,  width,  height,  New_width,
New_height):
    fx, fy = New_width/width, New_height/height
    k = len(bbrel)
    arrbox = []
    bbox = [0,0,0,0,0]
    if k > 0 and k%5 == 0:
        for iter in  range(0,k,5):
            if fx < fy:
                objecttype  = int(bbrel[iter] + 0.5)
                center_x    = int(bbrel[iter + 1] *width*fy + 0.5)
                center_y    = int(bbrel[iter + 2] *height*fy + 0.5)
                falf_width  = int(bbrel[iter + 3] *width*fy*0.5 +
0.5)
                falf_height = int(bbrel[iter + 4] *height*fy*0.5 +
0.5)
            else:
                objecttype  = int(bbrel[iter])
                center_x    = int(bbrel[iter + 1] *width*fx)
                center_y    = int(bbrel[iter + 2] *height*fx)
                falf_width  = int(bbrel[iter + 3] *width*fx*0.5)
                falf_height = int(bbrel[iter + 4]*height*fx*0.5)
            bbox[0] = objecttype
            bbox[1] = center_x - falf_width
            bbox[2] = center_y - falf_height
            bbox[3] = center_x + falf_width
            bbox[4] = center_y + falf_height
            arrbox.extend(bbox)
```

```
        return arrbox

def FindBoundBoxes(Res,thresh, SeedBright):
    h,w = Res.shape
    mask = np.ones((h,w), np.uint8)
    mask[1:h-1,1:w -1] = 0
    bboxes = []
    Sizes = []
    bbox = [0,0,0,0]
    for y in range(1,h-1):
        for x in range(1,w-1):
            stack = []
            if mask[y,x] == 0 and Res[y,x] >= SeedBright:
                mask[y,x] = 1
                stack.append([x,y])
                b = 0
                e = 1
                N = 1
                while b < e:
                    xx = stack[b][0]
                    yy = stack[b][1]
                    b += 1
                    if  mask[yy-1,xx] == 0 and Res[yy - 1,xx] >=
thresh:
                        stack.append([xx,yy - 1])
                        mask[yy-1,xx] = 1
                        e += 1
                        N += 1
                    if  mask[yy + 1,xx] == 0 and Res[yy + 1,xx] >=
thresh:
                        stack.append([xx,yy + 1])
                        mask[yy + 1,xx] = 1
                        e += 1
                        N += 1
                    if  mask[yy,xx+1] == 0 and Res[yy,xx + 1] >=
thresh:
                        stack.append([xx + 1,yy])
                        mask[yy,xx+1] = 1
                        e += 1
                        N += 1
                    if  mask[yy,xx-1] == 0 and Res[yy,xx - 1] >=
thresh:
                        stack.append([xx - 1,yy])
                        mask[yy,xx-1] = 1
```

```
                        e += 1
                        N += 1
                topx = w
                topy  = h
                botx = 0
                boty = 0
                for j in range(e):
                    a = stack[j][0]
                    b = stack[j][1]
                    if a < topx:
                        topx = a
                    if b < topy:
                        topy = b
                    if a > botx:
                        botx = a
                    if b > boty:
                        boty = b
                bbox[0] = topx
                bbox[1] = topy
                bbox[2] = botx + 1
                bbox[3] = boty + 1
                bboxes.extend(bbox)
                Sizes.append(N)
    return Sizes, np.array(bboxes)

def PostProcBboxes(bboxratio,bboxes):
    k = len(bboxes)
    if k == 0:
        return bboxes
    k = len(bboxes)
    iter = k - 4
    bboxes1 = bboxes
    while iter >= 0:
        x0 = bboxes[iter]
        y0 = bboxes[iter + 1]
        x1 = bboxes[iter + 2]
        y1 = bboxes[iter + 3]
        if x1 - x0 > bboxratio*(y1 - y0 + 1) or bboxratio*(x1 - x0
+ 1) < y1 - y0:
            bboxes1          =          np.delete          (bboxes,
[iter,iter+1,iter+2,iter+3])
        iter -= 4
    k = len(bboxes1)
    if k == 0:
```

```
        return bboxes1
    m = int(k/4)
    ind = np.zeros(m,int)
    for iter in range(0,k,4):
        l1x = bboxes1[iter]
        l1y = bboxes1[iter + 1]
        r1x = bboxes1[iter + 2]
        r1y = bboxes1[iter + 3]
        s = (r1x - l1x)*(r1y - l1y)
        for iter1 in range(0,k,4):
            if iter == iter1:
                continue
            l2x = bboxes1[iter1]
            l2y = bboxes1[iter1 + 1]
            r2x = bboxes1[iter1 + 2]
            r2y = bboxes1[iter1 + 3]
            if (l1x <= r2x and l2x <= r1x and l2y <=r1y and l1y <=
r2y):
                ss = (r2x - l2x)*(r2y - l2y)
                if ss < s:
                    ind[int(iter1/4)] = 1
                if ss > s:
                    ind[int(iter/4)] = 1
    iter = m - 1
    bboxes2 = bboxes1
    while iter >= 0:
        if ind[iter] == 1:
            n = 4*iter
            bboxes2 = np.delete(bboxes2, [n,n+1,n+2,n+3])
        iter -= 1
    return bboxes2

def Intersection(l1x,l1y,r1x,r1y,l2x,l2y,r2x,r2y):
    intersect = 0
    if l1x<=r2x and l2x<=r1x and l1y<=r2y and l2y<=r1y:
        intersect = (min(r1x,r2x) - max(l1x,l2x))*(min(r1y,r2y) -
max(l1y,l2y))
    return intersect

def IoU(l1x,l1y,r1x,r1y,l2x,l2y,r2x,r2y):
    intersect = Intersection(l1x,l1y,r1x,r1y,l2x,l2y,r2x,r2y)
    if intersect == 0:
        IoU = 0
    else:
```

```
        IoU = intersect/((r1x - l1x)*(r1y - l1y) + (r2x - l2x)*(r2y
- l2y) - intersect)
    return 100*IoU

def MatchGrTr_and_yolo(iouthresh, grtrframes, D):
    n_grtrframes = len(grtrframes) // 5
    m_yolo = len(D)
    grtrmatches = [0] * n_grtrframes
    CNNmatches = [0] * m_yolo
    for k in range(n_grtrframes):
        a = 0
        ind = -1
        for m in range(m_yolo):
            j1, j2 = k * 5, m * 4
            l1x, l1y, r1x, r1y = grtrframes[j1 + 1], grtrframes[j1
+ 2], grtrframes[j1 + 3], grtrframes[j1 + 4]
            l2x, l2y, r2x, r2y = D[m][0], D[m][1], D[m][2], D[m][3]
            iou = IoU(l1x, l1y, r1x, r1y, l2x, l2y, r2x, r2y)
            if iou > a:
                a = iou
                ind = m
        if a >= iouthresh:
            grtrmatches[k] = a
            CNNmatches[ind] = 1
    return grtrmatches, CNNmatches

def BoxS(box):
    x_min, y_min, x_max, y_max = box
    a = x_max-x_min
    b = y_max-y_min
    return a*b

PathTrainImages = '/Large_dataset/test/images/'
PathTrainFrames = '/Large_dataset/test/labels/'
full_img_pathes = glob.glob(PathTrainImages + '*.png')
sorted_full_img_pathes = sorted(full_img_pathes, key = len)
full_frame_pathes = glob.glob(PathTrainFrames + '*.txt')
sorted_full_frame_pathes = sorted(full_frame_pathes, key = len)

if len(sorted_full_img_pathes) == len(sorted_full_frame_pathes):
    print('Fine! Num of images = num of frames')
else:
    print('Bad! Num of images != num of frames')
    sys.exit()
```

```
print(len(sorted_full_img_pathes), len(sorted_full_frame_pathes))
NumberOfFiles = len(sorted_full_img_pathes)
New_width, New_height, Nchan = 640, 480, 3
TruePos, TrueNeg, FalsePos, FalseNeg = 0,0,0,0
model = YOLO('/yolo12_large_dt/weights/best.pt',verbose = False)
for iter in range(0, NumberOfFiles):
    imgpath = sorted_full_img_pathes[iter]
    filename = os.path.basename(imgpath).split('/')[-1]
    I, ImTensor, height, width = ImageResizeAndTensor( imgpath,
New_width, New_height, Nchan)
    results = model.predict(I, verbose = False)
    amount = len(results[0].boxes.data)
    D = []
    for j in range(amount):
        box = [int(results[0].boxes.data[j][0]),
               int(results[0].boxes.data[j][1]),
               int(results[0].boxes.data[j][2]),
               int(results[0].boxes.data[j][3])]
        D.append(box)
    groundtr_bbox_relative         =         GroundTruthFrameRead(
sorted_full_frame_pathes[iter])
    groundtr_bbox_absolut        =        GroundTruthFrameResize(
groundtr_bbox_relative, width, height, New_width, New_height)
    matchgrtr,       matchCNN       =       MatchGrTr_and_yolo(50,
groundtr_bbox_absolut, D)
    for j in range(len(matchgrtr)):
        if matchgrtr[j] > 0:
           TruePos += 1
        else:
            FalseNeg += 1
    for j in range(len(matchCNN)):
            FalsePos += 1 - matchCNN[j]

Precision = TruePos/( TruePos + FalsePos)
Recall = TruePos/( TruePos + FalseNeg)
Accuracy = TruePos/( TruePos + FalsePos + FalseNeg)
print('TP = ', TruePos, ', FP = ', FalsePos, ', FN = ', FalseNeg)
print('Acc = ',Accuracy,',Prec = ',Precision ,', Rec = ', Recall)
```