



Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:

Метод подсчёта количества человек на видео на основе сверточных
нейронных сетей

Студент

ИУ7И-82Б

(группа)

(подпись)

Фам Минь Хиен

(инициалы, фамилия)

Руководитель ВКР

(подпись)

Никульшина Т. А.

(инициалы, фамилия)

Нормоконтролер

(подпись)

Мальцева Д. Ю.

(инициалы, фамилия)

2025 год

РЕФЕРАТ

Расчетно-пояснительная записка 59 с., 24 рис., 3 табл., 27 источн., 2 прил.
ОБНАРУЖЕНИЕ ОБЪЕКТОВ, ПОДСЧЕТ, YOLO, ВИДЕО

В работе представлена разработка метода подсчета количества человек на видео на основе сверточных нейронных сетей.

В аналитическом разделе было приведено сравнение существующих методов нахождения объектов на изображении. Была сформулирована цель и представлена формализованная постановка задачи в виде IDEF0 диаграммы.

В конструкторском разделе рассмотрены требования, предъявляемые к разрабатываемому методу и к программному обеспечению, реализующему разработанный метод. Также в разделе были описаны основные этапы разработанного метода. Была представлена общая структура программного обеспечения.

В технологическом разделе были выбраны средства для реализации предлагаемого метода. Был описан формат входных и выходных данных. Была представлена реализация разработанного метода и проведено тестирование метода.

В исследовательском разделе проведены исследования зависимости точности разработанного метода от уровня освещенности и качества видео. Проведены исследования зависимости времени работы метода от качества видео.

СОДЕРЖАНИЕ

РЕФЕРАТ	5
ВВЕДЕНИЕ	8
1 Аналитический раздел	9
1.1 Задача нахождения объектов на изображении	9
1.2 Нейронные сети	10
1.2.1 Архитектура нейронной сети	10
1.2.2 Типы слоев и функции	11
1.2.3 Сверточная нейронная сеть	13
1.3 Существующие методы нахождения объекта на изображении . . .	15
1.3.1 Алгоритм YOLO	16
1.3.2 Алгоритм SSD	20
1.3.3 Модифицированные алгоритмы R-CNN	22
1.4 Сравнение рассматриваемых методов	25
1.5 Сравнение методов нахождения объектов семейства YOLO . . .	26
1.5.1 Метрики	26
1.5.2 Результаты сравнения	27
1.6 Формулировка цели	27
1.7 Формализация постановки задачи в виде IDEF0-диаграммы . . .	28
1.8 Вывод	29
2 Конструкторский раздел	30
2.1 Требование	30
2.2 Основные этапы разрабатываемого метода	30
2.2.1 Функциональная схема разрабатываемого метода	31
2.2.2 Раскадровка видео для обработки	33
2.2.3 Обнаружение и отслеживание человека на кадре	33
2.2.4 Определение зависимости положения человека относительно по- лиголов	35
2.2.5 Объединение результатов и отрисовка на кадре	37
2.3 Функциональная схема обучения модели	37
2.4 Структура программного обеспечения	38

2.5	Вывод	38
3	Технологический раздел	40
3.1	Выбор средств программной реализации	40
3.2	Описание формата входных и выходных данных	41
3.3	Сборка программного обеспечения	41
3.4	Демонстрация работы программы	42
3.5	Тестирование	43
3.6	Вывод	45
4	Исследовательский раздел	46
4.1	Технические характеристики	46
4.2	Зависимость точности метода от уровня освещенности	46
4.3	Зависимость точности метода от качества видео	48
4.4	Зависимость времени работы метода от качества видео	49
4.5	Вывод	49
ЗАКЛЮЧЕНИЕ		51
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ		54
ПРИЛОЖЕНИЕ А		55
ПРИЛОЖЕНИЕ Б		59

ВВЕДЕНИЕ

Компьютерное зрение — это одно из наиболее активно развивающихся направлений в области искусственного интеллекта, целью которого является автоматический анализ, интерпретация и понимание визуальной информации, получаемой с камер и других оптических устройств [1]. Благодаря достижениям в области машинного обучения, особенно в глубоких нейронных сетях, компьютерное зрение стало неотъемлемой частью множества прикладных задач: от распознавания лиц и автономного вождения до диагностики заболеваний по медицинским изображениям.

Одной из важных и практически значимых задач компьютерного зрения является подсчёт количества людей на изображениях и на видео [2]. Такая задача имеет широкий спектр применения — в системах видеонаблюдения и обеспечения безопасности, при мониторинге загруженности общественных пространств, на массовых мероприятиях, в торговых центрах и на транспортных узлах.

Цель работы — разработать метод подсчета количества человек на видео на основе сверточных нейронных сетей.

Для достижения поставленной цели требуется решить следующие задачи:

- рассмотреть существующие методы поиска объектов на изображениях;
- спроектировать и реализовать метод подсчета количества человек на видео на основе сверточных нейронных сетей;
- разработать программное обеспечение, реализующее предлагаемый метод;
- исследовать характеристики реализованного метода.

1 Аналитический раздел

1.1 Задача нахождения объектов на изображении

В области компьютерного зрения выделяют три основные задачи анализа изображений [3]:

1. Классификация заключается в определении, присутствует ли на изображении объект заданного класса, без указания его местоположения.
2. Обнаружение объектов направлено на выявление всех объектов на изображении с указанием их типов и точных координат.
3. Сегментация — детализированная задача, при которой каждому пикслю изображения присваивается метка, указывающая принадлежность к определенному объекту.

На рисунке 1.1 представлены основные задачи анализа изображений.

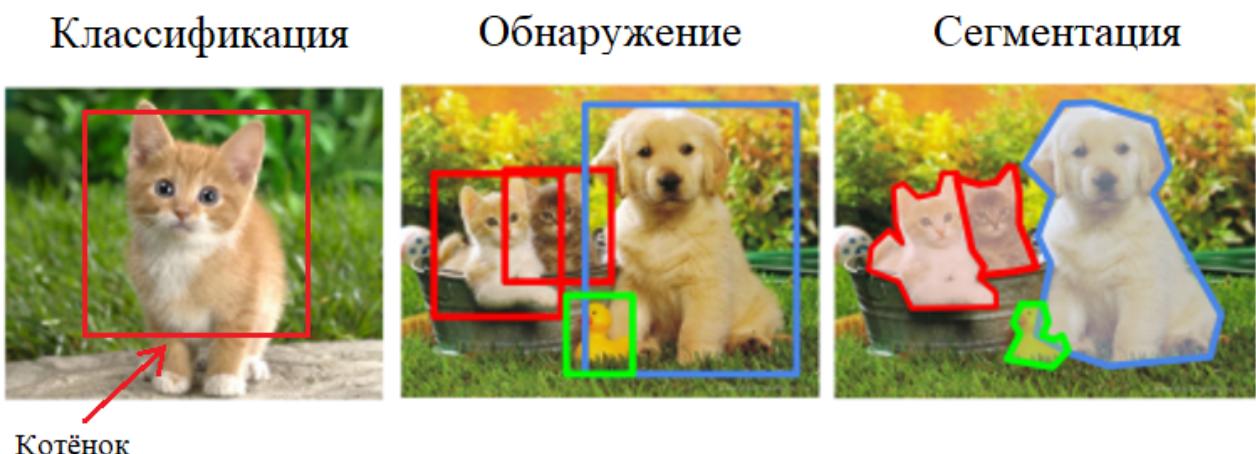


Рисунок 1.1 – Основные задачи анализа изображений [3]

Задача нахождения объектов на изображении относится ко второй категории — обнаружению. Ее суть заключается в определении местоположения и типа объектов, присутствующих на изображении. Необходимо не только установить факт наличия определенного объекта, но и точно указать его границы — обычно в форме прямоугольной области (bounding box), охватывающей объект. Таким образом, задача объединяет два компонента: локализацию и идентификацию объектов.

Нахождение объектов особенно важно при работе с изображениями, на которых могут одновременно присутствовать различные объекты — как по

типу, так и по количеству. Результатом решения задачи является список объектов, каждый из которых сопровождается меткой (классом) и координатами на изображении. Эти данные могут служить основой для дальнейшего анализа, автоматизированного управления, мониторинга, обеспечения безопасности и многих других практических приложений. В этом контексте задача нахождения объектов является важной ступенью на пути к машинному восприятию и интерпретации визуальной информации окружающего мира.

1.2 Нейронные сети

1.2.1 Архитектура нейронной сети

Нейронные сети вдохновлены принципами функционирования нейронных сетей человеческого мозга, где каждый «нейрон» получает сигналы от других нейронов, обрабатывает их и передает результат далее [4]. Каждый искусственный нейрон получает на вход взвешенную сумму входных сигналов, добавляет смещение и пропускает результат через функцию активации, чтобы ввести нелинейность.

$$y = \phi \left(\sum_{i=1}^n w_i x_i + b \right), \quad (1.1)$$

где x_i — входы, w_i — веса, b — смещение, ϕ — функция активации.

Нейронная сеть обычно состоит из трех основных типов слоев [5]:

1. Входной слой (input layer) — осуществляет приём информации из внешнего мира, такой как изображение в виде набора пикселей или текст в виде последовательности символов. Его задача — преобразовать исходные данные в структуру, подходящую для восприятия и обработки внутренними слоями нейронной сети.
2. Скрытый слой (hidden layer) — выполняет последовательную обработку входной информации, постепенно выделяя более абстрактные и значимые признаки, которые позволяют сети распознавать сложные зависимости и структуры в данных.
3. Выходной слой (output layer) — завершает работу сети, преобразуя обработанную скрытыми слоями информацию в итоговый результат.

На рисунке 1.2 представлена архитектура нейронной сети.

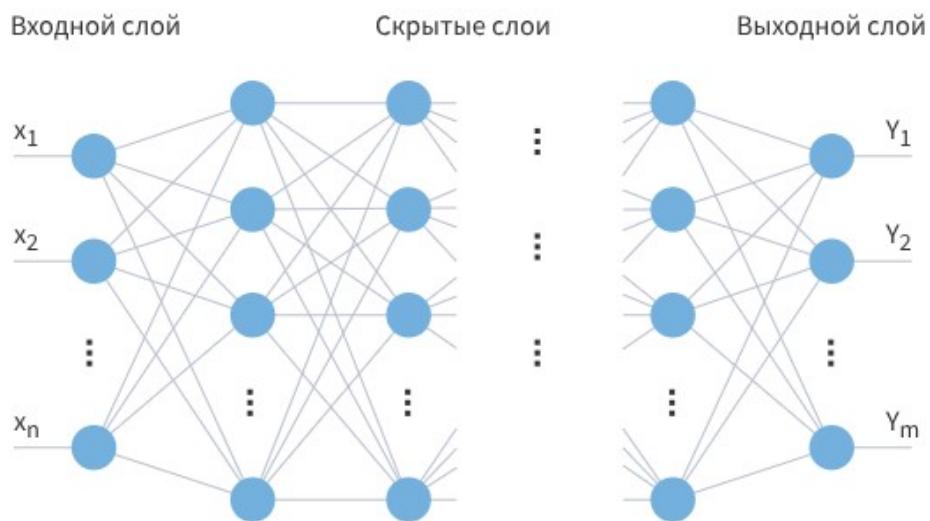


Рисунок 1.2 – Архитектура нейронной сети

1.2.2 Типы слоев и функции

Существует несколько типов слоев в нейросетях, каждый из которых выполняет свою уникальную функцию в обработке данных.

Основные типы слоев [5]:

1. Полносвязный (Fully Connected). Каждый нейрон соединен с каждым нейроном предыдущего слоя. Используется для обобщения признаков, особенно в завершающей части сети.
2. Сверточный слой (Convolutional Layer) применяет фильтры (ядра свертки) к входному изображению, выделяя локальные признаки. Если нейронная сеть имеет 1 сверточный слой, то такая сеть называется сверточной нейронной сетью.
3. Слой подвыборки (Pooling Layer) выполняет операцию агрегации (чаще всего max-pooling), уменьшая размерность признаков и ускоряя вычисления.
4. Рекуррентные слои (Recurrent Layer) — эти слои обрабатывают последовательные данные, например текст, и запоминают информацию о предыдущих элементах с помощью внутреннего состояния. Это позволяет учитывать контекст при обработке каждого следующего шага.

Функция активации — математическая функция, которая применяется к выходу каждого нейрона и позволяет ввести нелинейность в модель [6]. Без активационных функций нейронная сеть сводилась бы к обычному линейному преобразованию, неспособному решать сложные задачи, такие как распознавание образов, обработка изображений и классификация. Основная роль функции активации заключается в том, чтобы преобразовать взвешенную сумму входных сигналов нейрона в выходное значение, передаваемое следующему слою. Это позволяет сети строить сложные нелинейные зависимости и приближать любые функции, согласно теореме универсальной аппроксимации.

Основные типы функций активации [6]:

1. Сигмоида (Sigmoid). Эта функция вычисляется по формуле 1.2:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \quad (1.2)$$

Она ограничивает выходное значение диапазоном от 0 до 1. Наиболее часто используется в выходных слоях моделей для бинарной классификации.

2. Гиперболический тангенс (Tanh). Функция \tanh представляет собой центрированную версию сигмоиды:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (1.3)$$

Её значения лежат в диапазоне от -1 до 1 , что позволяет улучшить сходимость модели.

3. ReLU (Rectified Linear Unit). Одна из самых популярных и простых функций активации:

$$f(x) = \max(0, x). \quad (1.4)$$

Её преимущество — высокая скорость вычислений и эффективность в глубоких нейронных сетях. Недостаток — возможное вымирание нейронов при $x < 0$.

4. Leaky ReLU — модифицированная версия ReLU:

$$f(x) = \begin{cases} x, & x > 0, \\ \alpha x, & x \leq 0, \end{cases} \quad (1.5)$$

где $\alpha \in (0, 1)$, обычно $\alpha = 0.01$. Она сохраняет небольшую производную для отрицательных значений.

1.2.3 Сверточная нейронная сеть

Сверточная нейронная сеть (Convolutional Neural Network) — разновидность нейросетей для обработки данных с сеточной структурой: изображений и видео [7]. В отличие от полно связных сетей, где каждый нейрон соединен со всеми нейронами предыдущего слоя, CNN использует сверточные слои, которые обрабатывают локальные участки входных данных, что позволяет эффективно выделять пространственные признаки.

В сверточных нейронных сетях существуют два главных типа слоев: сверточные слои, которые отвечают за выделение признаков из изображения, и слои подвыборки (пулинга), которые сокращают размер данных, оставляя при этом наиболее важную информацию [7].

Сверточный слой — это основной элемент сверточной нейронной сети. Его задача — отфильтровать несущественные детали, такие как края изображения или мелкие линии, и сохранить лишь те признаки, которые действительно важны для распознавания объектов. Свертка позволяет выделять ключевые элементы изображения, необходимые для анализа. Такие признаки могут включать в себя формы, текстуры или цветовые особенности.

В сверточных слоях используются так называемые фильтры или ядра свертки — это небольшие матрицы, чаще всего размером 3×3 . С их помощью сеть анализирует изображение и выделяет важные признаки, такие как контуры или границы объектов. Работа фильтров происходит поэтапно: они последовательно перемещаются по изображению и вычисляют отклик на каждом участке, позволяя выявить характерные особенности структуры изображения.

На рисунке 1.3 представлена операция свертки в сверточных нейронных сетях.

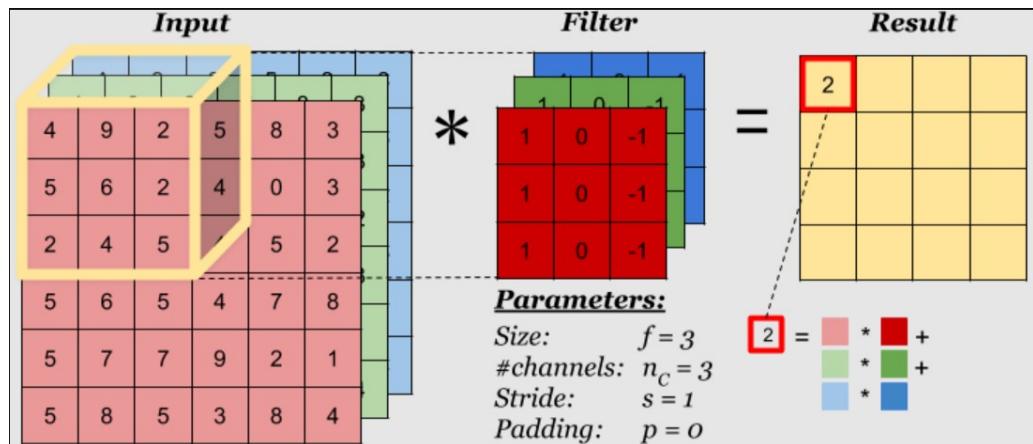


Рисунок 1.3 – Операция свертки

После сверточного слоя обычно следует пулинговый слой, который служит для уменьшения размеров карт признаков. Его задача — сохранить только наиболее значимую информацию и удалить лишние данные, тем самым снижая вычислительную нагрузку в последующих слоях.

Далее процесс может повторяться: снова применяется свертка, затем пулинг. Повторение этих этапов позволяет сети постепенно распознавать все более сложные признаки.

На рисунке 1.4 представлена операция подвыборки в сверточных нейронных сетях.

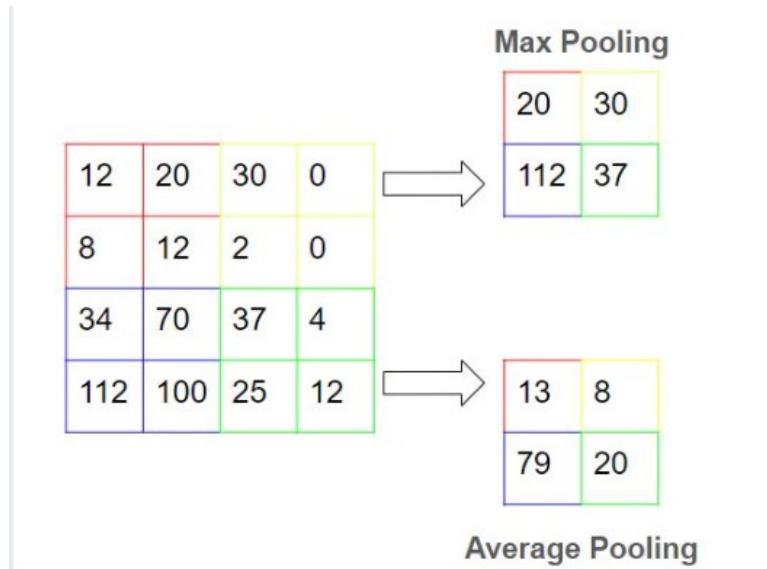


Рисунок 1.4 – Операция подвыборки [7]

1.3 Существующие методы нахождения объекта на изображении

Методы нахождения объектов делятся на одноэтапные и двухэтапные в зависимости от архитектуры алгоритма [8].

Одноэтапные методы как YOLO, SSD выполняют локализацию и классификацию объектов одновременно в рамках одной нейронной сети [9]. Их основным преимуществом является высокая скорость, что делает их пригодными для задач в реальном времени. Однако точность может быть ниже по сравнению с более сложными подходами.

Двухэтапные методы как Mask R-CNN, Fast R-CNN сначала находят области с потенциальными объектами (region proposals), а затем классифицируют их [10]. Такой подход обеспечивает высокую точность, особенно при наличии мелких или перекрывающихся объектов, но требует больше времени на обработку.

На рисунке 1.5 представлена архитектура этих методов.

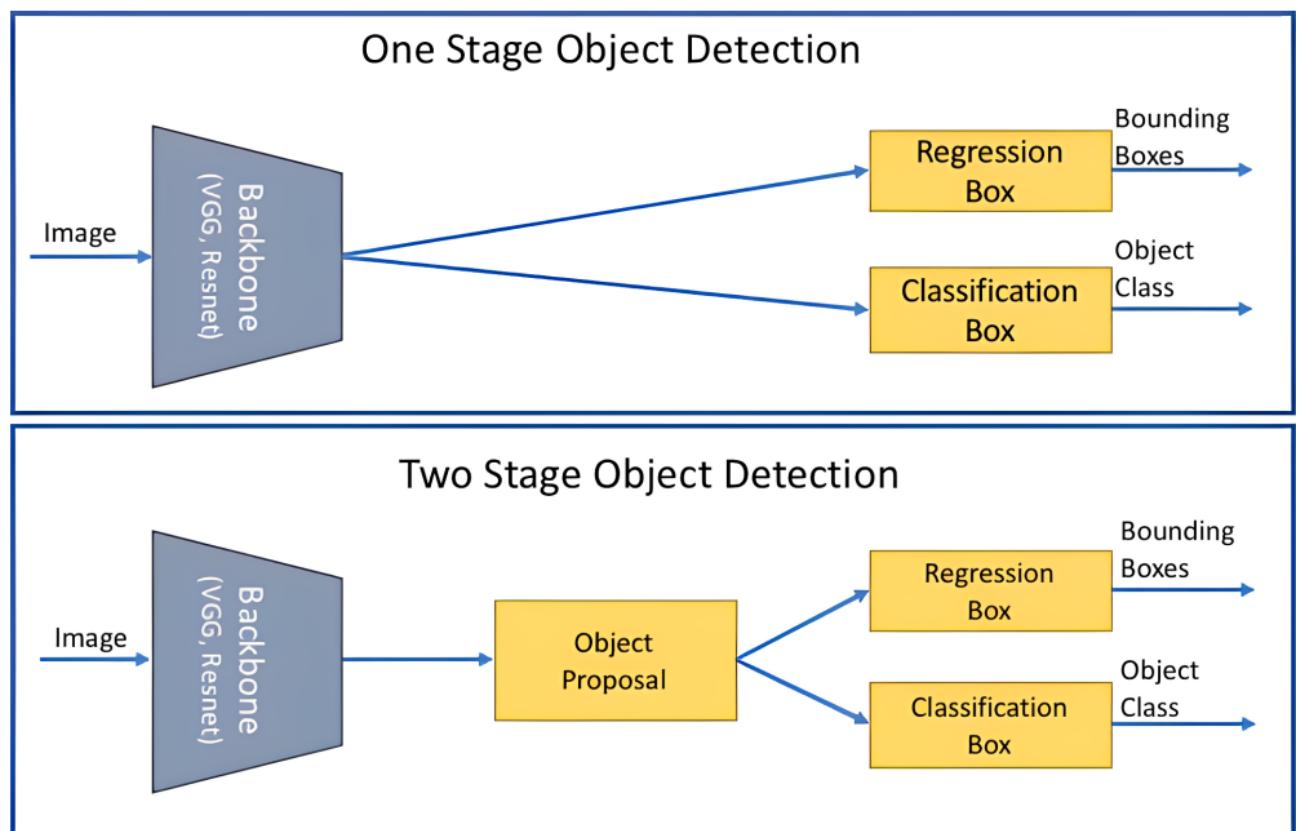


Рисунок 1.5 – Архитектура одноэтапных и двухэтапных методов

Оба подхода активно используются в практике. Выбор зависит от задачи:

для реального времени чаще применяются одноэтапные методы, для высокой точности — двухэтапные.

1.3.1 Алгоритм YOLO

YOLO — это один из самых популярных алгоритмов для обнаружения объектов в реальном времени. Он был разработан для решения задачи классификации объектов на изображениях с высокой скоростью и точностью [11]. YOLO продемонстрирован свою способность быстро и эффективно идентифицировать, находить и распознавать объекты на изображениях. Вместо того, чтобы обрабатывать изображение как одно изображение, YOLO делит изображение на сетку из небольших ячеек. Каждая ячейка сетки считается отдельным «частичным изображением» и может принадлежать другому классу объектов. Эта идея позволяет YOLO обрабатывать несколько объектов на изображении за одно сканирование.

Такой подход помогает YOLO добиться впечатляющей производительности и высокой точности. Он решает такие проблемы, как перекрывающиеся объекты, частично затемненные объекты, объекты, появляющиеся на изображении в разных положениях и размерах, а также наличие фона изображения.

YOLO использует поля привязки для измерения и прогнозирования положения и взаимного расположения объектов, что позволяет учитывать множество различных форм объектов. В то же время YOLO также использует технику немаксимального подавления, чтобы исключить ненужные результаты и сохранить только самые важные объекты.

YOLOv1

YOLOv1 — первоначальная модель YOLO рассматривала обнаружение объектов как проблему регрессии, что было значительным сдвигом от традиционного подхода к классификации.

На рисунке 1.6 представлена архитектура алгоритма YOLOv1.

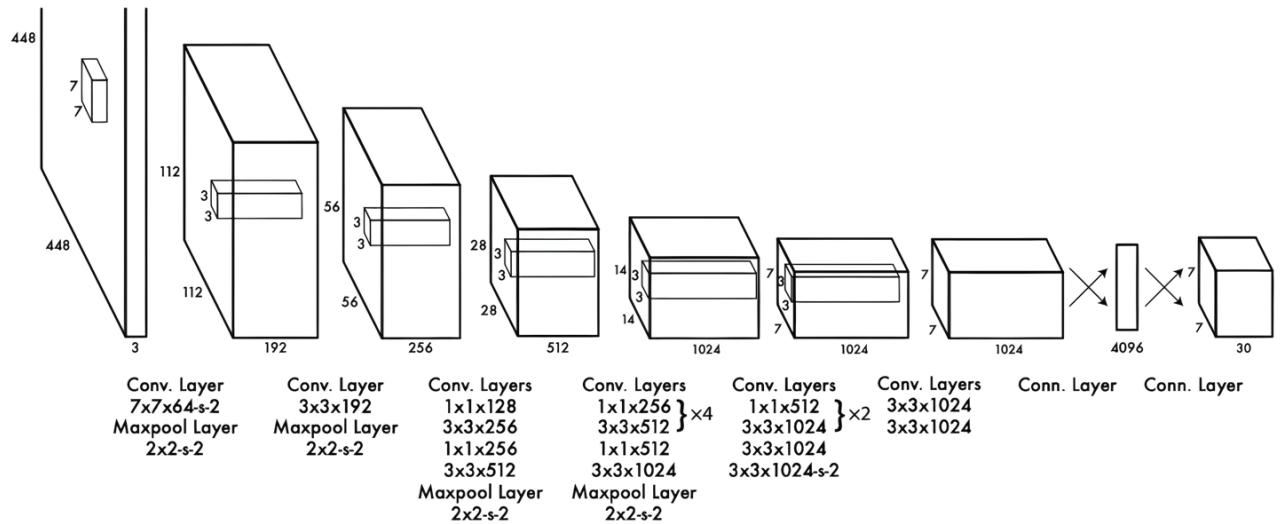


Рисунок 1.6 – Архитектура алгоритма YOLOv1 [12]

В ходе тестирования YOLOv1 происходит умножение условных вероятностей классов на прогнозируемые значения уверенности для отдельных ячеек. Это позволяет получить оценки для каждого блока, которые соответствуют конкретному классу, согласно формуле 1.6

$$Pr(Class_i|Object) \times Pr(Object) \times IOU_{pred}^{truth} = Pr(Class_i) \times IOU_{pred}^{truth}. \quad (1.6)$$

Она использовала одну сверточную нейронную сеть (CNN) для обнаружения объектов на изображениях, разделяя изображение на сетку, делая несколько прогнозов на ячейку сетки, отфильтровывая прогнозы с низкой достоверностью, а затем удаляя перекрывающиеся блоки для получения окончательного вывода.

YOLOv5

YOLOv5 представлен в четырех основных версиях: маленькая (s), средняя (m), большая (l) и очень большая (x), каждая из которых обеспечивает постепенно возрастающий уровень точности. При этом каждая из версий требует разного времени на обучение: чем больше и точнее модель, тем дольше процесс тренировки.

На рисунке 1.7 представлено сравнение разных версий YOLOv5.

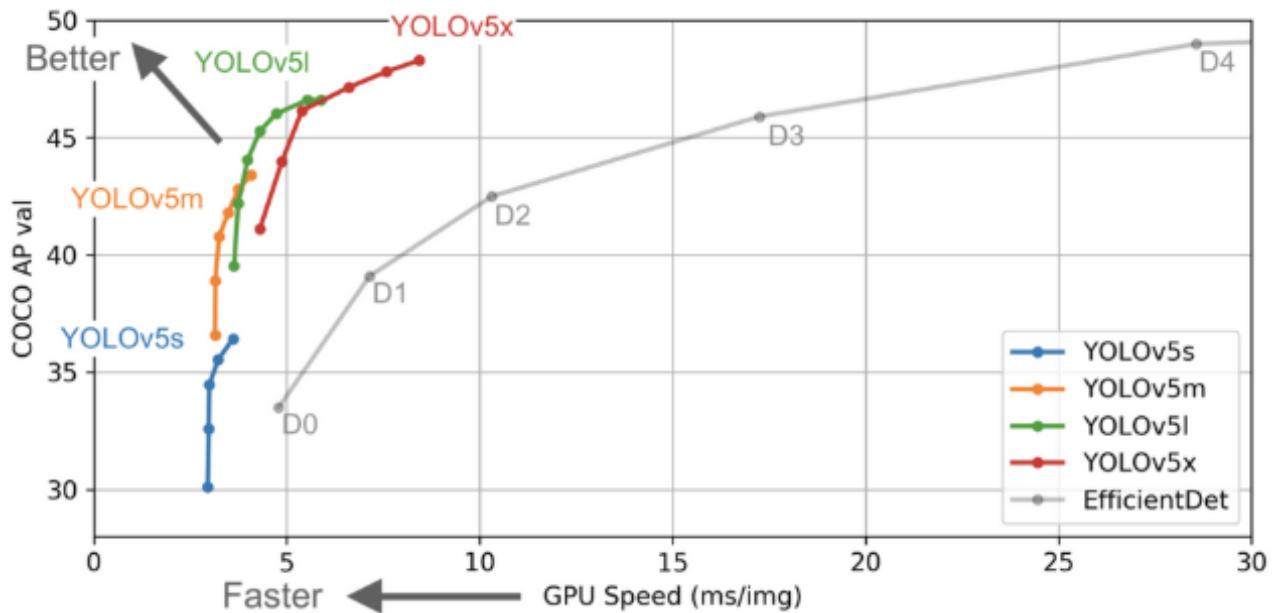


Рисунок 1.7 – Сравнение разных версий YOLOv5 [13]

Изображение проходит через входной слой (input) и передается в основную сеть (backbone) для извлечения признаков. Основная сеть получает карты признаков различных размеров, которые затем объединяются через сеть слияния признаков (neck), чтобы в итоге сформировать три карты признаков: P3, P4 и P5 (в YOLOv5 размеры выражаются как 80×80 , 40×40 и 20×20). Эти карты используются для обнаружения соответственно мелких, средних и крупных объектов на изображении.

После того как три карты признаков передаются в прогнозирующую голову (head), выполняются расчёт уровня уверенности (confidence) и регрессия ограничивающих рамок (bounding-box regression) для каждого пикселя карты признаков, используя заранее заданные якоря (prior anchors). Таким образом, получается многомерный массив (BBoxes), содержащий информацию о классе объекта, уровне уверенности класса, координатах рамки, ширине и высоте.

Затем, задавая соответствующие пороговые значения (confthreshold, objthreshold), фильтруется ненужная информация из массива. После этого выполняется процесс подавления немаксимумов (non-maximum suppression, NMS), чтобы получить финальную информацию об обнаруженных объектах.

Архитектура алгоритма YOLOv5 представлена на рисунке 1.8

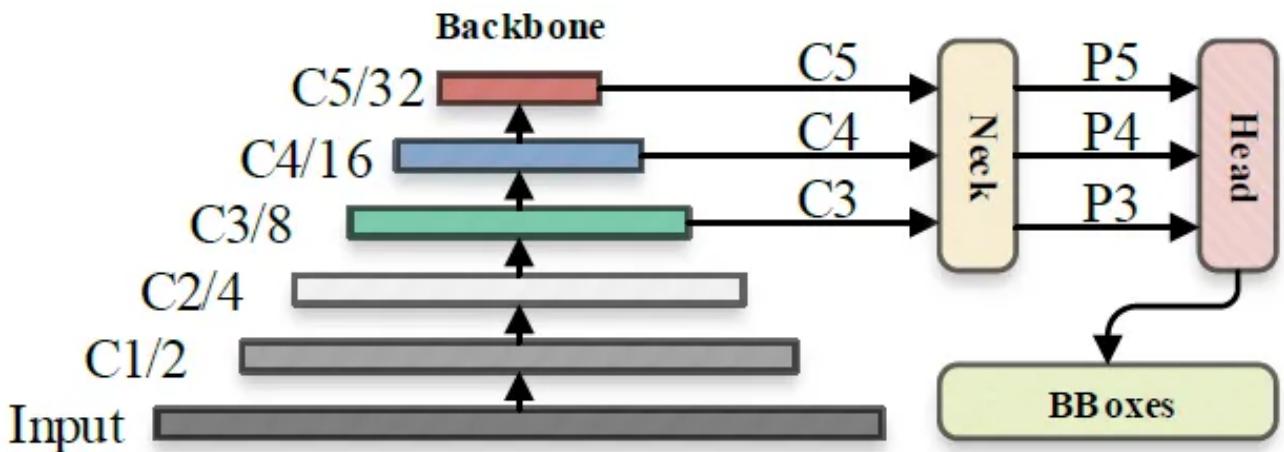


Рисунок 1.8 – Архитектура YOLOv5

YOLOv8

YOLOv8 — это нейросетевая архитектура, разработанная для решения задач компьютерного зрения, прежде всего — обнаружения объектов. Она представляет собой восьмую итерацию в развитии семейства YOLO, отличающуюся не только улучшенными характеристиками точности и скорости, но и архитектурными изменениями, направленными на повышение универсальности и адаптивности модели.

Основная идея, лежащая в основе всех моделей YOLO, заключается в том, чтобы выполнять обнаружение объектов в один проход нейросети. Вместо того чтобы сначала выделять регионы интереса, как в классических подходах типа R-CNN, модель YOLO делит изображение на сетку и одновременно предсказывает вероятности наличия объектов, их координаты и классы. Такой подход обеспечивает высокую скорость обработки — вплоть до реального времени — при сравнительно невысоких вычислительных затратах.

YOLOv8 внедряет ряд принципиальных улучшений по сравнению с предыдущими версиями [14]. Во-первых, в архитектуре модели реализован адаптивный Backbone — сеть, извлекающая признаки из изображения, которая была переработана для повышения плотности информации и устойчивости к искажениям. Во-вторых, улучшен Head-блок, отвечающий за генерацию финальных предсказаний, в том числе для задач не только обнаружения, но и сегментации и классификации.

С точки зрения функциональности YOLOv8 поддерживает задачи [14]:

- обнаружение объектов — локализация и классификация объектов на изображении;
- классификация изображений — определение класса всего изображения;
- сегментация объектов — идентификация и выделение отдельных объектов на изображении на уровне пикселей.

Ещё одной важной особенностью YOLOv8 является её модульная и масштабируемая архитектура. Это означает, что модель может быть адаптирована под различную вычислительную мощность (от мобильных устройств до серверных решений), а также под разные по сложности задачи. YOLOv8 поставляется в нескольких вариантах: от лёгких (YOLOv8n) до мощных и глубоких (YOLOv8x).

Кроме того, в YOLOv8 улучшены механизмы обобщения и регуляризации, что позволяет достигать высокой точности даже при наличии ограниченного объёма обучающих данных. Также реализована более эффективная система потерь (loss function), позволяющая быстрее и стабильнее обучать модель.

В целом, YOLOv8 представляет собой современное, гибкое и высокоэффективное решение для систем машинного зрения, сочетающее в себе преимущества быстродействия, точности и универсальности. Её теоретическая основа делает модель одной из наиболее применимых в таких сферах, как робототехника, медицина, безопасность и интеллектуальный анализ видеопотока.

1.3.2 Алгоритм SSD

SSD — это алгоритм для обнаружения объектов, который также ориентирован на высокую скорость и точность. Он был разработан для решения задачи локализации и классификации объектов на изображениях [15]. Основная идея SSD заключается в том, что он использует однопроходную нейронную сеть для предсказания ограничивающих рамок и классов объектов, что позволяет ему эффективно обрабатывать изображения.

На рисунке 1.9 представлена архитектура алгоритма SSD.

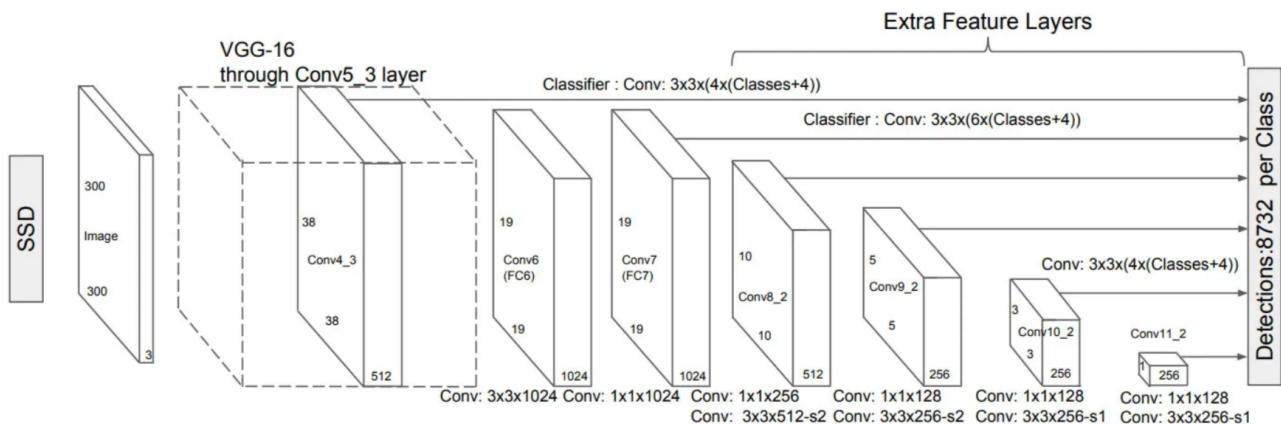


Рисунок 1.9 – Архитектура алгоритм SSD [16]

Основные характеристики алгоритма SSD можно описать следующими приведенными факторами:

1. Однопроходная архитектура: Как и YOLO, SSD выполняет обнаружение объектов за один проход через сеть, что значительно ускоряет процесс по сравнению с традиционными методами, требующими многократного анализа изображения.
2. Многоуровневая структура: SSD использует несколько уровней (или слоев) для предсказания объектов, что позволяет ему обнаруживать объекты разных размеров. На каждом уровне сети генерируются ограничивающие рамки и вероятности классов, что улучшает точность обнаружения.
3. Использование различных аспектов: Алгоритм SSD применяет различные соотношения сторон и масштабы для ограничивающих рамок, что позволяет ему более эффективно обнаруживать объекты с различными формами и размерами.
4. Быстрая обработка: SSD обеспечивает высокую скорость обработки, что делает его подходящим для приложений, требующих реального времени, таких как видеонаблюдение, автономные транспортные средства и мобильные устройства.
5. Обучение на больших наборах данных: SSD обучается на больших аннотированных наборах данных, что позволяет ему эффективно распо-

знавать различные классы объектов и адаптироваться к различным условиям.

1.3.3 Модифицированные алгоритмы R-CNN

R-CNN представляет собой значительный шаг вперед в области обнаружения объектов, который был предложен Россом Бахи и его коллегами в 2014 году [17]. Алгоритм использует двухступенчатый подход, который сочетает в себе методы селекции регионов и глубокое обучение. На первом этапе R-CNN генерирует набор предложений о регионах интереса (region proposals) с помощью алгоритма Selective Search, который выделяет потенциальные области, содержащие объекты. Эти регионы затем обрабатываются с использованием сверточной нейронной сети (CNN), что позволяет извлекать высокоуровневые признаки из каждого региона.

На втором этапе R-CNN применяет классификатор, обученный на извлеченных признаках, для определения класса объекта в каждом предложенном регионе. Для повышения точности алгоритм использует метод SVM (Support Vector Machine) для классификации и регрессию для уточнения границ объектов. Несмотря на свою эффективность, R-CNN имеет некоторые недостатки, такие как высокая вычислительная сложность и длительное время обработки, что связано с необходимостью обработки каждого региона отдельно [17]. Эти ограничения стали основой для разработки более совершенных моделей, таких как Fast R-CNN и Faster R-CNN, которые стремятся улучшить скорость и точность обнаружения объектов, сохраняя при этом преимущества, заложенные в оригинальной архитектуре R-CNN.

Алгоритм Fast R-CNN

Fast R-CNN — улучшенная версия R-CNN, разработанная Россом Гиршком, которая устраняет основные недостатки R-CNN, такие как высокая вычислительная сложность и длительное время обработки [18]. Fast R-CNN использует слой ROI Pooling для преобразования предложений регионов (ROIs) в векторы фиксированной длины, что позволяет выполнять сверточные операции один раз для всего изображения и делить результаты между всеми регионами.

Вместо трёх отдельных этапов (генерация регионов, извлечение призна-

ков и классификация) Fast R-CNN объединяет их в единую архитектуру. В результате модель работает быстрее и эффективнее. Однако Fast R-CNN всё ещё зависит от медленного алгоритма Selective Search для генерации предложений регионов, что ограничивает её производительность. Этот недостаток исправляется в Faster R-CNN, где вводится собственная сеть для генерации регионов.

Архитектура алгоритма Fast R-CNN представлена на рисунке 1.10

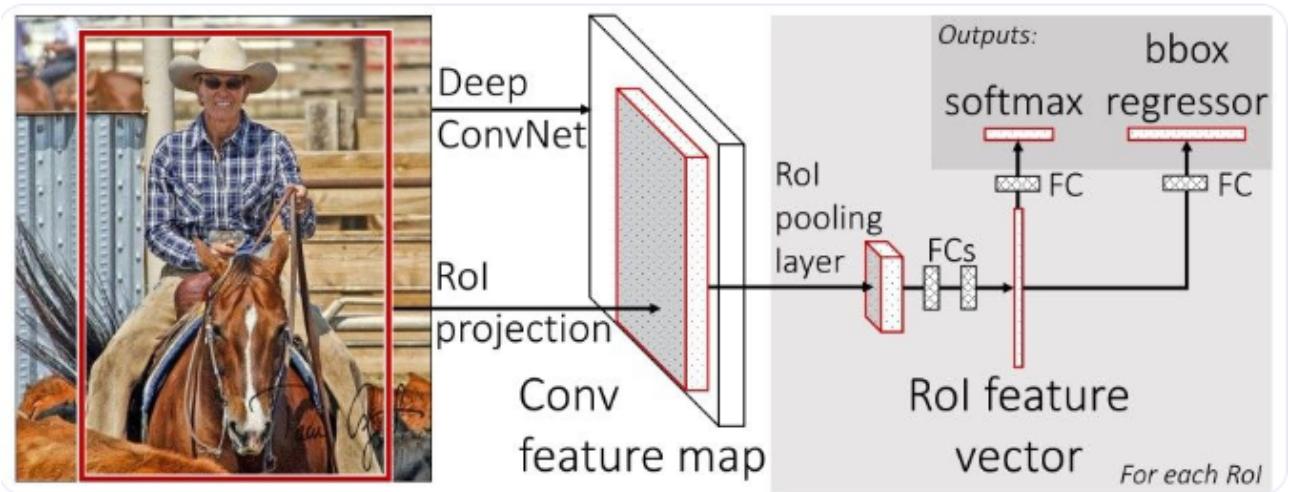


Рисунок 1.10 – Архитектура Fast R-CNN [18]

Алгоритм Mask R-CNN

Mask R-CNN — это нейронная сеть, предназначенная для задачи сегментации объектов (instance segmentation) в компьютерном зрении [19]. Она может выделять отдельные объекты на изображении или видео, предоставляя ограничивающие рамки (bounding boxes), классы объектов и их маски на уровне пикселей.

Архитектура алгоритма Mask R-CNN представлена на рисунке 1.11

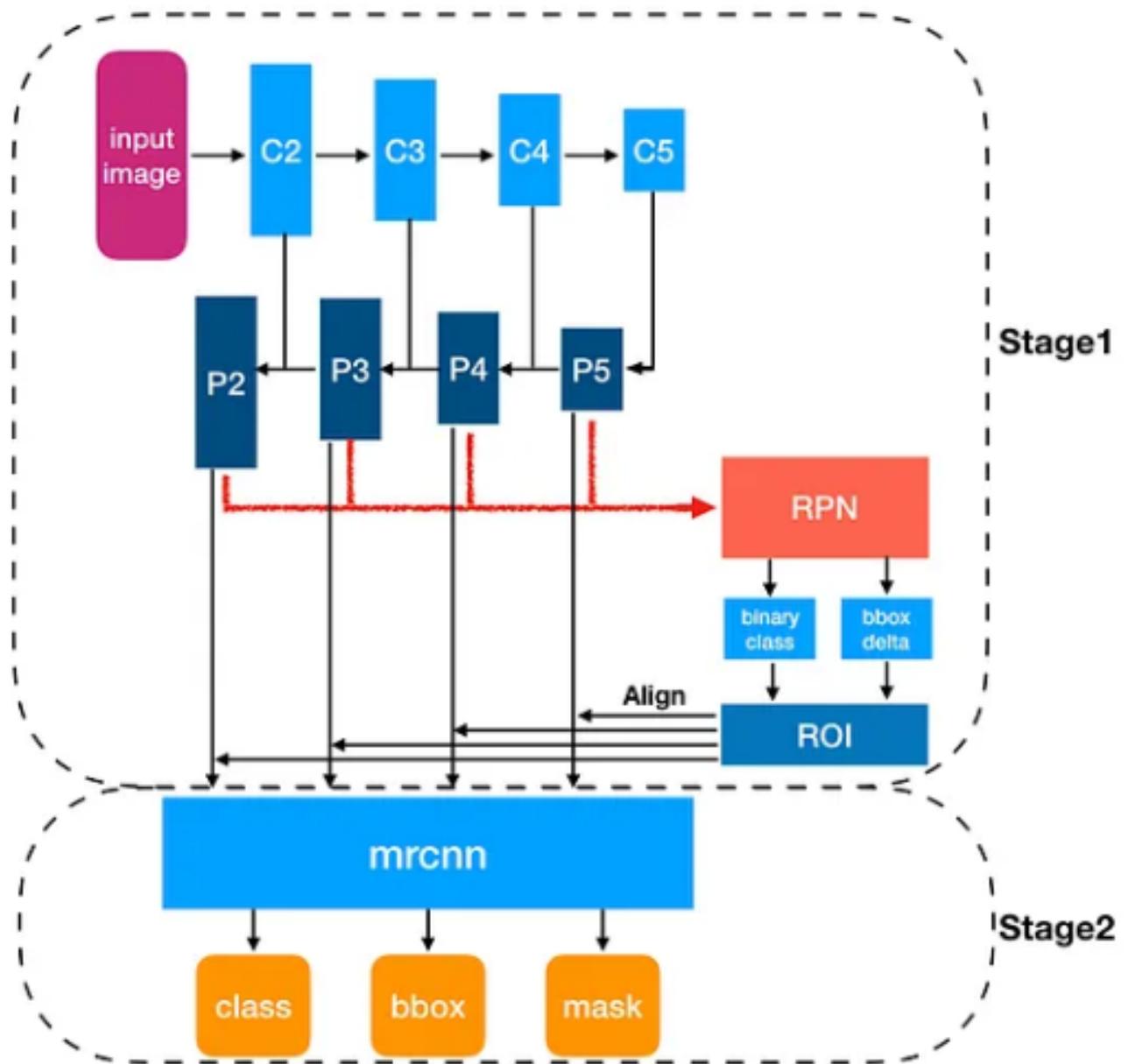


Рисунок 1.11 – Архитектура Mask R-CNN

Архитектура Mask R-CNN состоит из двух этапов. На первом этапе сеть генерирует предложения регионов (Region Proposal Network, RPN), которые могут содержать объекты. Для этого используется Feature Pyramid Network (FPN), которая обеспечивает извлечение признаков на разных масштабах. FPN включает в себя сверточную сеть (например, ResNet или VGG) для анализа изображения и механизм, сохраняющий информацию на разных уровнях разрешения.

На втором этапе сеть уточняет полученные регионы, определяет классы объектов, уточняет координаты ограничивающих рамок и создает маски

объектов. В отличие от первого этапа, на этом этапе используется метод ROIAlign, который позволяет точно сопоставить области на карте признаков с исходным изображением, обеспечивая высокую точность результатов [19].

Главное преимущество Mask R-CNN — способность работать с объектами на разных масштабах благодаря архитектуре FPN и использованию якорей (anchors). Это позволяет сохранять пространственные отношения между объектами на исходном изображении и их признаками на карте. Mask R-CNN активно используется в задачах медицинской диагностики, автономного вождения и видеоаналитики.

1.4 Сравнение рассматриваемых методов

В качестве критерии для сравнения вышеперечисленных методов выделены следующие признаки:

- является ли алгоритм одноэтапным;
- средняя точность обнаружения объектов (mAP);
- скорость работы (FPS).

В таблице 1.1 представлено сравнение рассматриваемых методов. Обозначения «Fast» и «Mask» в таблице соответствуют алгоритмам Fast R-CNN и Mask R-CNN соответственно.

Таблица 1.1 – Сравнение алгоритмов обнаружения объектов

Критерий	YOLO	SSD	Fast R-CNN	Mask R-CNN
Одноэтапная архитектура	Да	Да	Нет	Нет
Средняя точность ($mAP \geq 0.7$ на основании датасета COCO)	Нет	Нет	Да	Да
Скорость работы (FPS)	50–150	25–60	≤ 15	≤ 15

Из представленного сравнения видно, что алгоритм **YOLO** превосходит остальные методы по скорости работы, что делает его оптимальным выбором для задач в реальном времени. В то же время **Fast R-CNN** и особенно **Mask R-CNN** демонстрируют более высокую точность, однако требуют значительно больше времени на обработку. Алгоритм **SSD** занимает промежуточное

положение — он быстрее, чем R-CNN-методы, но уступает им по точности, и наоборот — точнее, чем YOLO, но медленнее.

Таким образом:

- **YOLO** — лучший выбор для практических приложений с ограничениями по времени и ресурсам;
- **Fast R-CNN** и **Mask R-CNN** — предпочтительны, если приоритетом является качество детекции;
- **SSD** — компромиссный вариант между точностью и быстродействием.

1.5 Сравнение методов нахождения объектов семейства YOLO

1.5.1 Метрики

Для сравнения рассматриваемых методов будут использоваться следующие критерии [20]:

1. Точность mAP (IoU 0.5—0.95).
2. Скорость (ms).
3. Количество параметров (M).
4. Количество операций (FLOPs).

Критерий mAP@(0.5:0.95) — это усреднённая метрика точности, рассчитываемая как среднее значение AP при разных порогах IoU от 0.5 до 0.95 с шагом 0.05. Критерий «mAP» вычисляется по следующей формуле 1.7.

$$mAP = \sum_{i=1}^N AP_i/N, \quad (1.7)$$

где N — количество классов, AP (average precision) — метрика, используемая для оценки точности работы детекторов объектов.

Скорость показывает, сколько миллисекунд требуется модели в формате ONNX для обработки одного изображения на процессоре (CPU), что важно для оценки производительности на реальных устройствах без использования

GPU. Количество параметров (M) показывает общий объём обучаемых весов модели в миллионах, определяя её размер и ресурсоемкость. Количество операций (FLOPs) отражает количество вычислений с плавающей запятой, необходимых модели для обработки одного изображения, характеризуя её вычислительную сложность.

1.5.2 Результаты сравнения

На основе вышеперечисленных метрик приведено сравнение разных версий YOLO. Замеры проводились на основе датасета COCO [21]. Результаты, представленные в таблице 1.2, были получены при использовании входных изображений размером 640×640 пикселей.

Таблица 1.2 – Сравнение моделей YOLOv5 и YOLOv8 по метрикам [20]

Модель	mAP IoU(50–95)	Speed (ms)	Параметры (M)	FLOPs (B)
YOLOv5n	28.0	73.6	2.6	7.7
YOLOv5m	45.4	233.9	25.1	64.2
YOLOv5x	50.7	763.2	97.2	246.4
YOLOv8n	37.3	80.4	3.2	8.7
YOLOv8m	50.2	234.7	25.9	78.9
YOLOv8x	53.9	479.1	68.2	257.8

Модель YOLOv8n представляется собой как оптимальная модель для данной работы, обеспечивающая баланс между точностью распознавания и вычислительной эффективностью. Согласно таблице 1.2, она значительно превосходит YOLOv5n по метрике mAP IoU(50–95) (37.3 против 28.0), при незначительном увеличении числа параметров (3.2M) и FLOPs (8.7B). Время обработки одного изображения составляет всего 80.4 мс, что делает модель подходящей для задач в реальном времени и на устройствах с ограниченными ресурсами.

1.6 Формулировка цели

В ходе выполнения работы требуется разработать метод автоматического подсчёта количества людей на видео с использованием сверточных нейронных сетей. Метод должен удовлетворять следующим требованиям:

- для достижения поставленной цели должны быть использованы сверточные нейронные сети.

- обеспечение устойчивой работы на видеозаписях с различными условиями съёмки (освещение, перспектива, плотность толпы);
- поддержка обработки видео в реальном времени;
- предусмотрено использование предобученных архитектур сверточных нейронных сетей с возможностью их адаптации под конкретные датасеты;

1.7 Формализация постановки задачи в виде IDEF0-диаграммы

С целью формализации основной задачи разработки метода подсчёта количества человек на видео с использованием сверточных нейронных сетей была составлена IDEF0-диаграмма нулевого уровня. Она позволяет отразить ключевые функции системы на абстрактном уровне, определить входные и выходные данные, а также средства управления и механизмы, задействованные в процессе. Такая диаграмма служит основой для дальнейшей декомпозиции задач и построения более детализированной архитектуры системы.

IDEF0-диаграмма нулевого уровня представлена на рисунке 1.12

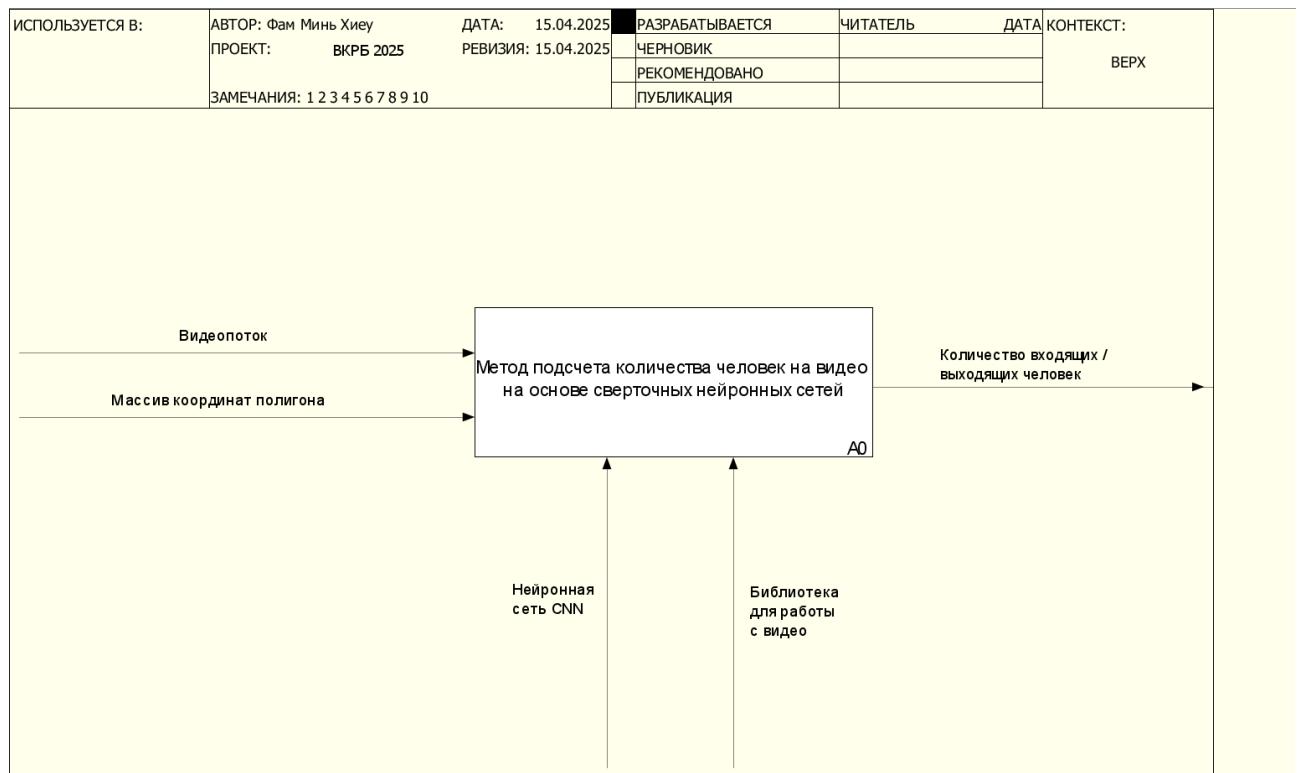


Рисунок 1.12 – IDEF0-диаграмма нулевого уровня

1.8 Вывод

В этом разделе рассмотрены фундаментальные понятия предметной области и выполнен обзор современных методов распознавания объектов. Кроме того, были определены ключевые критерии оценки и выделены характерные особенности анализируемых подходов.

Результаты анализа показали, что алгоритмы семейства YOLO демонстрируют наилучшее соотношение между точностью, скоростью и требуемыми вычислительными ресурсами, что обуславливает выбор именно этого подхода. Из доступных версий архитектуры YOLO была выбрана модель YOLOv8n, поскольку она демонстрирует достаточную точность распознавания объектов при невысоком потреблении вычислительных ресурсов.

Сформулирована цель и представлена формализованная постановка задачи в виде IDEF0-диаграммы.

2 Конструкторский раздел

2.1 Требование

Требование к разрабатываемому методу

К методу подсчета количества человек на видео на основе сверточных нейронных сетей предъявляются следующие требования:

- поддержка популярных форматов: mp4, avi, mov, и др;
- использование сверточных нейронных сетей для обнаружения объектов;
- возможность отслеживания перемещающихся объектов.

В рамках проектирования и реализации разрабатываемого метода был сформулирован и представлен ряд ограничений:

- при высокой плотности толпы возможны ошибки подсчета: одна фигура может быть распознана дважды, или наоборот — несколько людей как один объект;
- при подвижной камере подсчёт может стать нестабильным без дополнительных алгоритмов стабилизации.

Требование к программному обеспечению

Программное обеспечение, реализующее интерфейс к разработанному методу, должно позволять выполнять следующие действия:

- возможность загрузки видео с помощью интерфейса;
- возможность задать полигонов для алгоритма подсчета;
- возможность просмотра результата работы метода.

2.2 Основные этапы разрабатываемого метода

Разрабатываемый метод представляет собой последовательную и структурированную процедуру анализа видеопотока, направленную на автоматический подсчет количества человек в кадре с использованием сверточных

нейронных сетей. Основной целью метода является не только фиксация факта присутствия человека, но и точное определение его положения относительно заранее заданных зон интереса — так называемых полигонов. Эти зоны, устанавливаемые пользователем на каждом видеокадре, служат границами для подсчёта входящих и выходящих человек. Метод включает в себя этапы предварительной обработки видеоданных, детектирования человек, их идентификации и отслеживания по последовательности кадров.

2.2.1 Функциональная схема разрабатываемого метода

На рисунке 2.1 представлена IDEF0-диаграмма первого уровня, отражающая более детализированное представление функциональной структуры разрабатываемого метода подсчёта количества человек на видео на основе сверточных нейронных сетей. Данная диаграмма демонстрирует декомпозицию основной функции верхнего уровня на несколько подфункций, описывающих отдельные этапы обработки видео.

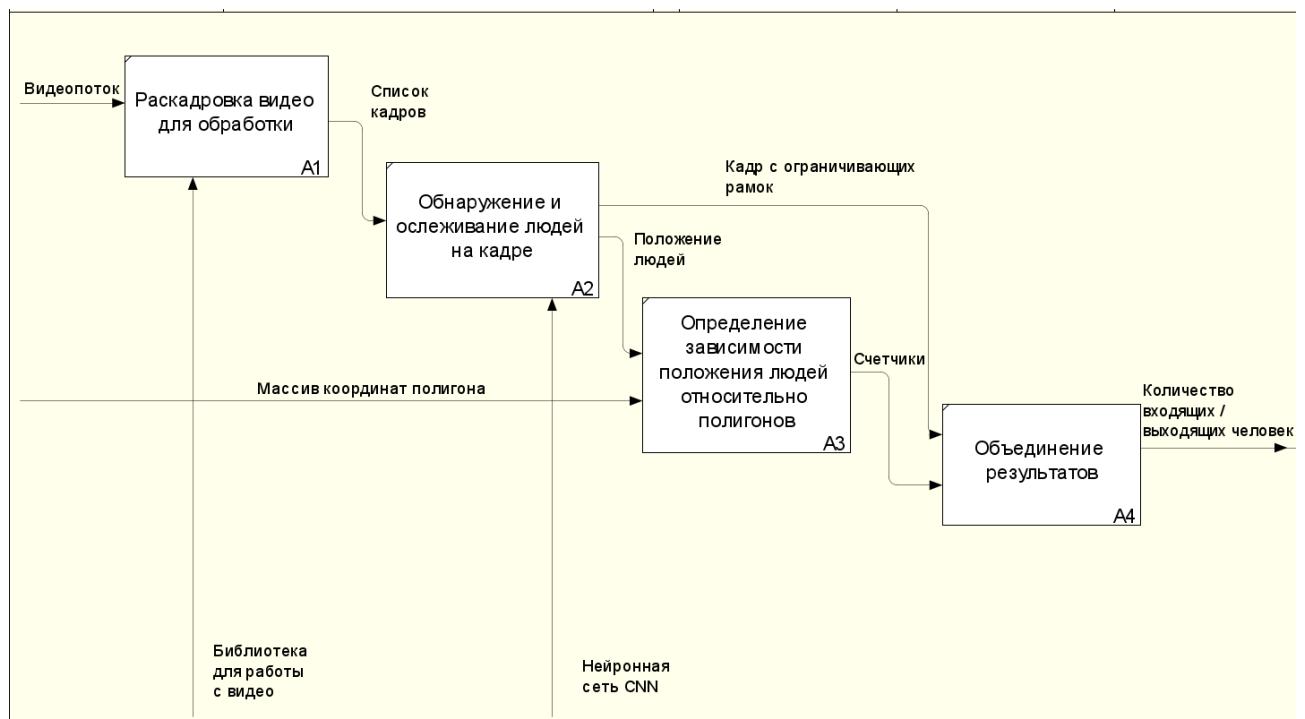


Рисунок 2.1 – IDEF0-диаграмма первого уровня

Для детализации логики функционирования системы, был разработан алгоритм, представленный в виде блок-схемы на рисунке 2.2.

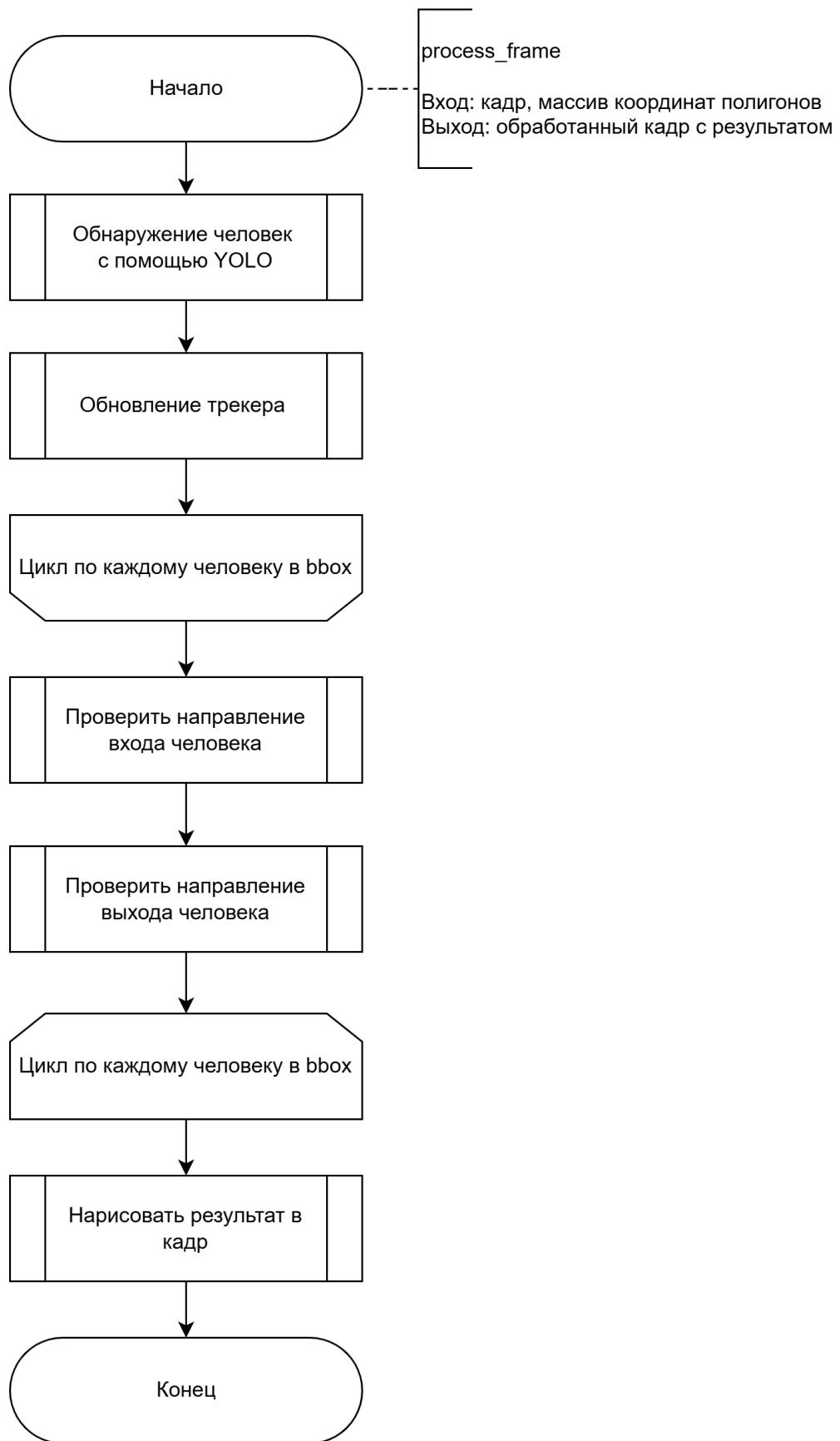


Рисунок 2.2 – Алгоритм работы разрабатываемого метода

2.2.2 Раскадровка видео для обработки

На первом этапе обработки входной видеопоток проходит процедуру раскадровки, которая заключается в его разбиении на последовательность отдельных статичных изображений — кадров. Эта операция является важной, поскольку именно покадровый подход позволяет эффективно применять современные алгоритмы компьютерного зрения, большинство из которых ориентированы на анализ неподвижных изображений. Раскадровка обеспечивает переход от непрерывного потока информации к дискретной форме представления данных, что упрощает задачу обработки и анализа.

Каждый извлечённый кадр далее обрабатывается независимо, что позволяет выявлять объекты на каждом изображении, отслеживать их положение во времени и проводить дальнейшие вычисления, необходимые для реализации метода подсчёта количества человек.

2.2.3 Обнаружение и отслеживание человека на кадре

Следующий этап включает в себя обнаружение людей на каждом кадре с последующим отслеживанием их перемещений. Для решения задачи обнаружения применяется современная модель YOLOv8n. Эта нейросетевая модель позволяет с высокой точностью находить людей на изображении и локализовать их в виде ограничивающих прямоугольников.

После первичного обнаружения объектов активируется модуль трекинга (отслеживания), задача которого — сохранять уникальные идентификаторы обнаруженных людей при их перемещении по кадрам. Это необходимо для предотвращения повторного учета одного и того же объекта.

На рисунке 2.3 представлен алгоритм отслеживания перемещающихся объектов.

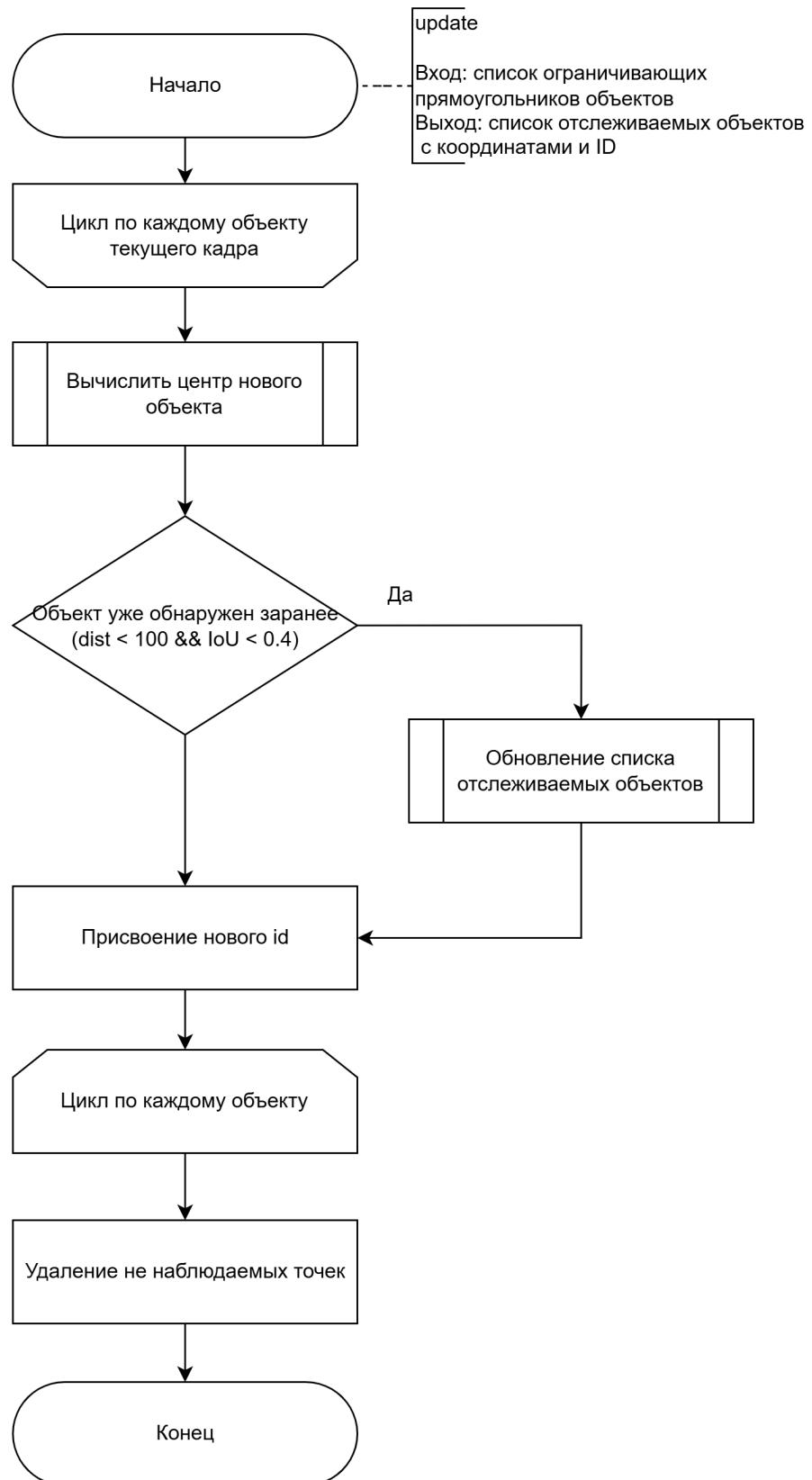


Рисунок 2.3 – Алгоритм отслеживания перемещающихся объектов

2.2.4 Определение зависимости положения человека относительно полигонов

На данном этапе определяется зависимость положения человека относительно полигонов. Для каждого отслеживаемого человека вычисляется его положение — обычно используется нижний правый угол рамки, ограничивающей объект. Затем проверяется, попадает ли эта точка в один из заранее заданных полигонов.

На рисунке 2.4 представлен алгоритм подсчета количества входящих человек.

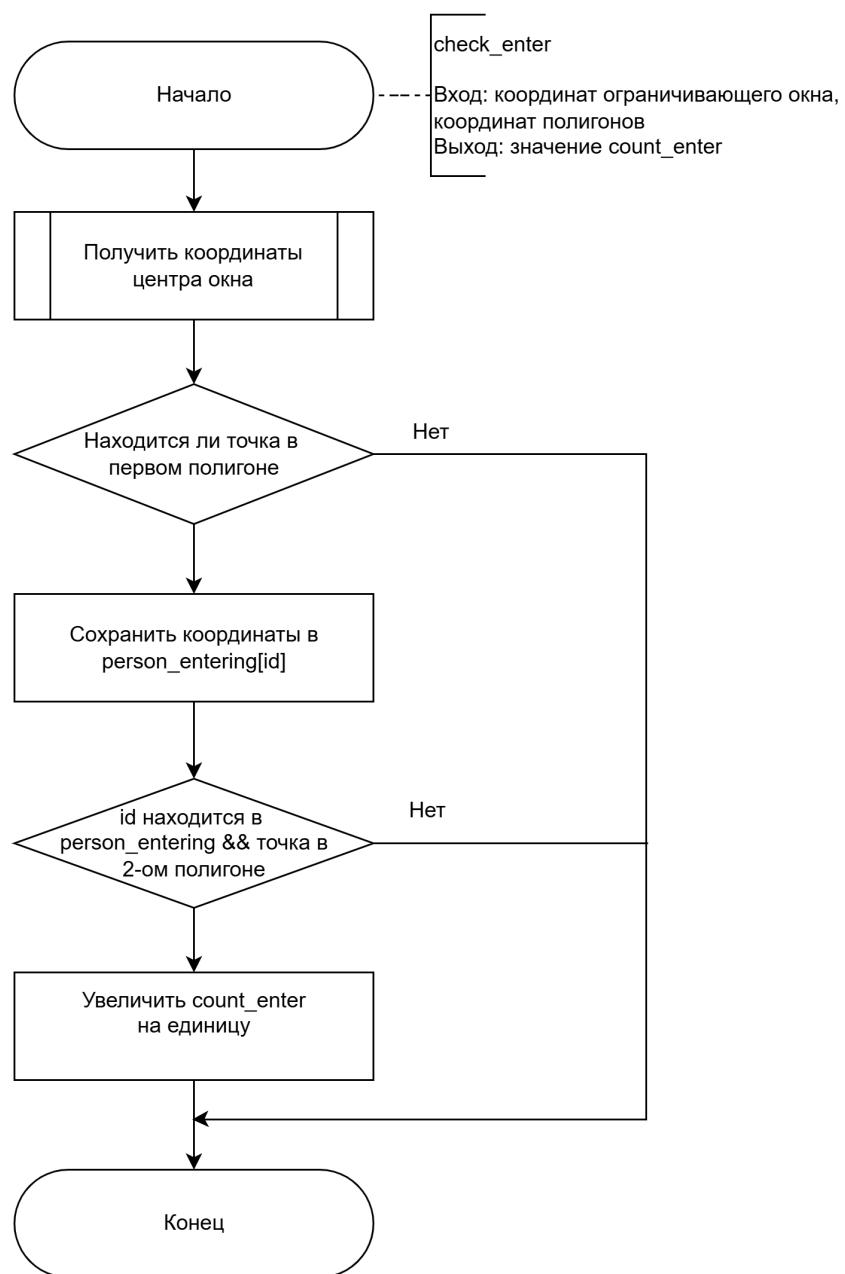


Рисунок 2.4 – Алгоритм подсчета количества входящих человек

Аналогично и наоборот. Если человек сначала замечен во втором полигоне, он считается выходящим. Если затем он появляется в первом полигоне, его движение считается выходом из зоны.

На рисунке 2.5 представлен алгоритм подсчета количества выходящих человек.

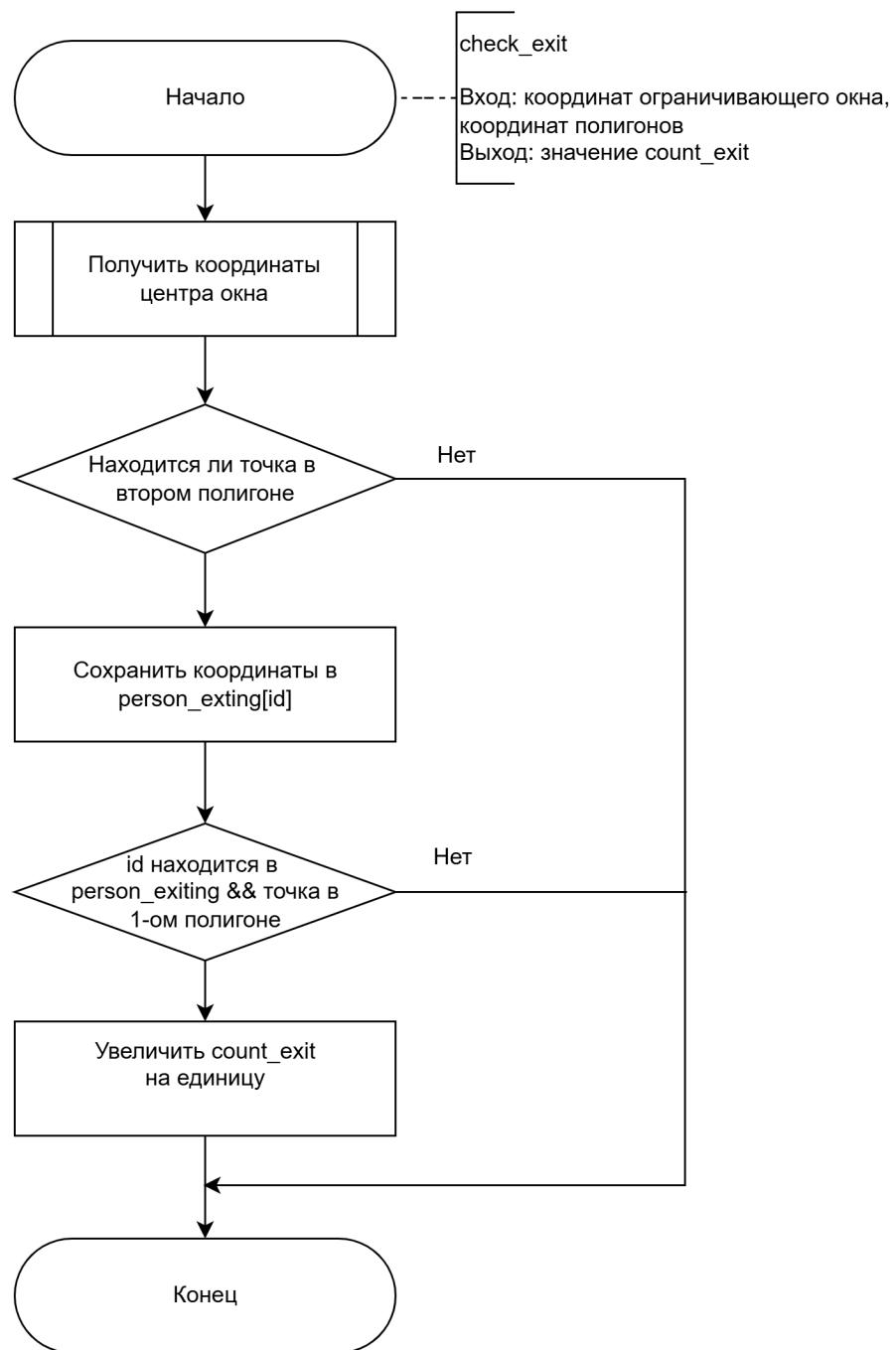


Рисунок 2.5 – Алгоритм подсчета количества выходящих человек

2.2.5 Объединение результатов и отрисовка на кадре

На изображении рисуются два многоугольника — те самые зоны, которые используются для определения входа и выхода людей. Это делается для наглядности: чтобы было понятно, в каких областях кадра производится анализ перемещения.

Затем на экране выводится статистика: сколько входящих человек и сколько выходящих человек. Эти данные обновляются в режиме реального времени и отображаются в виде текста на кадре.

2.3 Функциональная схема обучения модели

Для решения задачи подсчёта количества человек на видео была обучена модель на основе сверточной нейронной сети. В качестве обучающего набора использовались аннотированные изображения с указанием положения людей. Модель обучалась выявлять объекты класса человек и оценивать их количество на каждом кадре. Обучение проводилось на Google Colab.

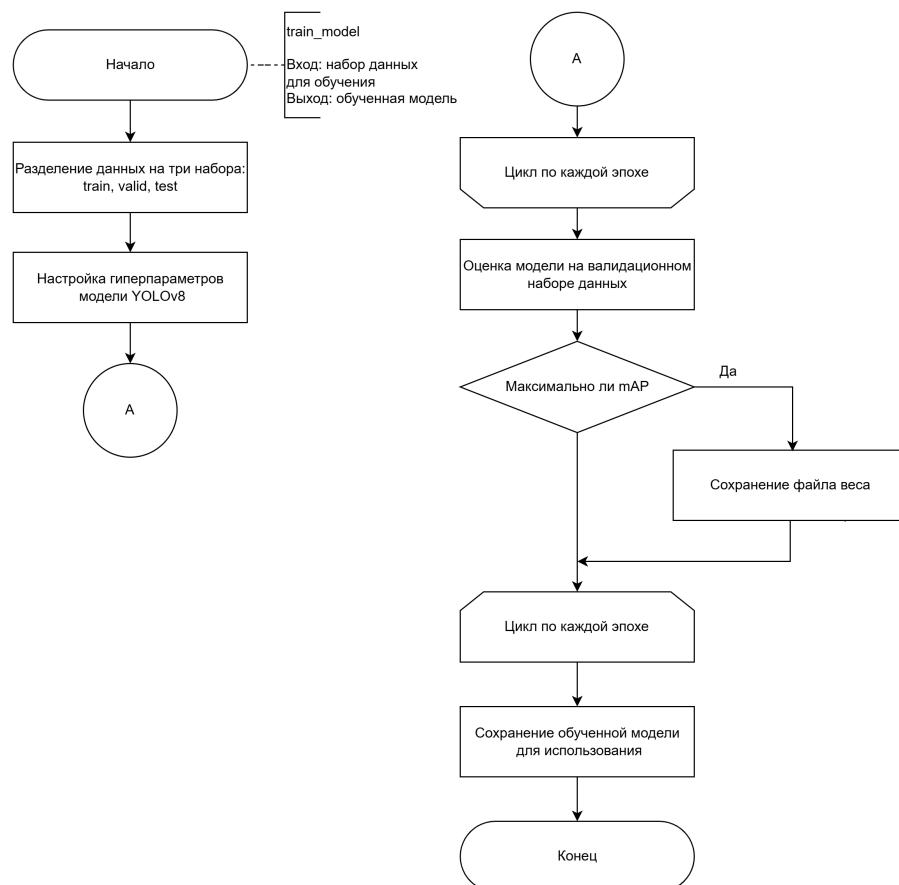


Рисунок 2.6 – Алгоритм обучения модели YOLOv8n

2.4 Структура программного обеспечения

Программное обеспечение состоит из следующих основных модулей, каждый из которых выполняет определённую функцию в рамках общей архитектуры системы:

- модуль взаимодействия с пользователем;
- модуль отслеживания объектов;
- модуль подсчета количества человек;
- модуль сверточной нейронной сети YOLOv8n.

Схема архитектуры программного обеспечения, реализующего метод подсчета количества человек на видео на основе сверточных нейронных сетей, представлена на рисунке 2.7

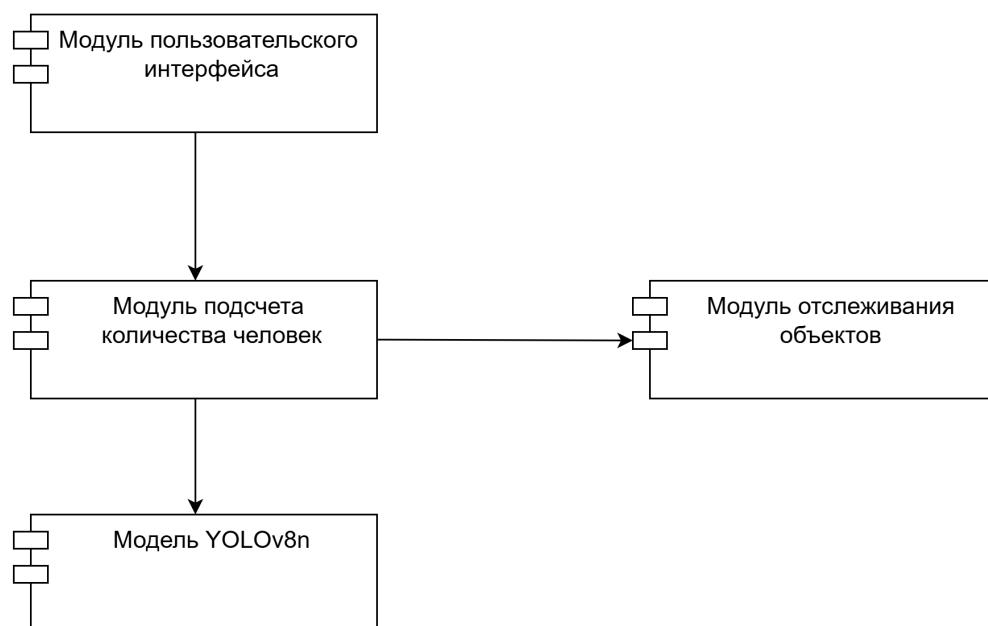


Рисунок 2.7 – Диаграмма компонентов разработанного программного обеспечения

2.5 Вывод

В данном разделе были рассмотрены требования, предъявляемые к разрабатываемому методу подсчёта количества человек на видео на основе сверточных нейронных сетей. Были приведены требования к программному обеспечению, реализующему разработанный метод, включая необходимость

обеспечения удобного пользовательского интерфейса, возможности загрузки видеоданных, задания пользовательских полигонов и отображения результатов обработки.

Также в разделе были выделены и описаны основные этапы, из которых состоит предложенный метод. Эти этапы были логически структурированы и представлены в виде IDEF0-диаграмм первого уровня. Была представлена общая структура программного обеспечения, включающая в себя основные модули и их взаимодействие.

3 Технологический раздел

3.1 Выбор средств программной реализации

В качестве среды разработки был использован Visual Studio Code [22].
Данный выбор обусловлен следующими причинами:

- VS Code предоставляет удобные инструменты для программирования на Python [23].
- VS Code имеет встроенную поддержку Git, что позволяет удобно отслеживать изменения и работать в команде [24].
- VS Code обладает хорошей интеграцией с библиотеками , как OpenCV, PyQt5, YOLO.

В качестве языка программирования был выбран язык Python, так как он имеет некоторые преимущества [23]:

- Python имеет огромное количество библиотек для работы с данными, машинным обучением, обработкой изображений и видео.
- Python известен своей простотой и читаемостью кода, что облегчает разработку и поддержку проекта.
- Python является стандартным языком для работы с моделями глубокого обучения.

В качестве библиотеки для работы с видео был выбран OpenCV [25]. Он предоставляет все необходимые инструменты для работы с видеопотоком и изображениями, включая чтение кадров, их обработку и отображение результатов. Библиотека легко сочетается с нейросетевыми моделями, что делает её эффективным решением для задач обнаружения и отслеживания объектов.

Для создания пользовательского интерфейса был выбран PyQt5 [26]. Он позволяет создать интуитивно понятный интерфейс с поддержкой отображения видео и элементов управления, таких как кнопки и поля ввода.

3.2 Описание формата входных и выходных данных

Входными данными, необходимыми для разрабатываемого программного обеспечения, являются параметры, поступающие от пользователя в процессе взаимодействия с интерфейсом системы. К этим параметрам относятся:

- путь к видео, загружаемому пользователем. Поддерживаются популярные форматы мультимедиа, такие как .mp4, .mov, .avi;
- массив координат, определяющих вершины полигонов. Эти координаты задаются в целочисленном формате и используются для обозначения зон интереса в кадре, через которые производится подсчёт входящих и выходящих человек.

Результатом работы программного обеспечения является последовательность видеокадров, на которых визуально отображаются результаты анализа. В частности, на каждом кадре выполняется отрисовка полигонов и нанесение текстовой информации, содержащей статистику по количеству людей, вошедших и вышедших через заданные зоны.

3.3 Сборка программного обеспечения

Сборка программного обеспечения осуществляется в изолированной среде на базе виртуального окружения Python (venv) [27]. Виртуальное окружение позволяет обеспечить стабильность работы и избежать конфликтов между зависимостями различных библиотек.

На этапе подготовки создаётся виртуальное окружение в корневой директории проекта с помощью следующей команды:

```
python -m venv venv
```

В результате формируется директория `venv`, содержащая все необходимые исполняемые файлы и каталоги для изолированной работы Python-приложения.

Перед установкой зависимостей необходимо активировать виртуальное окружение. Команды активации зависят от операционной системы:

- Для Windows:

```
venv\Scripts\activate
```

- Для Linux/macOS:

```
venv/bin/activate
```

После активации в командной строке появится префикс (`venv`), указывающий на активное виртуальное окружение.

Все зависимости проекта перечислены в файле `requirements.txt`. Установка производится с помощью команды:

```
pip install -r requirements.txt
```

3.4 Демонстрация работы программы

В ходе разработки было создано программное обеспечение, реализующее предложенный метод подсчета количества человек на видео.

Интерфейс взаимодействия с пользователем представлен на рисунке 3.1

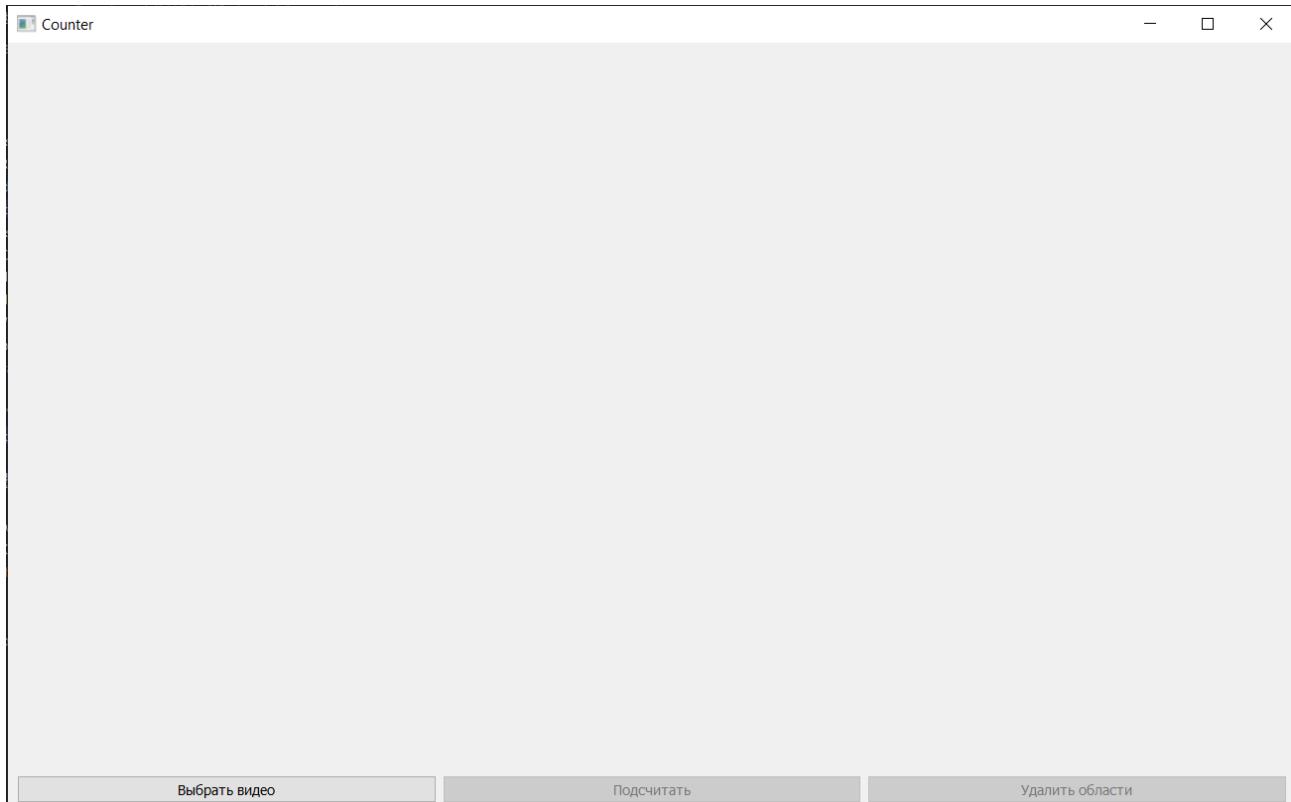


Рисунок 3.1 – Пользовательский интерфейс

Краткое описание способа обращения к разрабатываемому программному обеспечению, реализующему предлагаемый метод:

- Выбрать видео через диалоговое окно кнопкой «Выбрать видео».
- На экране появится демонстрации видео, на котором левой кнопкой мыши задаются два полигона для определения направления движения человека. Так же очищаются заданные полигоны кнопкой «Удалить области».
- После задания полигонов, нажать «Подсчитать» для начала подсчета количества человек.

На рисунке 3.2 представлены примеры работы программы.



Рисунок 3.2 – Демонстрация работы программы

3.5 Тестирование

После разработки программного обеспечения было выполнено тестирование разработанного метода подсчета количества человек на видео.

Тестирование разработанного метода подсчета количества человек на видео проводилось на следующих случаях:

- видео с небольшим количеством людей, сцена с количеством людей 3–5;
- видео с большим числом людей с высокой плотностью;
- видео с людьми, несущими предметы (сумки и рюкзаки), и с присутствием животных (собаки);
- видео с перекрытием объектов, люди частично закрывают друг друга;
- видео с недостаточным освещением, съёмка с низкой освещённостью;
- видео, на котором отсутствуют люди но присутствуют другие объекты.

Для оценки точности подсчёта использовалась простая метрика: если эталонное значение не равно нулю, точность определялась по формуле 3.1:

$$\text{Точность} = 100\% - \left(\frac{|N_{\text{предсказано}} - N_{\text{эталон}}|}{N_{\text{эталон}}} \times 100\% \right) \quad (3.1)$$

В случае, если эталонное значение равно нулю, а система также не зафиксировала присутствия людей, точность считалась равной 100%. Результаты тестирования приведены в таблице 3.1.

Таблица 3.1 – Сравнение эталонных значений/предсказанных значений

Случай	Вход (эталон/предсказанное)	Выход (эталон/предсказанное)
1	4/4	1/1
2	0/0	8/9
3	1/1	15/14
4	0/0	23/20
5	0/0	9/8
6	0/0	0/0

В рамках тестирования системы были проанализированы шесть различных видеосценариев, для каждого из которых заранее были заданы эталонные значения количества входящих и выходящих людей. На основе результатов работы алгоритма проводилось сравнение предсказанных данных с фактическими, с последующим расчётом точности.

По результатам расчетов средняя точность системы при определении входящих людей равна 100.0%, что свидетельствует о полной корректности

алгоритма на рассматриваемом наборе данных. Для выходящих людей средняя точность оказалась равной 92.78%, что также подтверждает эффективность подсчёта, несмотря на отдельные незначительные отклонения от эталонных значений.

3.6 Вывод

В данном разделе были выбраны средства для реализации предлагаемого метода. В качестве языка программирования был выбран Python, в качестве среды разработки был выбран Visual Studio Code.

Также были приведены реализации программного обеспечения, был описан формат входных и выходных данных.

Представлено краткое описание пользовательского интерфейса и проведена демонстрация работы программы и проведено тестирование.

4 Исследовательский раздел

4.1 Технические характеристики

Исследование проводилось с использованием персонального компьютера, обладающего следующими техническими характеристиками:

- устройство функционировало под управлением операционной системы Windows 10 Home Single Language;
- объём установленной оперативной памяти составлял 8 гигабайт, что позволяло эффективно обрабатывать массивы данных и выполнять ресурсоемкие вычисления;
- в качестве центрального процессора использовался 4-ядерный Intel(R) Core(TM) i7-1165G7 одиннадцатого поколения с тактовой частотой 2.80 ГГц.

На протяжении всего периода проведения испытаний устройство находилось в стационарном режиме и было подключено к внешнему источнику электропитания, что исключало влияние изменений в уровне заряда аккумулятора на стабильность вычислений. Также одновременно с основной системой замера на устройстве были запущены фоновые приложения окружения (включая редакторы кода, инструменты визуализации и средства мониторинга ресурсов), что моделировало реальные условия эксплуатации программного обеспечения в рабочей среде и позволяло получить более достоверные результаты производительности и устойчивости функционирования.

4.2 Зависимость точности метода от уровня освещённости

Целью данного исследования является изучение влияния освещённости видео на точность работы разработанного метода.

В качестве входных данных использовалось видео в формате MOV с фиксированным разрешением и частотой кадров. Для имитации различных условий освещённости применялась линейная коррекция яркости каждого

кадра с использованием функции `cv2.convertScaleAbs` из библиотеки *OpenCV*:

$$I_{\text{новый}} = \alpha \cdot I_{\text{исходный}} + \beta, \quad (4.1)$$

где: I — значение яркости пикселя, α — коэффициент контрастности, β — смещение по яркости.

Значение параметра β фиксировалось равным нулю, чтобы не вносить дополнительных смещений по яркости. Параметр α варьировался в диапазоне от 0.1 до 1.0 с шагом 0.1. Точность метода определяется по формуле 3.1.

На графике 4.1 представлена зависимость точности метода от уровня освещенности.

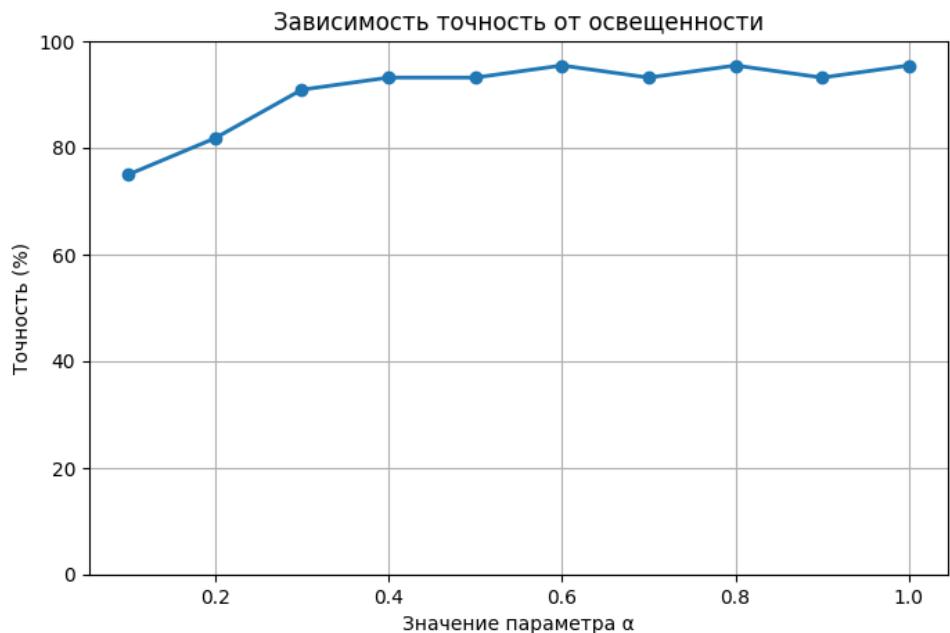


Рисунок 4.1 – Зависимости точности метода от уровня освещенности

Исходя из построенного графика, можно сделать вывод, что точность метода изменяется в зависимости от уровня освещённости, задаваемого параметром α . При низких значениях α (0.1—0.3), соответствующих затемненному видео, точность существенно снижается — до 75–90%. При дальнейшем увеличении яркости ($\alpha = 0.4$ и выше), наблюдается стабилизация точности на уровне 93–95%.

4.3 Зависимость точности метода от качества видео

Целью данного исследования является изучение влияния качества видео на точность разработанного метода.

В качестве входных данных использовалось видео с фиксированным разрешением. Для имитации ухудшения качества применялось Гауссово размытие, осуществляемое с помощью функции `cv2.GaussianBlur()` из библиотеки OpenCV:

$$\text{Кадр}_{\text{размыт}} = \text{GaussianBlur}(\text{Кадр}_{\text{исходный}}, (k, k), \sigma), \quad (4.2)$$

где: k — размер ядра, σ — стандартное отклонение (можно опустить, если по умолчанию).

Параметр k варьировался в диапазоне от 1 до 17 с шагом 2. Точность метода определяется по формуле 3.1.

На графике 4.2 представлена зависимость точности метода от уровня освещенности.

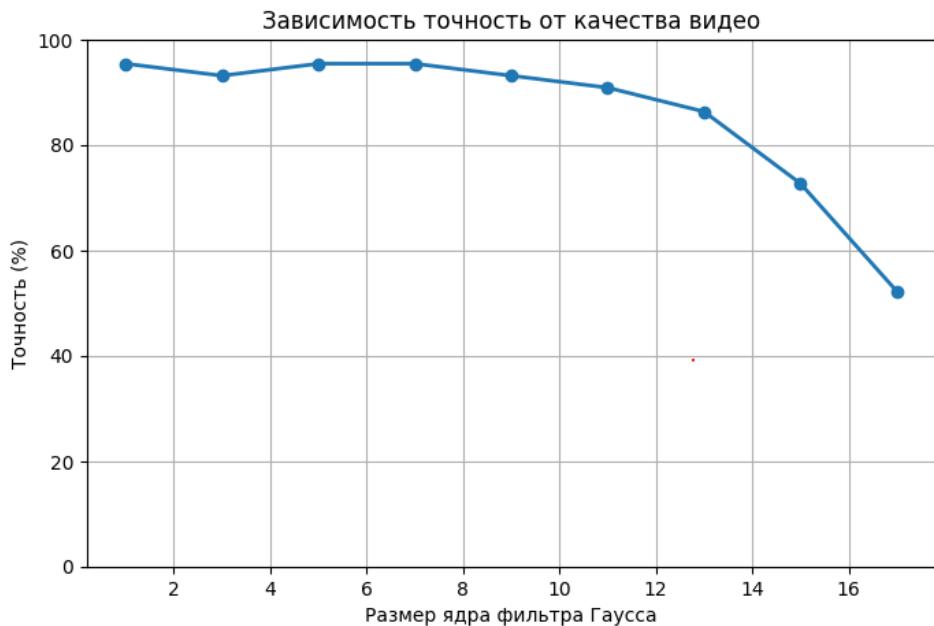


Рисунок 4.2 – Зависимость точности метода от качества видео

Результаты показывают, что увеличение параметра k , определяющего размер ядра фильтра Гаусса, оказывает влияние на точность работы метода.

При небольших значениях k (от 1 до 7) метод демонстрирует стабильную и высокую точность — 93–95%. Однако, начиная с $k = 9$, наблюдается

постепенное снижение точности: до 90.91% при $k = 11$, 86.36% при $k = 13$, и резкое падение до 72.73% и 52.27% при $k = 15$ и $k = 17$ соответственно.

4.4 Зависимость времени работы метода от качества видео

Целью данного исследования является изучение влияния качества видео на время работы разработанного метода.

В данном исследовании использовались те же входные данные, что и в предыдущей секции. Параметр размера ядра фильтра Гаусса k в диапазоне от 1 до 17 с шагом 2.

На графике 4.3 представлена зависимость времени работы метода от качества видео.

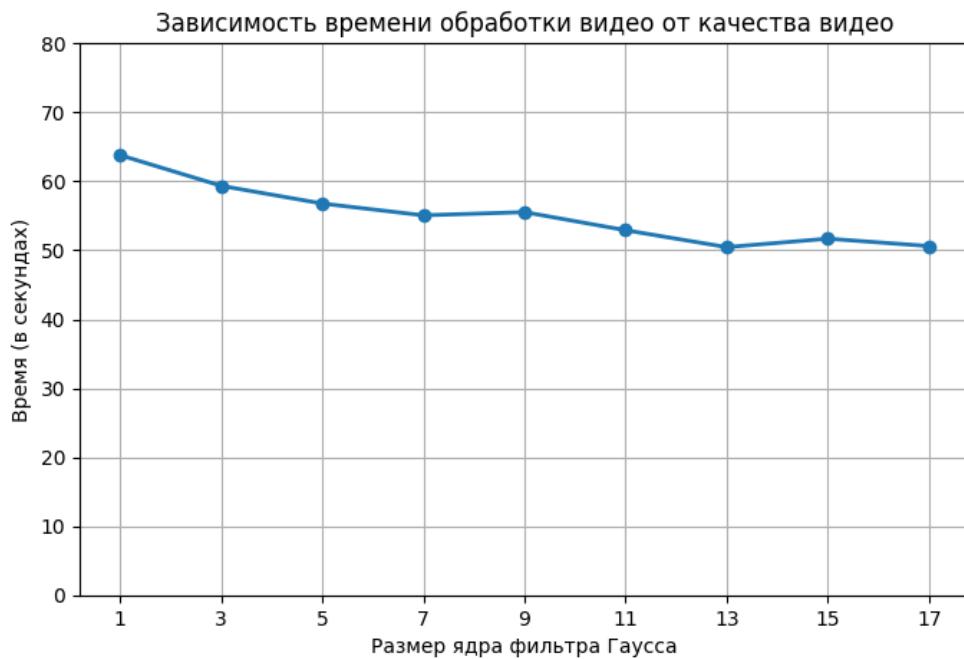


Рисунок 4.3 – Зависимость времени работы метода от качества видео

Проведенное исследование показало, что снижение качества видео, реализованное путем увеличения степени размытия (увеличения k), приводит к сокращению общего времени работы метода, поскольку размытые кадры содержат меньше деталей, и алгоритм обрабатывает их быстрее.

4.5 Вывод

Проведенные исследования показали, что точность разработанного метода зависит как от уровня освещенности, так и от качества входного видео.

При снижении яркости (низкие значения α) точность уменьшается, тогда как при умеренном освещении она стабильно высокая. Аналогично, увеличение степени размытия (рост параметра k) приводит к снижению точности, особенно при сильном ухудшении качества. В то же время, снижение качества видео положительно сказывается на времени обработки: более размытые кадры обрабатываются быстрее.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы был разработан метод подсчета количества человек на видео на основе сверточных нейронных сетей.

Для достижения поставленной цели в рамках данной работы были решены все предусмотренные задачи:

- рассмотрены существующие методы поиска объектов на изображениях;
- спроектирован и реализован метод подсчета количества человек на видео на основе сверточных нейронных сетей;
- разработано программное обеспечение, реализующее предлагаемый метод;
- исследованы характеристики реализованного метода подсчета количества человек на видео на основе сверточных нейронных сетей.

В результате исследования было установлено, что точность метода снижается при слабом освещении, точность метода увеличивается при увеличении качества видео. Так же время работы разработанного метода увеличивается при уменьшении качества видео.

Для дальнейшего развития реализованного метода и повышения его прикладной ценности можно предложить следующие направления:

- реализовать возможность создания еженедельного/ежемесячного отчета;
- реализовать расширение метода, т.е подсчитать не только людей а другие объекты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Манюкова Н. В.* Компьютерное зрение как средство извлечения информации из видеоряда // Математические структуры и моделирование. — 2015. — 4 (36). — С. 123—128.
2. *Конушин А. С., Филиппов И. В., Кононов В. А.* Подсчет количества людей в видеопоследовательности на основе детектора головы человека // Программные продукты и системы. — 2015. — 1 (109). — С. 121—126.
3. Computer Vision Tasks supported by Ultralytics. — [Электронный ресурс]. — Режим доступа: <https://docs.ultralytics.com/tasks/> (дата обращения: 14.04.2025).
4. *Татьянкин В. М.* Алгоритм формирования оптимальной архитектуры многослойной нейронной сети // Новое слово в науке: перспективы развития. — 2014. — № 2. — С. 187—188.
5. Как устроена нейронная сеть и зачем ей слои. — [Электронный ресурс]. — Режим доступа: <https://www.arcsinus.ru/blog/neuronet-architecture> (дата обращения: 17.04.2025).
6. *Sharma Sagar, Sharma Simone, Athaiya Anidhya.* Activation functions in neural networks // Towards Data Sci. — 2017. — Т. 6, № 12. — С. 310—316.
7. Гайд по работе сверточных нейронных сетей. — [Электронный ресурс]. — Режим доступа: <https://blog.skillfactory.ru/gayd-po-rabote-svertochnyh-neyronnyh-setey/> (дата обращения: 17.04.2025).
8. *Верхов К. А.* Обнаружение объектов на изображении с использованием машинного обучения // Новые информационные технологии в научных исследованиях. — 2020. — С. 226—227.
9. One-Stage Object Detectors. — [Электронный ресурс]. — Режим доступа: <https://www.ultralytics.com/glossary/one-stage-object-detectors> (дата обращения: 23.04.2025).
10. Two-Stage Object Detectors. — [Электронный ресурс]. — Режим доступа: <https://www.ultralytics.com/glossary/two-stage-object-detectors> (дата обращения: 26.04.2025).

11. YOLO Algorithm: Real-Time Object Detection from A to Z. — [Электронный ресурс]. — Режим доступа: <https://kili-technology.com/data-labeling/machine-learning/yolo-algorithm-real-time-object-detection-from-a-to-z> (дата обращения: 13.04.2025).
12. YOLOv1. — [Электронный ресурс]. — Режим доступа: <https://paperswithcode.com/method/yolov1> (дата обращения: 13.04.2025).
13. What is YOLOv5? A Guide for Beginners. — [Электронный ресурс]. — Режим доступа: <https://blog.roboflow.com/yolov5-improvements-and-evaluation/> (дата обращения: 14.04.2025).
14. YOLOv8 Architecture Explained: Exploring the YOLOv8 Architecture. — [Электронный ресурс]. — Режим доступа: <https://yolov8.org/yolov8-architecture-explained/> (дата обращения: 16.04.2025).
15. *Shuai Qianjun, Wu Xingwen.* Object detection system based on SSD algorithm. — 2020.
16. *Zhai Shang.* DF-SSD: An improved SSD object detection algorithm based on DenseNet and feature fusion // IEEE access. — 2020. — Т. 8. — С. 24344—24357.
17. What is R-CNN? — [Электронный ресурс]. — Режим доступа: <https://blog.roboflow.com/what-is-r-cnn/> (дата обращения: 18.04.2025).
18. *Girshick, Ross.* Fast r-cnn // Proceedings of the IEEE international conference on computer vision. — 2015. — С. 1440—1448.
19. *Bharati, Ankita.* Deep learning techniques—R-CNN to mask R-CNN: a survey // Computational Intelligence in Pattern Recognition: Proceedings of CIPR 2019. — 2020. — С. 657—668.
20. YOLOv8 vs YOLOv5 A Detailed Comparison. — [Электронный ресурс]. — Режим доступа: <https://docs.ultralytics.com/compare/yolov8-vs-yolov5/> (дата обращения: 29.04.2025).
21. MS COCO (Microsoft Common Objects in Context). — [Электронный ресурс]. — Режим доступа: <https://paperswithcode.com/dataset/coco> (дата обращения: 17.04.2025).
22. Visual Studio Code - Code Editing. — [Электронный ресурс]. — Режим доступа: <https://code.visualstudio.com/> (дата обращения: 11.04.2025).

23. The Python Tutorial. — [Электронный ресурс]. — Режим доступа: <https://www.python.org/> (дата обращения: 11.04.2025).
24. Git. — [Электронный ресурс]. — Режим доступа: <https://git-scm.com/> (дата обращения: 25.04.2025).
25. OpenCV - Open Computer Vision Library. — [Электронный ресурс]. — Режим доступа: <https://opencv.org/> (дата обращения: 22.04.2025).
26. PyQt5 Tutorial 2025, Create Python GUIs with Qt. — [Электронный ресурс]. — Режим доступа: <https://www.pythonguis.com/pyqt5-tutorial/> (дата обращения: 24.04.2025).
27. Как создать виртуальное окружение Python. — [Электронный ресурс]. — Режим доступа URL: <https://sky.pro/media/kak-sozdat-virtualnoe-okruzhenie-python> (дата обращения: 21.04.2025).

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация метода подсчета людей на видео (часть 1)

```
class MyObjectCounter:  
    def __init__(self, model_path="best.pt", polygon1=None,  
                 polygon2=None, classes=[0,1], show=True):  
        self.model = YOLO(model_path)  
        self.tracker = Tracker()  
        self.polygon1 = polygon1 if polygon1 else []  
        self.polygon2 = polygon2 if polygon2 else []  
        self.classes = classes  
        self.show = show  
        self.person_entering = {}  
        self.person_exiting = {}  
        self.entering = set()  
        self.exiting = set()  
    def is_inside_polygon(self, point, polygon):  
        if polygon:  
            return cv2.pointPolygonTest(np.array(polygon,  
                                              dtype=np.int32), point, False) >= 0  
  
    def process_frame(self, frame, test=False):  
        results = self.model(frame, classes=self.classes,  
                             verbose=False)[0]  
        detections = []  
  
        for r in results.boxes.data.tolist():  
            x1, y1, x2, y2, conf, cls = r  
            detections.append([x1, y1, x2, y2])  
  
        bbox_id = self.tracker.update(detections)  
  
        for bbox in bbox_id:  
            x3, y3, x4, y4, id = bbox  
            x3, y3, x4, y4 = int(x3), int(y3), int(x4), int(y4)  
            cx = int((x3 + x4) / 2)  
            cy = int((y3 + y4) / 2)  
            x0, y0 = cx, cy  
            cv2.rectangle(frame, (x3, y3), (x4, y4), (125, 125,  
                125), 2)
```

Листинг А.2 – Реализация метода подсчета людей на видео (часть 2)

```
cv2.putText(frame, f'{id}', (x3, y3),
           cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255,
           0), 1)

if self.is_inside_polygon((x0, y0), self.polygon1):
    self.person_entering[id] = (x0, y0)
    cv2.rectangle(frame, (x3, y3), (x4, y4), (0,
        255, 255), 2)
    cv2.circle(frame, (x0, y0), 5, (255, 0, 255),
               -1)
if id in self.person_entering:
    if self.is_inside_polygon((x0, y0),
        self.polygon2):
        cv2.rectangle(frame, (x3, y3), (x4, y4), (0,
            0, 0), 2)
        cv2.circle(frame, (x0, y0), 5, (0, 0, 0),
                   -1)
        self.entering.add(id)

if self.is_inside_polygon((x0, y0), self.polygon2):
    self.person_exiting[id] = (x0, y0)
    cv2.rectangle(frame, (x3, y3), (x4, y4), (0, 0,
        255), 2)
    cv2.circle(frame, (x0, y0), 5, (255, 0, 255),
               -1)

if id in self.person_exiting:
    if self.is_inside_polygon((x0, y0),
        self.polygon1):
        cv2.rectangle(frame, (x3, y3), (x4, y4),
                      (255, 255, 0), 2)
        cv2.circle(frame, (x0, y0), 5, (123, 123,
            123), -1)
        self.exiting.add(id)

if self.polygon1:
    cv2.polylines(frame, [np.array(self.polygon1,
        np.int32)], True, (0, 255, 255), 1)
```

Листинг А.3 – Реализация метода подсчета людей на видео (часть 3)

```
if self.polygon2:  
    cv2.polylines(frame, [np.array(self.polygon2,  
        np.int32)], True, (0, 255, 255), 1)  
    cv2.putText(frame, f'People entering:  
    {len(self.entering)}', (30, 50),  
               cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 0, 255),  
               3)  
    cv2.putText(frame, f'People exiting:  
    {len(self.exiting)}', (30, 100),  
               cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 0, 255),  
               3)  
if self.show:  
    cv2.imshow("MyObjectCounter", frame)  
if test:  
    return len(self.entering)  
else:  
    return frame
```

Листинг А.4 – Реализация метода отслеживания объектов (часть 1)

```
class Tracker:  
    def __init__(self):  
        self.tracked_objects = {}  
        self.id_count = 0  
  
    def compute_iou(box1, box2):  
        x1, y1, w1, h1 = box1  
        x2, y2, w2, h2 = box2  
        x1_max, y1_max = x1 + w1, y1 + h1  
        x2_max, y2_max = x2 + w2, y2 + h2  
  
        inter_x1 = max(x1, x2)  
        inter_y1 = max(y1, y2)  
        inter_x2 = min(x1_max, x2_max)  
        inter_y2 = min(y1_max, y2_max)  
        inter_area = max(0, inter_x2 - inter_x1) * max(0,  
            inter_y2 - inter_y1)  
        area1 = w1 * h1  
        area2 = w2 * h2  
        union_area = area1 + area2 - inter_area  
        return inter_area / union_area if union_area > 0 else 0
```

Листинг А.5 – Реализация метода отслеживания объектов (часть 2)

```
def update(self, objects_rect):
    objects_bbs_ids = []
    updated_ids = set()
    for rect in objects_rect:
        x, y, w, h = rect
        cx = (x + x + w) // 2
        cy = (y + y + h) // 2
        best_match_id = None
        best_score = float("inf")

        for obj_id, old_rect in self.tracked_objects.items():
            ox, oy, ow, oh = old_rect
            ocx = (ox + ox + ow) // 2
            ocy = (oy + oy + oh) // 2

            dist = math.hypot(cx - ocx, cy - ocy)
            iou = compute_iou(rect, old_rect)

            if dist < 100 and iou > 0.4:
                if dist < best_score:
                    best_score = dist
                    best_match_id = obj_id

        if best_match_id is not None:
            self.tracked_objects[best_match_id] = rect
            objects_bbs_ids.append([x, y, w, h,
                                   best_match_id])
            updated_ids.add(best_match_id)
        else:
            self.tracked_objects[self.id_count] = rect
            objects_bbs_ids.append([x, y, w, h,
                                   self.id_count])
            updated_ids.add(self.id_count)
            self.id_count += 1
    self.tracked_objects = {i: bb for i, bb in
                           self.tracked_objects.items() if i in updated_ids}
    return objects_bbs_ids
```

ПРИЛОЖЕНИЕ Б

Презентация к выпускной квалификационной работе состоит из 14 слайдов.