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

ФАКУЛЬТЕТ «Фундаментальные науки»

КАФЕДРА «Математическое моделирование»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Разработка нейронных сетей для обнаружения автомобилей на изображениях

Студент ФН12-41м (Группа)	(Подпись, дата)	А. А. Комлев (И.О.Фамилия)
Руководитель ВКР	(Подпись, дата)	Д. А. Фетисов (И.О.Фамилия)
Консультант	(Подпись, дата)	(И.О.Фамилия)
Консультант	(Подпись, дата)	(И.О.Фамилия)
Нормоконтролер	(Подпись, дата)	М. А. Велищанский (И.О.Фамилия)

20 23 г.

РЕФЕРАТ

Расчетно-пояснительная записка 49 с., 16 рис., 17 источников.

НЕЙРОННЫЕ СЕТИ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, ОБНАРУЖЕНИЕ ОБЪЕКТОВ.

Предметом исследования является сравнение методов обнаружения объектов на изображениях, а также программная реализация используемых алгоритмов.

Цель работы — разработка и написание программного кода, реализующего обнаружения автомобилей на изображениях.

В качестве исходного набора нейросетевых архитектур используются сети YOLO и ViT. Проводится исследование точности моделей, и сравнение скорости работы.

По результатам, полученным в ходе программной реализации, проведен сравнительный анализ используемых методов.

Содержание

ВВЕДЕНИЕ

Задача обнаружения объектов, связанная с выделением и классификацией объектов на изображении, является важной областью компьютерного зрения.

Одним из ранних методов обнаружения является метод Виолы-Джонса [1, 2], основанный на каскадных классификаторах, который был популярен в прошлом, но имеет некоторые недостатки, такие как невысокая точность обнаружения на изображениях в условиях изменения масштаба и поворота объектов.

Современные подходы для обнаружения объектов на изображениях основаны на использовании нейронных сетей.

R-CNN [3] – это первый метод обнаружения объектов на изображении, основанный на нейронных сетях. Он использует сверточные нейронные сети для извлечения признаков из областей изображения, а затем классифицирует и локализует объекты. Однако сеть R-CNN работает медленно из-за необходимости обрабатывать каждую область изображения отдельно.

Fast R-CNN [4] – улучшенная версия R-CNN, которая использует более эффективный способ обработки областей изображения с помощью RoI-pooling. Он также улучшает скорость и точность обнаружения объектов на изображении.

Faster R-CNN [5] – это метод, основанный на более эффективном процессе выделения областей изображения и RoI-pooling, называемом Region Proposal Network (RPN). Он может автоматически генерировать регионы интереса, что увеличивает скорость обнаружения.

С другой стороны, YOLO (You Only Look Once) [6] – это метод, который выделяет объекты и классифицирует их на изображении в один прогон. Он использует сеть, которая делит изображение на сетку ячеек, каждая из которых ответственна за предсказание объектов в этой ячейке.

SSD (Single Shot MultiBox Detector) [7] – похож на YOLO, это ещё один метод одноступенчатого обнаружения объектов, который предсказывает класс и ограничивающие рамки в одну попытку. Однако он использует другой подход к разделению изображения на набор якорных коробок с заранее определен-

ными соотношениями сторон, масштабами и позициями.

Трансформеры [8] – это архитектура нейронных сетей, которая изначально была разработана для обработки текстов, но также может быть использована для обработки изображений с помощью «механизма внимания». Использование трансформеров позволяет задействовать глобальный контекст изображения и агрегировать информацию из различных групп областей изображения. Примерами нейронных сетей, основанных на трансформерах, которые были успешно применены в задаче обнаружения объектов на изображении, являются DeTr и ViT.

DeTr [9] – это одна из первых нейронных сетей, полностью основанных на трансформерах. Она показала превосходство над другими методами обнаружения объектов на изображении и демонстрирует высокую точность обнаружения.

ViT [10]– это нейронная сеть, которая была разработана для обработки изображений в виде сетки пикселей, и основана на трансформере. Она также показывает хорошую точность при решении задачи обнажения объектов на изображениях.

1 ЗАДАЧА ОБНАРУЖЕНИЯ ОБЪЕКТОВ

Популярная в области компьютерного зрения задача обнаружения заключается в нахождении на изображениях и видеозаписях различных объектов, а также локализации их местоположения. Преимущественно данная область ограничивается прямоугольником, допускающим определенную параметризацию – изменение угла, ширины,высоты и т.д. В соответсвии с формой объекта вместо прямоугольника может быть выбрана другая фигура, например, эллипс или окружность.

Формально задачу можно представить следующим образом:

1) пусть задан набор s исследуемых классов, которым принадлежат объекты;

2) на вход подаются цветные (черно-белые) изображения $I \in R^{N \times M \times C}$;

3) требуется получить выходной набор обнаруженных объектов

$$Obj = \{Obj_i | i = 1, ..., p\},$$

где Obj_i – структура, которая описывает местоположение объекта (bounding box), а также класс $c_i \in [1, s]$, которому он принадлежит.

Достоверность обнаружения зависит от нескольких факторов. Так, чтобы улучшить точность локализации, необходимо добавить угол поворота объекта. Описание границы объекта или области может способствовать более точному описанию его положения.

Искомые объеты могут быть условно разделены на две группы. К первой категории относятся объекты, обнаружение которых может быть ограничено прямоугольником – люди, животные, материальные предметы и т.д. Вторую группу образовывают «неисчисляемые» материалы, которые не имеют конкретной формы, размера, а их описание задано какой-либо текстурой – вода, небо, земля и т.д.

Ограничивающая рамка может быть задана с использованием четырех пространственных координат в двух видах:

1) классический формат – задание координат (x_{min}, y_{max}) левого верхнего и координат (x_{max}, y_{min}) правого нижнего углов;

2) центрированное задание – определение координат центра (c_x, c_y) прямоугольника, а также высоты h и ширины w.

В настоящее время задача обнаружения объектов является довольно частой в сфере автотранспортного движения в связи с развивающимися технологиями беспилотного управления и контроля трафика. Большое число существующих методов требует изучения их применения и усовершенствования для реализации задач обнаружения в режиме реального времени.

2 ПОДХОДЫ К ЗАДАЧЕ ОБНАРУЖЕНИЯ

Наиболее популярным направлением в области распознавания и обнаружения объектов являются искусственные нейронные сети. В тех случаях, когда заранее известны возможные типы объектов, хорошее качество решения задачи показывают алгоритмы, использующие сверточные нейронные сети (CNN). Использование подобного семейства сетей позволяет обнаружать объекты разных классов и пространственных размеров.

Можно выделить два основных подхода применения CNN:

1) ряд задач может быть реализован посредством применения готовых или уже предобученных архитектур; они могут находить объекты, на которых производилось обучение;

2) для других более частных задач может быть применена новая база изображений с целью дообучения модели, для которой имеется предобученная архитектура, в соответствии с поставленной задачей; данных случай предполгает настройку весов, в основном, исключительно для последних полносвязных слоев.

Изучение и развитие детекторов (алгоритмов, позволяющих выделять объекты), а также хорошие показатели качества их работы привело к возникновению большого разнообразия нейросетевых архитектур.

2.1 Метод скользящего окна

Первым, однако наиболее затратным с точки зрения вычислений, считается метод скользящего окна. При таком подходе происходит пошаговый проход по всему изображению, во время которого происходит извлечение определенного региона, а также классифицирование объекта, попавшего в него.

Можно предположить, что простым вариантом послужит построение модели обнаружения объектов на основе модели, реализующей классификацию. Имея классификатор, который показывает хорошие результаты, можно обнаружать объекты, сдвигая «окно» по изображению и определяя, относится ли часть, ограниченная окном, к желаемому классу. Однако такой подход ограничивается двумя проблемами:

1) какой размер окна считать оптимальным, чтобы объект точно содержался в нем; даже однотипные объекты (к примеру, теннисный мяч, футбольный мяч, относящиеся к классу мяч) могут иметь сильно отличащиеся пространственные размеры;

2) соотношение сторон ограничивающего прямоугольника – сильно отличающиеся по форме объекты могут иметь разные пропорции сторон.

Решение описанных проблем требовало бы перебора различных скользящих окон, отличающихся размерами и формой. Это приводит к значительным вычислительным затратам.

2.2 Метод предложения регионов

Другим подходом в задаче обнаружения является метод «предложений регионов» (region proposals), сочетающий в себе два алгоритма. Первый из них создает множество предположений – это регионы, в которых может находиться объект. Второй алгоритм производит классифицирование объекта.

Данный метод предполагает сначала нахождение некоторых областей на изображении, а заем объединение их в группу, согласно определенной иерархии. Такие области впоследствии объединяются согласно различным цветовым пространствам, а также факторам схожести. В результате имеется несколько предложений регионов, в которых может быть обнаружен объект за счет слияния нескольких небольших регионов.

Главным направлением развития глубоких нейронных сетей стало обнаружение объектов в реальном времени. Здесь можно отметить сеть R-CNN (Region Based). Данный подход характеризуется разпознаванием не по всему входному изображению, а в локализованной области. Основные шаги в реализации обнаружения с использованием R-CNN моделей [11]:

1) генерация областей интереса для входного изображения – предложение регионов, которые предположительно могут содержать объект;

2) формирования карты признаков – происходит масштабирование сгенерированных на предыдущем этапе областей до размеров, соответствующих архитектуре нейронной сети CNN, эти данные поступают на вход CNN;

3) классификация – каждый объект определяется к своему классу посредством применения полученного вектора признаков с использованием метода опорных векторов.

Прежде чем осуществить классификацию для каждого предложенного региона используется алгоритм подавления немаксимумов. Его суть заключается в том, что исключительно локальные максимумы могут быть рассмотрены в качестве контура объекта. Применение такого алгоритма обусловлено необходимостью избавления от дублирующих областей на входном изображении.

Оценка качества произведенной классификации производится с помощью часто применяемой в задачах компьютерного зрения метрики Intersection over Union (IoU). Данное значение равняется отношению площади пересечения прямоугольника (области интереса), который был получен в ходе обнаружения, и прямоугольника из разметки к площади их объединения:

$$IoU = \frac{B_1 \cup B_2}{B_1 \cup B_2}. \tag{2.1}$$

Среди минусов использования R-CNN сетей можно отметить их относительно медленную работу, а также то, что в ходе алгоритма рассматриваются лишь отдельные области, а не изображение в целом.

2.3 Обнаружение за один проход

Еще одну группу методов обнаружения составляют такие, в которых локализация и классификация осуществляются одновременно, то есть алгоритм проходит один раз. Данный подход получил название «Single Shot Detection» (SSD, «обнаружение за один проход»).

SSD-алгоритм представляет собой комбинацию двух компонентов – базовой модели и «головы». Первая модель, базовая, является сетью для классификации (средством для извлечения признаков на изображении), которая была предварительно обучена. Преимущественно ею является сеть наподобие ResNet, обученная на ImageNet. При этом из данной сети удален последний полносвязный слой. В результате имеется глубокая нейронная сеть, умеющая находить семантическое значение для изображения, поступающего на вход,

для которого сохранена пространственная структура, несмотря на более низкое разрешение.

Головная часть SSD задается одним или несколькими слоями, добавленными к базовой модели. Данные выхода могут быть интерпретированы как ограничивающие прямоугольники и классы объектов в пространственном расположении активаций конечных слоев. Архитектура сверточной нейронной сети с SSD-детектером схематично представлена на рис. 1.

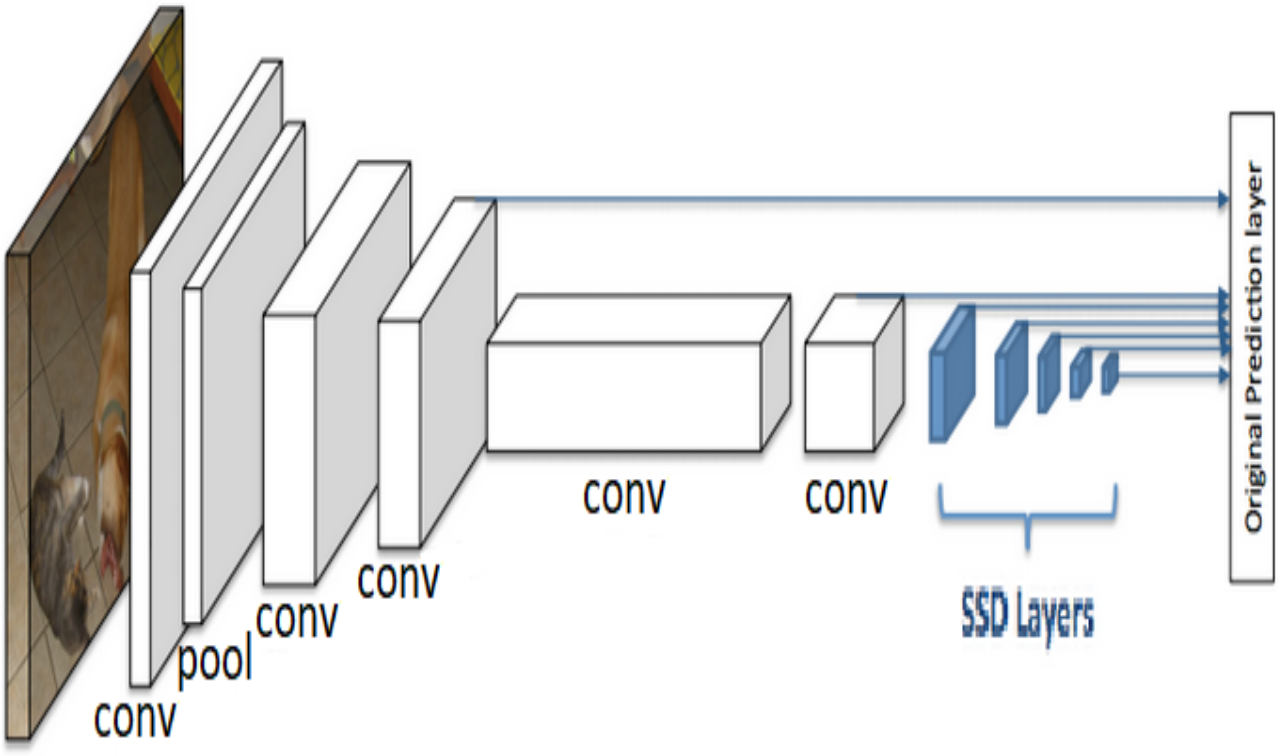


Рис. 1. Сеть с SSD-детектером

Белые прямоугольники на рис. 1 соответствуют базовой модели, а последние слои, представленные синими прямоугольниками, отвечают голове SSD.

В отличие от метода скользящего окна, SSD-алгоритм использует разделение изображения на сетку, каждая ячека которой производит нахождение объектов в свооей области. В случае отсутствия объекта он рассматривается в качестве фонового класса, а его местоположение игнорируется. Расположение и форма объекта выводится самостоятельно каждой ячейкой.

Для случая, когда в одной ячейке могут содержаться сразу несколько объектов, вводятся такие понятия, как якорная рамка (фиксированная рамка) и рецептивное поле.

1. **Якорная рамка (anchor box).** Каждой клетке сетки может соответствовать несколько якорных/приоритетных рамок. Метод SSD опирается на фазу сопоставления в процессе обучения. По сути происходит предсказание класса искомого объекта и его местоположения. На практике каждый якорный блок определяется соотношением сторон и уровнем масштабирования.

Описываемая архитектура позволяет учесть, что форма предметов необязательно может быть квадратной, за счет предварительно заданных соотношений сторон для anchor box. Размер якорной рамки необязательно должен совпадать с размером ячейки сетки. С этой целью вводится параметр zooms – значение, определяющиее, насколько должны увеличиваться или уменьшаться рамки по отношению к каждой ячейке сетки.

2. **Рецептивное поле.** Рецептивным полем называется такая область входного пространства, на которую воздействует конкретная функция CNN. Оно является главной предпосылкой SSD-архитектур, поскольку позволяет обнаруживать объекты разных масштабов, а также определять более узкую рамку.

Говоря о методах, позволяющих обнаружать объекты в один прогон, следует также упомянуть алгоритм YOLO (You Only Look Once).

3 НЕЙРОСЕТЕВЫЕ АРХИТЕКТУРЫ

3.1 Архитектура Transformer

Большое разнообразие нейросетевых архитектур для решения задач компьютерного зрения позволяет находить наилучшие комбинации, которые улучшают точность алгоритмов и скорость их работы. Одним из таких инструментов стала архитектура Transformer-моделей.

Изначально данный агоритм использовался для решения задач обработки естественного языка. Однако модель трансформера оказалась довольно универсальной и масштабируемой, что позволило использовать ее и для других проблем глубокого обучения.

Предпосылкой создания такой архитектуры стало несовершенство использования рекуррентных нейронных сетей (RNN), котрые являлись основным методом работы с последовательностями. Среди их минусов можно выделить проблемы, описанные ниже.

1. В RNN-сетях информация о последовательности находится в закрытом состоянии. При этом обновление такого состояние происходит с каждым новым шагом. При необходимости модели вернуться к той информации, которая была получена многими шагами ранее, данные должны быть сохранены внутри самого скрытого состояния без изменений. Это приводит к необходимости выбора: либо хранить большой объем информации для скрытого состояния, либо отказаться от ее хранения совсем, следовательно, потерять эти данные.

2. Обучение RNN-сетей трудно поддается процессу распараллеливания. Это связано с тем, что для получения скрытого состояния на $i + 1$-ом шаге возникает необходимость получения состояния на i-ом шаге. В результате для батча примеров длиной в 1000 обработка будет составлять 1000 последоватлеьных действий. Такая процедура занимает довольно много времени, а также демонстрирует низкую эффективность при работе на GPU, которые используются при параллельных вычислениях.

Таким образом, описанные проблемы осложняют использование рекуррентных сетей для дейчтвительно длинных последовательностей. Требуется подход, который мог бы считывать последовательность таким образом, что-

бы вернуться к любому прошедшему моменту можно было за определенное время, сохранив исходную информацию. Этим свойством обладает структура self-attention («внимание к себе»), которая лежит в основе архитектуры Transformer.

Большая часть моделей, используемых для преобразования последовательностей, имееют структуру кодер-декодер [12]. Здесь кодировщик отвечает за отображений входной последовательности представлений символов $(x_1, ..., x_n)$ в последовательность непрерывных представлений $z = (z_1, ..., z_n)$. С учетом z декодер затем генерирует вывод последовательности $(y_1, ..., y_m)$ символов по одному элементу за раз. Transformer следует этой общей архитектуре. Общая схема сети Transformer представлена на рис. 2.

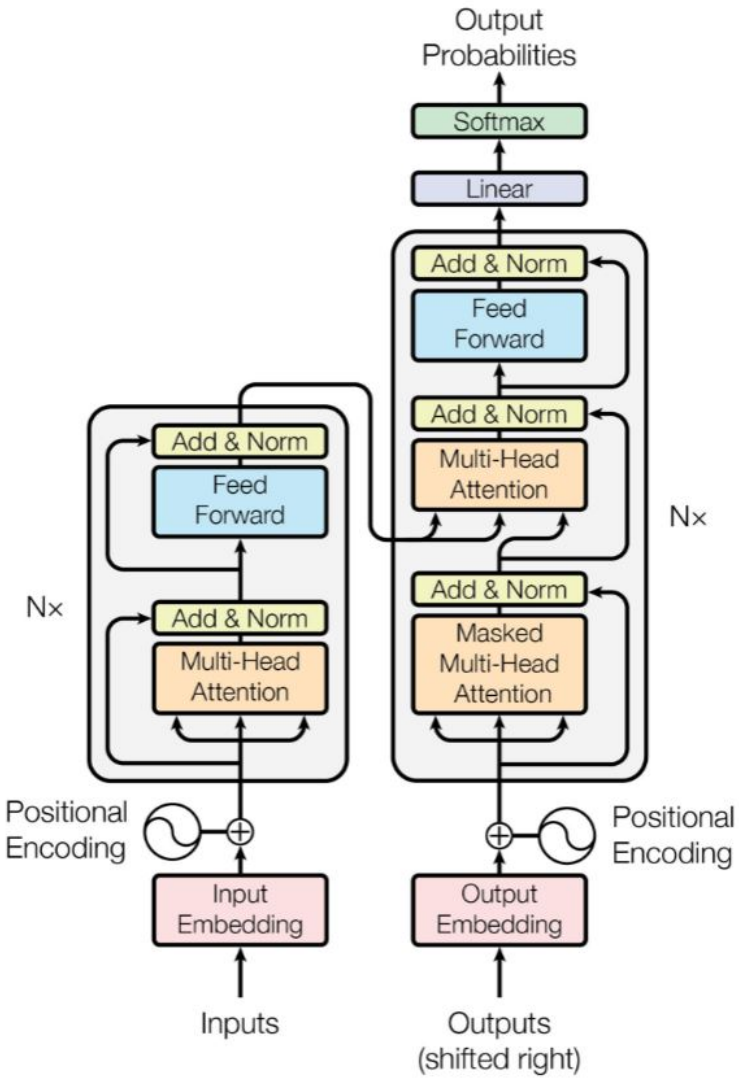


Рис. 2. Архитектура Transformer-сети

Левая часть схемы представляет собой структуру энкодера. Он является

стеком из идентичных слоев, каждый из которых имеет два подслоя [13]. Первый из них – multi-head self-attention («механизм самоконтроля с несколькими головами»). Второй – это простая позиционная полносвязная нейронная сеть прямого распространения. После прохождения каждого из этих слоев выход и вход складываются (такой подход получил название residual connection), а затем следует слой layer normalization (блок «Add & Norm» на рис. 2).

Декодерная часть также представляется несколькими слоями. В дополнение к двум подуровням в каждом слое кодера, декодер вставляет третий подуровень, в котором используются выходы энкодера. Аналогично кодирующей части, происходит использование остаточных соединений для каждого из подслоев с последующей нормализацией слоя.

Проведем более детальный разбор основных составляющих архитектуры Transformer.

Слой внимания. Как было сказано выше, в сети трансформера используется слой self-attention [14]. Данный слой отличается от обычного внимания тем, что на выходе получаются другие представления элементов последовательности, которая была подана изначально. При этом происходим взаимодействие элементов между собой.

Более детально, при вычислении внимания используется обучение трех матриц W_Q, W_K, W_V.Происходит умножение каждой из этих матриц на представление x_i для каждого элемента последовательности на входе. При этом получаются вектора q_i, k_i, v_i, где i соответствует номеру элемента, причем

1) q_i является запросом к базе данных;

2) k_i – это ключи, хранящиеся в базе; по ним будет происходить поиск;

3) v_i отвечает самим значениям.

Понять, насколько близко значение ключа к запросу можно с использованием соотношения 3.2:

$$self - attention - weights_i = softmax\left(\frac{q_i k_1 T}{C}, \frac{q_i k_2 T}{C}, ...\right), \tag{3.2}$$

где C – нормировочное константное значение. При этом в качестве C может быть выбран квадратный корень $\sqrt{d_k}$ из размерности ключей и значений.

Суммирование v_i с новыми полученными коэффициентами являются вы-

ходом для $self - attention$-слоя. В матричном виде это можно представить выражением 3.3:

$$self - attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{3.3}$$

Слой Multi-head attention. Конкретный набор значений Q, K, V отовечает конкретному виду соотношений для токенов, данные матрицы получают при этом ограниченных набор данных из входных. Решением данной проблемы стал multi-head-attention подход, который заключается в том, что один слой внимания заменяется сразу несколькими параллельными, в каждом из которых используются свои веса. После этого результаты объединяются. Схематично данный слой представлен на рис. 3.

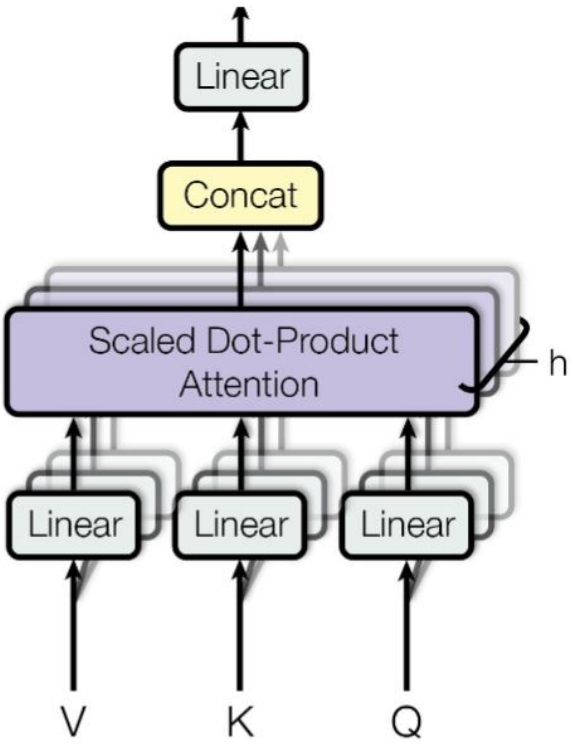


Рис. 3. Слой multi-head attention

Полносвязный слой. Следующая часть блока сети Transformer – полносвязная нейронная сеть прямого распространения (feed-forward network, FFN). Она включает в себя два полносвязных слоя, каждый из которых может независимо применяться к любому элементу для последовательности входа. В последнее время для архитектур выходное значение после прохождения первого слоя (промежуточное значение) может превышать выходы для

блока в четыре раза. В связи с этим следует учесть, что для больших моделей FFN может работать значительно дольше, чем self-attention.

Сеть Vision transformer

Одним из наиболее значимых примеров того, где нашли применение Transformer-модели, стало компьютерное зрение. Архитектура, называемая Vision transformer, в свое время продемонстрировала наилучшие результаты в плане качества для задач классификации изображений. Этому поспособствовала реализация self-attention-идеи для изображений, которые были разделены на множество квадратных сегментов. В данном подходе изображения представляются в виде визуальных токенов, после чего подаются на вход сети.

Общий алгоритм работы сети для решения задач компьютерного зрения представляется следующим образом:

1) получение карты признаков (feature map) для исходного изображения;

2) конвертация карт признаков в визуальные токены;

3) подача токенов на вход сети трансформера.

Выход сети может быть использван для решения задач классификации. В том случае, если объеденить выход с картой признаков, то может быть реализована задача сегментации.

Архитектура Vision transformer (ViT) представляет собой комбинацию трех частей – токенизатора, трансформера и проектора.

Первый компонент, токенизатор, как следует из названия, отвечает за извлечение токенов. Второй осуществляет прохождение через сеть, которая была описана ранее, а третий производит сложение feature map и выхода трансформера. Схема ViT представлена на рис. 4.

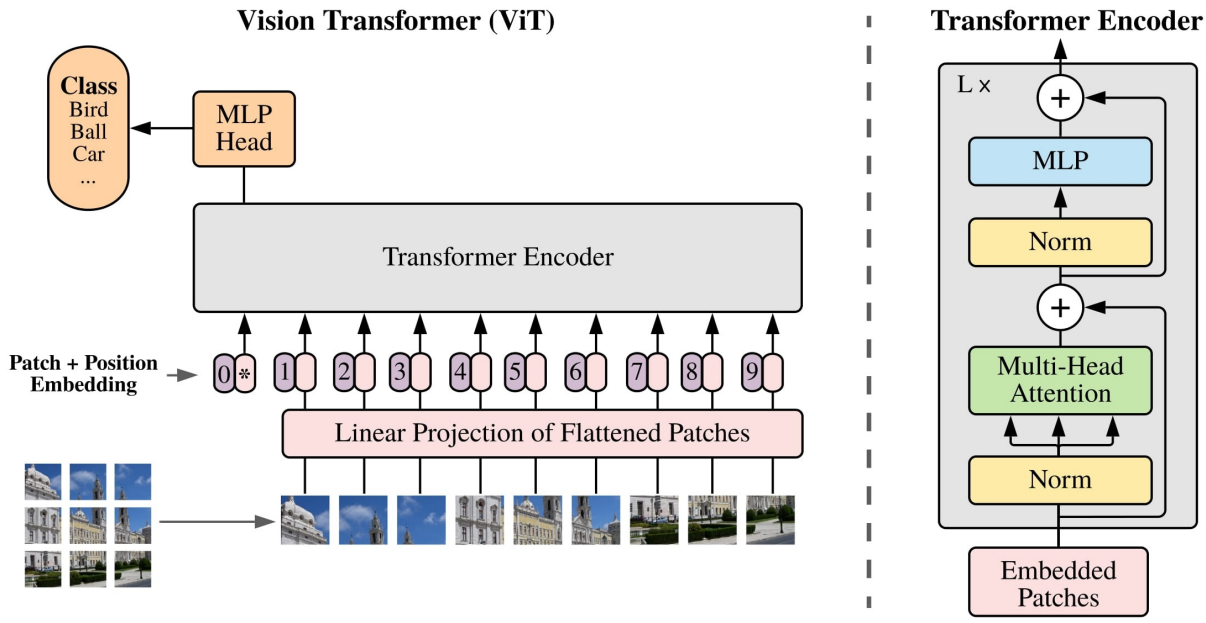


Рис. 4. Схема сети ViT

Сеть DETR

Еще одной модификацией, которая используется для обнаружения, является применение сети DETR (Object Detection with Transformers). В этом случае так же, как и для сети ViT, трансформер применяется не к самому изображению, а к признакам, которые были выделены сверточной сетью. Схема работы для сети DETR представлена на рис. 5.

Однако данный подход имеет ряд существенных недостатков:

1) самая важная проблема – долгое и сложное обучение сети;

2) сеть DETR не является универсальной; существует ряд задач, в которых другие подходы проявляют себя значительно лучше

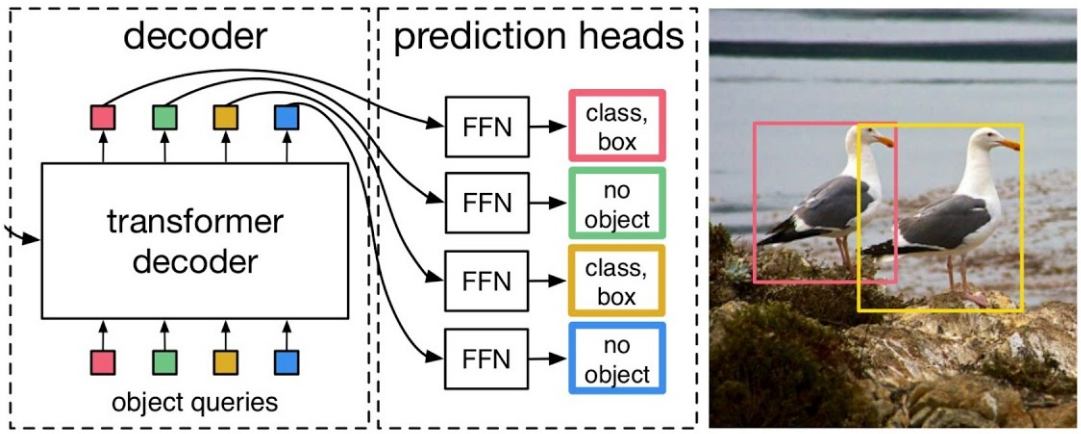


Рис. 5. Схема сети DETR

3.2 Архитектура YOLO

Другой архитектурой, применяемой при решении задачи обнаружения, является YOLO-сеть («You Only Look Once»). Данный алгоритм показал отличные возможности по скорости работы, что позволяет распознавать образы на изображении в режиме реального времени. Дословно аббревиатуру YOLO можно перевести как «стоит лишь раз взглянуть», такое название дано сети, поскольку изображение проходит через сверточную сеть только однажды. Другие же алгоритмы предусматривают многократное прохождение через CNN, что замедляет их работу.

Архитектура сети YOLO представлена на рис. 6.

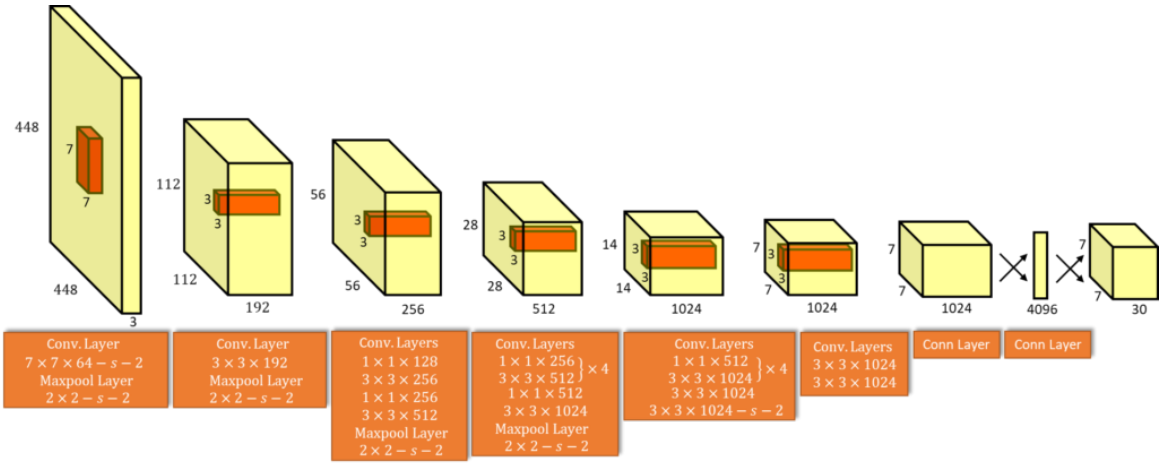


Рис. 6. Архитектура сети YOLO

Как было сказано выше, YOLO использует сверточные нейронные сети для извлечения признаков из изображении. Обычно используются предобученные сверточные сети, такие как Darknet, для извлечения признаков и последующего применения полносвязных слоев для предсказания ограничивающих рамок и классов. Данная модель также использует несколько слоев обнаружения на разных масштабах для обнаружения объектов разного размера. Более крупные объекты обнаруживаются на более высоких слоях, а меньшие объекты – на более низких слоях. Это позволяет YOLO эффективно обнаруживать объекты разного масштаба на изображении.

В методе YOLO результаты обнаружения представляются в виде тензора размером $7 \times 7 \times 1024$. Оценка вероятности нахождения объекта конкретного класса в текущем обрамляющем прямоугольнике – это произведение оценки

вероятности нахождения объекта в ячейке и оценки вероятности для конкретного класса.

Основная идея YOLO-архитектуры заключается в обнаружении как задаче регрессии. Изначально изображение с использованием сети разбивается на несколько ячеек. При его прохождении через CNN может быть получено как местоположение объекта, так и его категория и соответствующая доверительная вероятность. Таким образом, имеется сетка, каждый элемент (ячейка) которой должна определить следующее:

1) координаты t_x, t_y, t_w, t_h ограничивающей рамки, а также значение, определяющее, присутствует ли искомый объект в ячейке (так называемый «индекс объектности»);

2) несколько вероятностных классов.

Определение данных значений происходит методом $k-$средних. Пусть координаты (c_x, c_y) соответствуют левому верхнему углу. Тогда координаты центра (b_x, b_y) ограничивающей рамки, а также ее ширина b_w и высота b_h могут быть предсказаны, исходя из выражений (3.4):

$$\begin{aligned} b_x &= \sigma(t_x) + c_x, \\ b_y &= \sigma(t_y) + c_y, \\ b_w &= p_w e^{t_w}, \\ b_h &= p_h e^{t_h}, \end{aligned} \tag{3.4}$$

где p_w и p_h – ширина и высота привязки, $\sigma(\cdot)$ – сигмоидная функция активации. Графичсеки смысл используемых обозначений представлен ниже на рис. 7.

При использовании данной архитектуры может возникнуть следующая проблема: получение нескольких предсказанных ограничивающих рамок для одного класса. Можно было бы оставлять только одну рамку, соответствующую наибольшей вероятности нахождения в ней объекта, однако такой вариант неприменим в том случае, если на изображении имеется несколько экземпляров одного класса. С целью решения данной проблемы используется метод «немаксимального подавления». Изначально выбирается рамка с наибольшей вероятностью и сравнивается со всеми другими для конкретно-

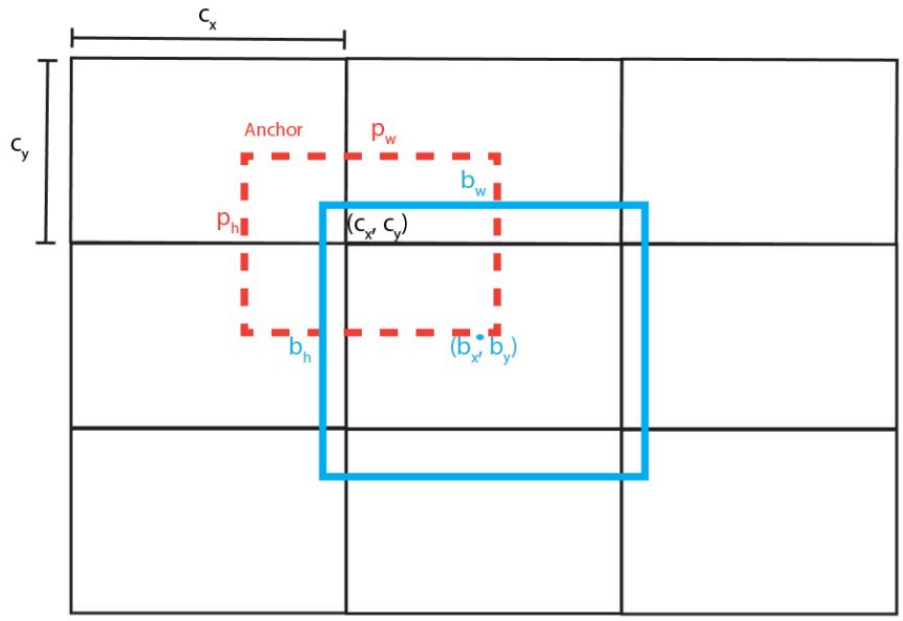


Рис. 7. Архитектура сети YOLO

го класса. Данная продедура использует IoU-метрику. В том случае, когда значение IoU меньше заданного порога (к примеру, 0.5), ограничивающий прямоугольник, соответсвующий меньшей вероятности, исключается из рассмотрения (ситуация, когда два прямоугольника указывают на один и тот же объект).

4 МОДЕЛИРОВАНИЕ НЕЙРОННОЙ СЕТИ

4.1 Описание используемого датасета

Реализация задачи обнаружения объектов в данной работе производилась на готовом датасете Car Object Detection, выгруженного с сайта, предлагающего готовые наборы данных для машинного обучения https://www.kaggle.com [15]. Указанный набор содержит медиафайлы, снятые в условиях дорожной обстановки. На кадрах фиксировались автомобили. Исходные данные представляют собой два видеоряда, каждый из которых был разрезан на стоп-кадры (которые и являются исходными изображениями). Общее количество изображений в датасете равно 1001.

Суть задачи сводится к тому, чтобы протестировать различные нейросетевые архитектуры, для обнаружения на изображении автомобилей. В датасете также имеются тренировочные изображения – каждый автомобиль на них отмечен своей ограничивающей рамкой.

Набор представляет собой изображения трех типов:

1) кадры, на которых имеется только один автомобиль;

2) кадры, на которых имеется несколько авиомобилей;

3) кадры, на которых нет автомобилей.

Максимальное количество машин на изображении равно семи. Таким образом, задача усложняется тем, что исходное количество объектов на изображении заранее неизвестно.

4.2 Предобработка датасета для сети ViT

У трансформеров есть несколько ограничений при использовании. Одно из них связано с ограниченным фиксированным выходом из нейронной сети. Для того, чтобы избежать этой проблемы количество выходных объектов K устанавливается значительно больше, чем максимальное ожидаемое количество объектов на изображениях.

Для возможности сопоставления истинных меток с предсказанными нейронной сетью, датасет был преобразован способом, описанным ниже.

1. Координаты каждой ограничивающей рамки были преобразованы из абсолютных размеров, измеряемых в пикселях, в относительные по отношению к размерам изображения;
2. Полученные координаты переведены из описания левого верхнего угла (x_{min}, y_{min}) и правого нижнего (x_{max}, y_{max}) ограничивающей рамки в описание центра (x_c, y_c) и размеров изображения (h, w);
3. Для каждого изображения все соответствующеи ему ограничивающие рамки были объединены в массив, который дополнили нулевыми ограничивающими рамками до размера массива равного K;
4. Каждой ограничивающей рамке была добавлена еще одна координата, соответствующая нахождению в ней объекта поиска, т.е. для ненулевых рамок 1, для остальных – 0.

Таким образом размерность метки для одного изображения равняется $(K, 5)$, при этом некоторые из рамок будут заполнены нулями.

4.3 Венгерский алгоритм

Постановка задачи о назначениях

Задача о назначениях [16] – это задача оптимизации, заключающаяся в поиске оптимального соответствия между двумя наборами элементов. В общем виде задача о назначениях формулируется следующим образом.

Имеется n работ и n кандидатов для их выполнения. Затраты i кандидата на выполнение j работы равны c_{ij}. Каждый кандидат может быть назначен только на одну работу, и каждая работа может быть выполнена только одним кандидатом. Требуется найти назначение кандидатов на работы, при котором суммарные затраты на выполнение работ минимальны.

Для построения математической модели вводятся переменные x_{ij}, принимающие значение 1 или 0 в зависимости от того, назначен или нет i механизм на j работу соответственно.

Тогда условие о том, что каждый кандидат выполняет только одну работу, запишется в виде

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = \overline{1,n}. \tag{4.5}$$

Условие о том, что каждая работа может выполняться одним кандидатом, запишется в виде

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = \overline{1,n}. \tag{4.6}$$

Целевая функция примет вид

$$F = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}. \tag{4.7}$$

В функцию войдут только те значения c_{ij}, для которых x_{ij} отличны от 0, т.е. затраты, соответствующие назначенным работам.

В результате математическую модель задачи о назначениях можно опи-

сать следующей системой уравнений

$$\begin{cases} F = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \longrightarrow min, \\ \sum_{j=1}^{n} x_{ij} = 1, \quad \forall i = \overline{1,n}, \\ \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j = \overline{1,n}, \\ x_{ij} \in \{0,1\}, \quad \forall i = \overline{1,n}, \forall j = \overline{1,n}. \end{cases} \tag{4.8}$$

Алгоритм решения

При решении задачи используется тот факт, что оптимальность не нарушается при уменьшении или увеличении строк и столбцов матрицы $C = \{c_{ij}\}$ на одно и тоже число. Основная идея венгреского алгоритма решения задачи о назначениях состоит в переходе от матрицы C к матрице C' такой, которая состоит из неотрицательных элементов и n независимых нулей (никакие два не пренадлежат одной строке или одному столбцу матрицы).

Алгоритм состоит из следующих этапов [17].

1. В каждой строке таблицы C найти наименьший элемент и вычесть его из всех элементов данной строки.

$$c_{ij} = c_{ij} - \min_{j}(c_{ij}), \qquad \forall i = \overline{1,n} \tag{4.9}$$

2. В каждом столбце полученной таблицы найти наименьший элемент и вычесть его из всех элементов данного стобца.

$$c_{ij} = c_{ij} - \min_{i}(c_{ij}), \qquad \forall j = \overline{1,n} \tag{4.10}$$

3. Найти строку, содержащую наименьшее количество нулей. Отметить один из нулей в этой строке. Остальные нули в этой строке и столбце зачеркиваем. Затем снова найти строку с наименьшим количеством незачеркнутых нулей и повторить процедуру среди незачеркнутых нулей.
4. Повторить действия из 3 пункта n раз. Если в результате количество отмеченных нулей совпадает с n, то получено оптимальное решение. В противном сучае перейти к шагу 5

5. Отметить все строчи, не содержащие отмеченные нули. Отметить все столбцы с нулями в этих строчках. Отметить все строчки, содержащие отмеченные нули. Повторять последние 2 шага, пока новые строки и столбцы не перестанут отмечаться. Провести линии через все отмеченные столбцы и неотмеченные строчки. Таким образом, на данном шаге покроются все нули минимальным количеством линий.
6. Из непокрытых линиями элементов матрицы найти наименьший, вычесть его из непокрытых элементов и прибавить к элементам на пересечении линий. После проделанного количество нулей, отмечаемых на шаге 3 должно вырасти минимум на 1. Вернуться к шагу 3.

В результате данного алгоритма матрица C будет преобразована в C', а решением $X = \{x_{ij}\}$ будет матрица, элементы которой равны единице, если c'_{ij} – ноль, отмеченный в пункте 3.

4.4 Функция потерь сети ViT и сопоставление истинных и предсказанных меток

В качестве функции потерь была выбрана следующая функция:

$$L\left(y_{true}, y_{pred}\right) = \lambda_1 BinCrossEntropy\left(p_{true}, p_{pred}\right) + \\ + I^{obj} L_{bbox}\left(bbox_{true}, bbox_{pred}\right), \quad (4.11)$$

где $y_{true} = (bbox_{true}, p_{true})$ – истинная метка, $y_{pred} = (bbox_{pred}, p_{pred})$ – метка, предсказанная нейросетью, $bbox_{true}$, $bbox_{pred}$ – ограничивающие рамки объекта, p_{true}, p_{pred} – вероятность нахождения объекта в ограничивающей рамке, λ_1 – гиперпараметр, I^{obj} – индикатор наличия объекта, т.е. $y_{true} = 1$.

Функции $BinCrossEntropy\left(p_{true}, p_{pred}\right)$ и $L_{bbox}\left(bbox_{true}, bbox_{pred}\right)$ определяются следующим образом:

$$BinCrossEntropy\left(p_{true}, p_{pred}\right) = -p_{true}\log(p_{pred}) - \\ - (1 - p_{true})\log(1 - p_{pred}), \quad (4.12)$$

$$L_{bbox}\left(bbox_{true}, bbox_{pred}\right) = -\lambda_2 \log(IoU\left(bbox_{true}, bbox_{pred}\right)) + \\ + \lambda_3 \parallel bbox_{true} - bbox_{pred} \parallel_1, \quad (4.13)$$

где λ_2, λ_3 – гиперпараметры, $\| \cdot \|_1$ – L1 норма.

IoU (Intersection over Union) – площадь пересечения ограничивающих рамок, деленная на площадь их объединения.

Для того чтобы определить соответствие между объектами на входном и выходном изображениях, используется венгерский алгоритм. Он находит наилучшее соответствие между объектами таким образом, чтобы минимизировать общую ошибку. Его использование реализовано следующим образом:

1. Для каждого y_{true} и y_{pred} вычисляется значение функции потерь L;
2. На полученной матрице C размера $K \times K$, используется венгерский алгоритм, описанный в главе 4.3.

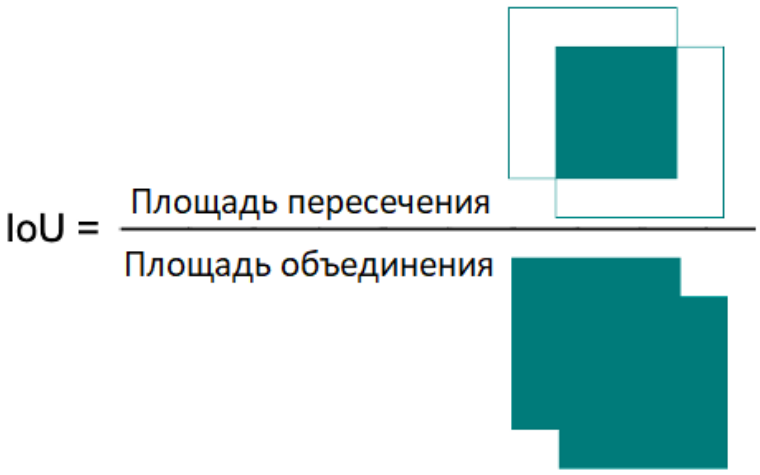


Рис. 8. IoU

4.5 Метрики сети ViT

Поскольку сети ViT используются нестандартные выходы из сети, то для оценки качества сети необходимо было разработать особые метрики. Отдельно стоит сказать, что все метрики расчитываются для перестановки выходов, оптимальной для функции потерь данной сети.

Использовались следующие метрики:

- Точность нахождения объектов – метрика, описывающая долю правильно предсказанных p_{pred}. Формула, по которой вычисляется метрика, представлена ниже.

$$P = \frac{\sum_{i=1}^{K} \delta_i}{K}, \tag{4.14}$$

где δ_i – символ Кронекера

$$\delta_i = \begin{cases} 1, & \text{если } (p_{pred})_i = (p_{true})_i, \\ 0, & \text{иначе.} \end{cases} \tag{4.15}$$

Таким образом, данная метрика является поэлементной точностью, но только для вероятности нахождения объекта

- Среднее IoU – среднее арифметическое значение метрики IoU среди выходов, где $p_{true} = 1$. Если данное условие ни разу не выполняется, то считаем, что значение метрики равно единице. Алгоритм расчета метрики IoU для одной ограничивающей рамки:

  1. Перевести координаты ограничивающей рамки из описания центра объекта, ширины и высоты к координатам левого верхнего и правого

нижнего углов:

$$bbox_{old} = (x_c, y_c, h, w), \tag{4.16}$$

$$bbox_{new} = (x_{min}, y_{min}, x_{max}, y_{max}). \tag{4.17}$$

2. Найти координаты левого верхнего и правого нижнего угла пересечения предсказанной ограничивающей рамки с истинной. Для этого для каждой из координат находим максимум из минимальных координат и минимум из максимальных:

$$x_{min}^{inter} = max(x_{min}^{true}, x_{min}^{pred}), \tag{4.18}$$

$$y_{min}^{inter} = max(y_{min}^{true}, y_{min}^{pred}), \tag{4.19}$$

$$x_{max}^{inter} = min(x_{max}^{true}, x_{max}^{pred}), \tag{4.20}$$

$$y_{max}^{inter} = min(y_{max}^{true}, y_{max}^{pred}). \tag{4.21}$$

3. Вычислить ширину и высоту пересечения:

$$h^{inter} = x_{max}^{inter} - x_{min}^{inter}, \tag{4.22}$$

$$w^{inter} = y_{max}^{inter} - y_{min}^{inter}. \tag{4.23}$$

4. Если ширина или высота получились меньше либо равны нуля, то площадь пересечения равняется нулю, поскольку это значит, что ограничивающие рамки не пересекаются. Иначе просто вычисляем площадь пересечения.

$$S^{inter} = \begin{cases} h^{inter} w^{inter}, & \text{если } h^{inter} > 0, w^{inter} > 0; \\ 0, & \text{иначе.} \end{cases} \tag{4.24}$$

5. Найти площадь объединения. Для этого необходимо посчитать площади каждой ограничивающей рамки, сложить и вычесть площадь пересечения.

$$S^{union} = h^{true} w^{true} + h^{pred} w^{pred} - S^{inter}. \tag{4.25}$$

6. Вычислить значение метрики IoU, поделив площадь пересечения на площадь объединения.

$$IoU = \frac{S^{inter}}{S^{union}}. \tag{4.26}$$

После вычисления метрики IoU для каждой ограничивающей рамки рассчитывается среднее IoU по формуле

$$IoU_{avg} = \frac{\sum_{i=1}^{K} \delta_i IoU_i}{\sum_{i=1}^{K} \delta_i}, \tag{4.27}$$

где δ_i – символ Кронекера

$$\delta_i = \begin{cases} 1, & \text{если } (p_{true})_i = 1, \\ 0, & \text{иначе.} \end{cases} \tag{4.28}$$

Совмещая данные две метрики, в дальнейшем строилась оценка обучения сети ViT.

4.6 Слои и структуры сети ViT

Полносвязный слой

Полносвязные слои (Fully Connected или Dense layers) - это тип слоев в нейронных сетях, который применяет линейную функцию к каждому входному элементу (нейрону) и создает новые признаки с заданной размерностью.

Пусть вход в слой имеет размерность (N, M), где N - это количество примеров (например, изображений), а M - это размерность каждого примера (например, количество пикселей в изображении). Полносвязный слой преобразовывает каждый пример размерности (1, M) в новый пример размерности (1, K), где K - это заданное число новых признаков. Этот процесс происходит путем умножения входа на матрицу весов размерности (M, K), после чего к результату прибавляется вектор смещения размерности (1, K). В общем случае, формула для вычисления выхода из полносвязного слоя будет выглядеть следующим образом:

$$y = Wx + b,$$

где x – входной тензор, W – матрица весов, b – вектор смещения, а y – выходной тензор.

После вычисления выхода из полносвязного слоя, к результату может быть применена некоторая нелинейная функция активации, такая как ReLU, Sigmoid, Tanh и другие. Нелинейная функция активации обычно используется для того, чтобы преобразовать выход полносвязного слоя в нелинейный диапазон значений и добавить нелинейность в модель, что может улучшить ее способность обобщения и улучшить качество предсказаний.

Слой Dropout

Слой Dropout - это техника регуляризации, которая используется в нейронных сетях для предотвращения переобучения. Идея Dropout заключается в том, чтобы случайно удалять (выключать) некоторые нейроны в каждом слое, во время обучения сети.

Каждый нейрон в слое Dropout имеет вероятность p (обычно 0.5) быть

выключенным в каждом проходе обучения. При этом, выходы всех нейронов будут умножены на 1/(1-p), чтобы уравнять суммарную величину выходных данных слоя.

Таким образом, когда происходит проход обучения сети, Dropout выключает случайно выбранные нейроны в каждом проходе. Это заставляет нейроны в слое работать более самостоятельно, а не перенаправляться на зависимости, которые могут быть связаны с конкретными нейронами. Dropout также заставляет сеть учитывать более разнообразное представление входных данных, что обычно приводит к улучшению обобщающей способности модели.

После обучения, Dropout отключается, и все нейроны начинают работать с соответствующими весами. В этом случае, устанавливается режим "inference"(применения), и каждый нейрон является активным.

Слой Dropout часто используется вместе с полносвязными слоями (Dense layers) или сверточными слоями (Convolutional layers) в глубоких нейронных сетях, особенно в случае, когда обучение сети сталкивается с проблемой переобучения. Dropout позволяет контролировать сложность модели, предотвращает переобучение и улучшает ее обобщающую способность.

Слой Patches

В реализации сети Transformer с архитектурой ViT каждое изображение делится на патчи. Для этого создается специальный слой Patches. На вход данный класс будет принимать только размеры самих патчей. При его вызове в него передается всё входное изображение. Далее это изображение будет разделено на патчи указанного размера и каждый из них будет вытянут в одномерный вектор признаков.

Слой PatchEncoder

Для каждого из патчей важно помнить правильное их расположение. Поэтому в сети используется слой PatchEncoder. На вход данный класс будет принимать количество патчей изображения и размерность обучаемых эмбе-

дингов. Эти эмбединги необходимы для кодирования номера патча в разделенном изображении. Каждый патч проходит через полносвязный слой с количеством выходов равным размерности эмбедингов. К полученному результату прибавляется значение эмбединга соответствующее номеру патча в изображении. Таким образом на выходе слоя PatchEncoder будет вектор с размерностью эмбедингов для каждого патча.

4.7 Предобработка датасета для сети YOLO

Для обучения нейронной сети YOLO необходимо метки каждого изображения привести к тензору формой $(S,\ S,\ num_{anc},\ 5)$, где S – размер сетки, на который разбиваем изображение, num_{anc} – количество якорных рамок.

Далее представлен алгоритм преобразования меток изображения:

1. координаты каждой ограничивающей рамки преобразовать из абсолютных размеров, измеряемых в пикселях, в относительные по отношению к размерам изображения;

$$(x_{min}, y_{min}, x_{max}, y_{max}) = \left(\frac{X_{min}}{H}, \frac{Y_{min}}{W}, \frac{X_{max}}{H}, \frac{Y_{max}}{W}\right) \tag{4.29}$$

2. полученные координаты перевести из описания левого верхнего угла (x_{min}, y_{min}) и правого нижнего (x_{max}, y_{max}) ограничивающей рамки в описание центра (x_c, y_c) и размеров изображения (h, w);

$$\begin{cases} x_c &= \dfrac{x_{max} + x_{min}}{2}, \\ y_c &= \dfrac{y_{max} + y_{min}}{2}, \\ h &= x_{max} - x_{min}, \\ w &= y_{max} - y_{min}. \end{cases} \tag{4.30}$$

3. каждой ограничивающей рамке добавить еще одну координату, соответствующую нахождению в ней объекта поиска.

$$(x_c,\ y_c,\ h,\ w) \longrightarrow (x_c,\ y_c,\ h,\ w,\ 1) \tag{4.31}$$

4. для каждой ограничивающей рамки посчитать значение метрики IoU с num_{anc} якорными рамками, найти номер рамки ind_{anc}, соответствующий мексимальному значению метрики;
5. найти индексы $(i,\ j)$ ячейки, в которой находится центр ограничивающей рамки.
6. для каждого изображения создать тензор формы $(S,\ S,\ num_{anc},\ 5)$. Первые два индекса вектора соответсвуют индексам сетки изображения, третий – номеру якорной рамки и четвертый – координатам ограничивающей рамки. Таким образом, элементы вектора с первыми тремя

идексами равными (i, j, ind_{anc}) необходимо заполнить координатами соответствующих ограничивающих рамок, а остальные заполнить нулями.

Полученный таким образом тензор и будет выходом сети YOLO.

Входные изображения необходимо поделить на 255 в целях нормализации.

4.8 Функция потерь сети YOLO

В качестве функции потерь была выбрана следующая функция:

$$L\left(y_{true}, y_{pred}\right) = -\left(\lambda_1\ p_{true}\ \log\left(p_{pred}\right) + \lambda_2\left(1 - p_{true}\right)\log\left(1 - p_{pred}\right)\right) + \\ + I^{obj} L_{bbox}\left(bbox_{true}, bbox_{pred}\right), \quad (4.32)$$

где $y_{true} = (bbox_{true}, p_{true})$ – истинная метка, $y_{pred} = (bbox_{pred}, p_{pred})$ – метка, предсказанная нейросетью, $bbox_{true}$, $bbox_{pred}$ – ограничивающие рамки объекта, p_{true}, p_{pred} – вероятность нахождения объекта в ограничивающей рамке, λ_1, λ_2 – гиперпараметры, I^{obj} – индикатор наличия объекта, т.е. $y_{true} = 1$. Функция $L_{bbox}\left(bbox_{true}, bbox_{pred}\right)$ определяется следующим образом:

$$L_{bbox}\left(bbox_{true}, bbox_{pred}\right) = \lambda_3\left[\left(x_{true} - x_{pred}\right)^2 + \left(y_{true} - y_{pred}\right)^2\right] + \\ + \lambda_4\left[\left(\sqrt{h_{true}} - \sqrt{h_{pred}}\right)^2 + \left(\sqrt{w_{true}} - \sqrt{w_{pred}}\right)^2\right], \quad (4.33)$$

где λ_3, λ_4 – гиперпараметры, $bbox_i = (x_i,\ y_i,\ h_i,\ w_i)$, x_{true}, x_{pred}, y_{true}, y_{pred} – координаты центра ограничивающей рамки, h_{true}, h_{pred}, w_{true}, w_{pred} – размеры ограничивающей рамки.

4.9 Метрики сети YOLO

Было решено использовать две метрики для оценки качества сети YOLO. Первая для оценки предсказания наличия объекта, а вторая – границ объекта.

- В данных огромный перевес метки "0" – отсутствия объекта, по сравнению с "1" – наличием объекта. Поэтому если брать в качестве оценки среднее количество правильно угаданных меток, то результат может стать не показательным. Например, если сеть будет постоянно выдавать нули, то она будет права в подавляющем большинстве случаев. Поэтому для оценки предсказания наличия объекта использовалась следующая метрика P.

$$\begin{cases} P_i = \dfrac{\sum_{j=1}^{N} I_i^{p_{pred}} I_i^{p_{true}}}{\sum_{j=1}^{N} I_i^{p_{true}}}, \ i = 0, 1 \ ; \\ P = \dfrac{P_0 + P_1}{2}, \end{cases} \tag{4.34}$$

  где I_i^p – индикатор $p = i$. Таким образом данная метрика есть среднее арифметическое по двум средним для каждого класса.
- Для оценки предсказания границ объекта было решено использовать метрику среднего IoU из раздела 4.5.

5 РЕЗУЛЬТАТЫ МОДЕЛИРОВАНИЯ

5.1 Сеть ViT

В качестве архитектуры сети Transformer была выбрана архитектрура ViT. Для проверки качества обучения сети использовались следующие метрики

1. Средний IoU (Intersection over Union) – Среднее арифметическое отношений пересечения площадей ограничивающих рамок и их объединения;
2. Точность – отношение количества правильно меток присутствия объекта к количеству всех предсказаний.

Обучение сети проводилось 800 эпох. Наилучшее достигнутое значение среднего IoU равняется 54.5%. Для точности – 98.4%. Результаты обучения сети представлены на рис. 9 и рис. 10.

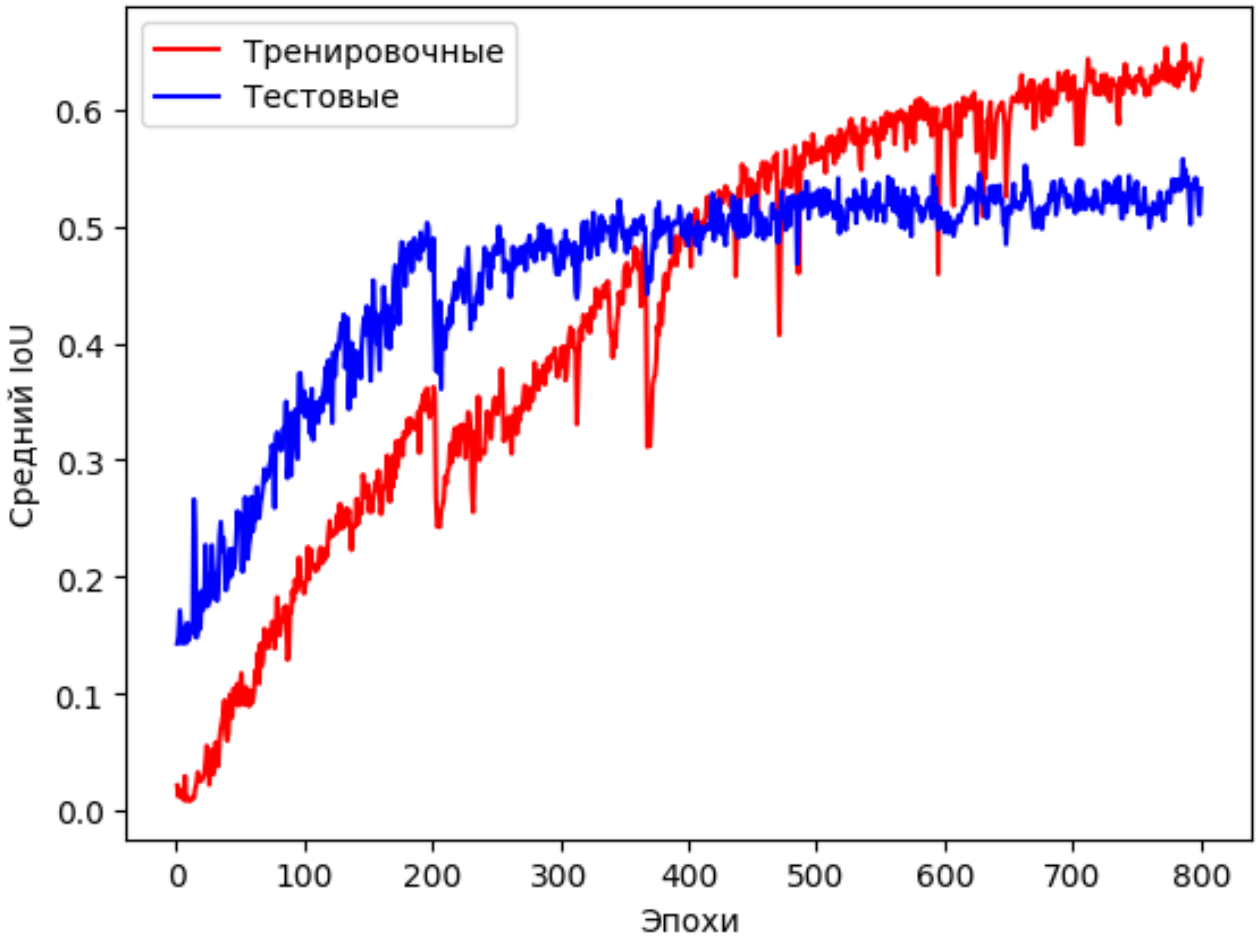


Рис. 9. График среднего IoU алгоритма ViT

Скачки, представленные на графиках, могут быть вызваны за счет использования венгерского алгоритма: поскольку у сети 10 разных выходов, каждый из которых обучается отдельно, в следствие чего в процессе обучения венгерский алгоритм возвращает разные перестановки выходов.

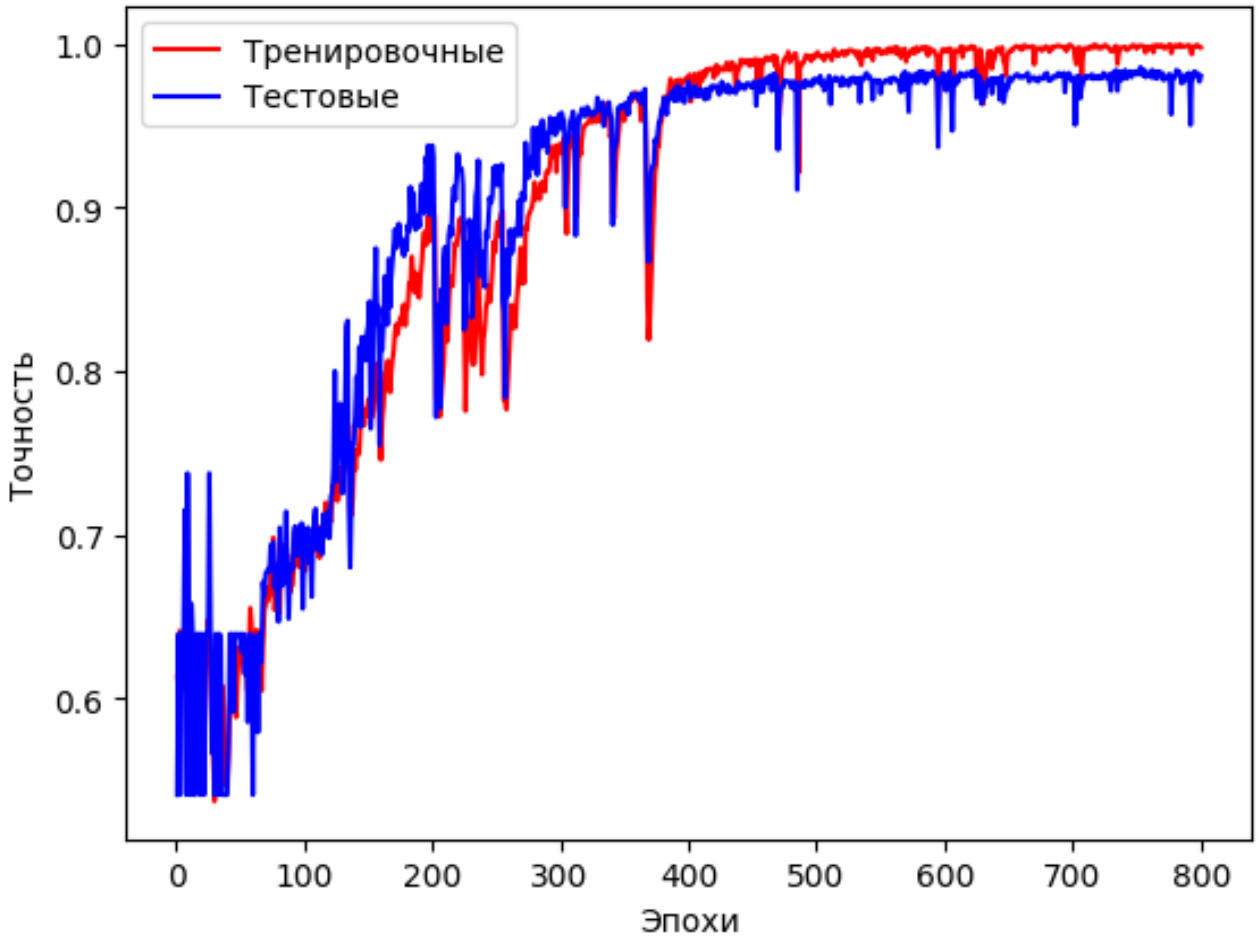


Рис. 10. График точности алгоритма ViT

Средняя скорость обработки сетью ViT составляет 272 изображения в секунду.

Ниже на рис. 11 и рис. 12 представлены примеры обнаружения автомобилей с помощью нейронной сети и сравнение предсказанных ограничивающих рамок с истинными.

Предсказанные рамки

Истинные рамки

Рис. 11. Первый пример обнаружения объектов ViT нейросетью

Предсказанные рамки

Истинные рамки

Рис. 12. Второй пример обнаружения объектов ViT нейросетью

5.2 Сеть YOLO

Обучение сети YOLO проводилось 100 эпох. Наилучшее достигнутое значение среднего IoU равняется 70.1%. Для средней по классам точности - 97.1%. Результаты обучения сети представлены на рис. 13 и рис. 14.

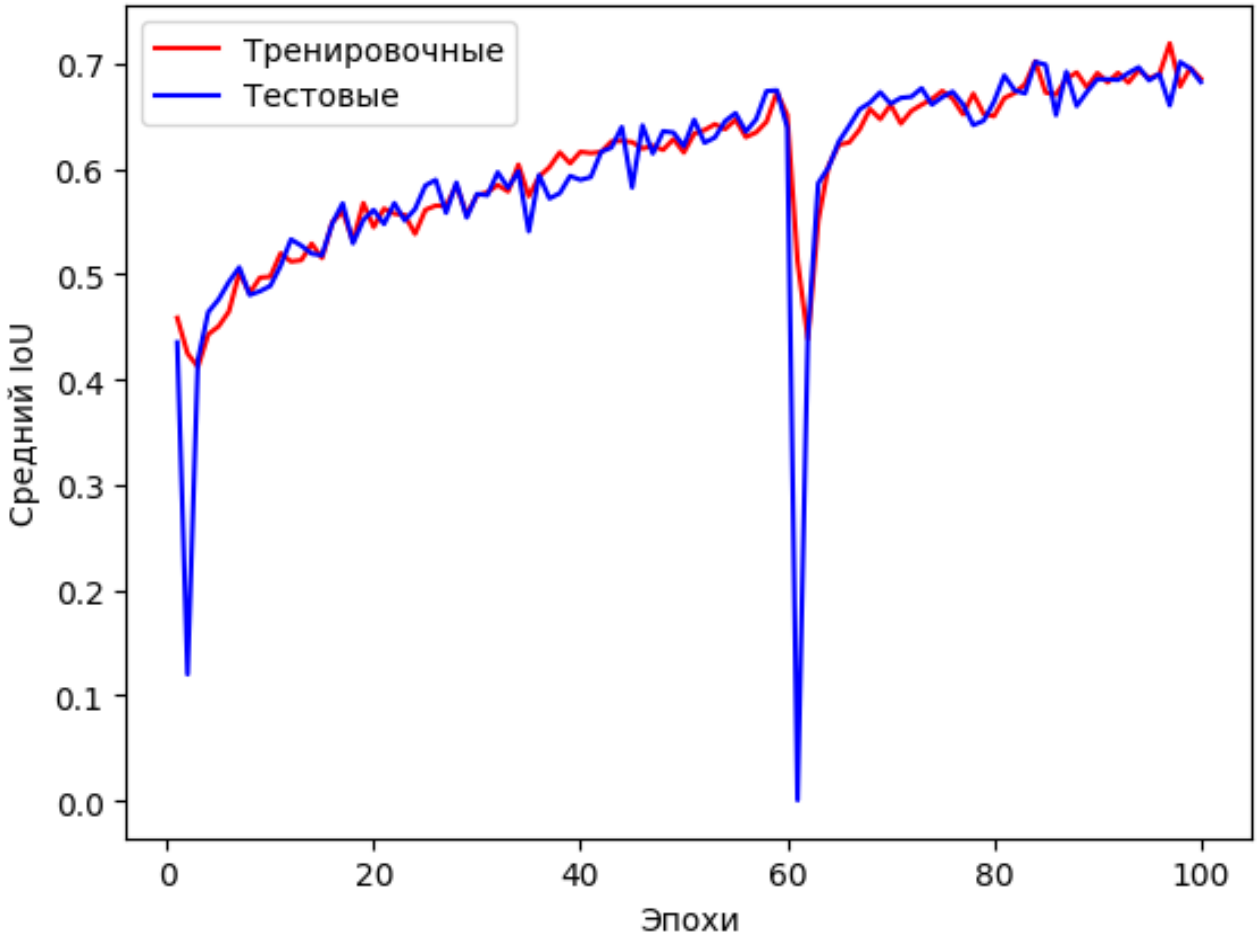


Рис. 13. График среднего IoU алгоритма YOLO

Средняя скорость обработки сетью YOLO составляет 101 изображение в секунду.

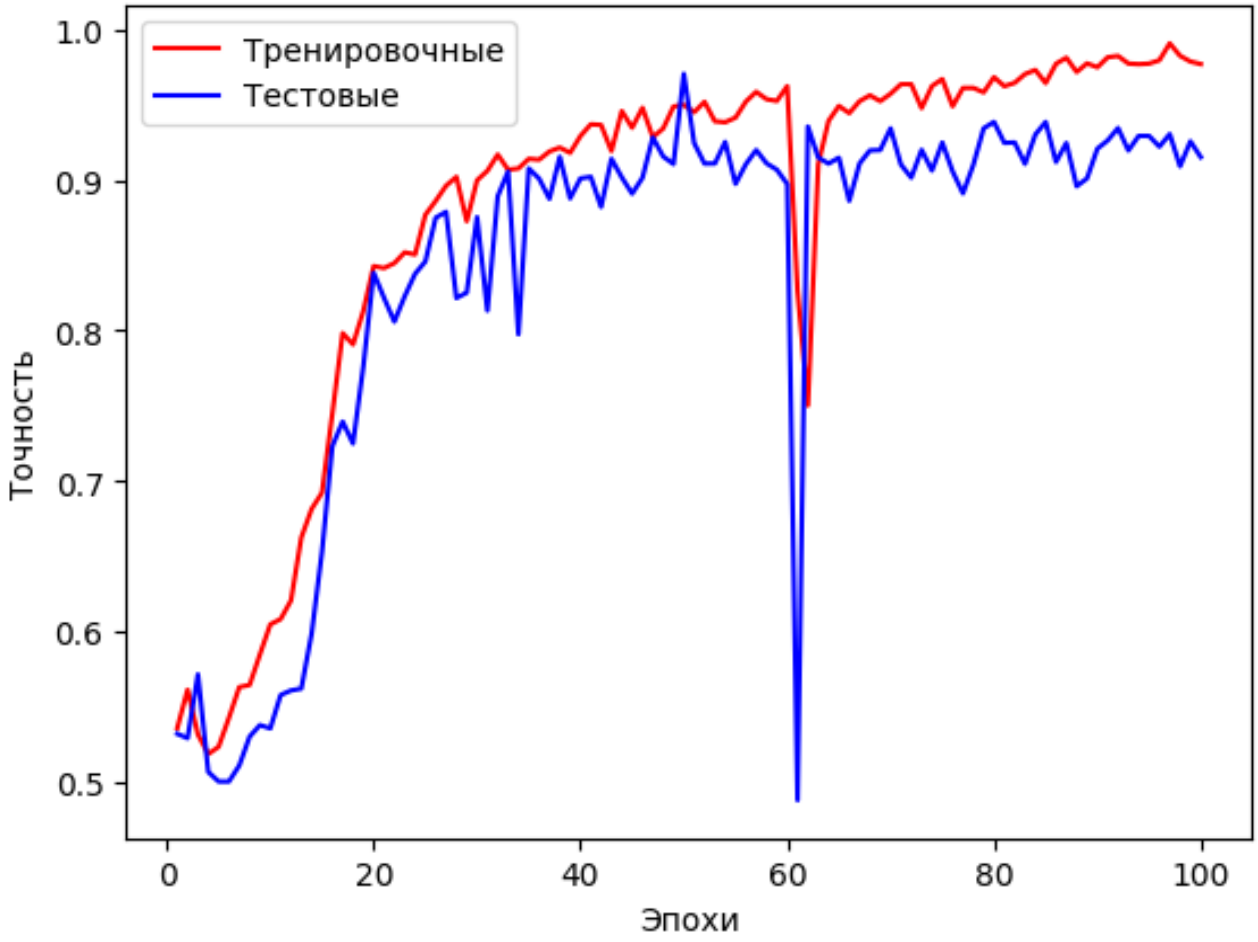
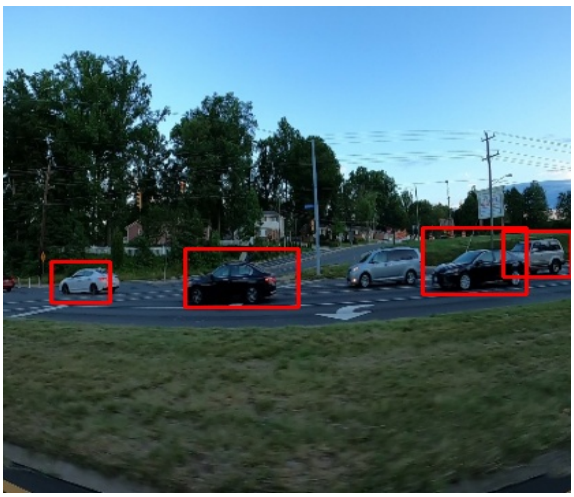
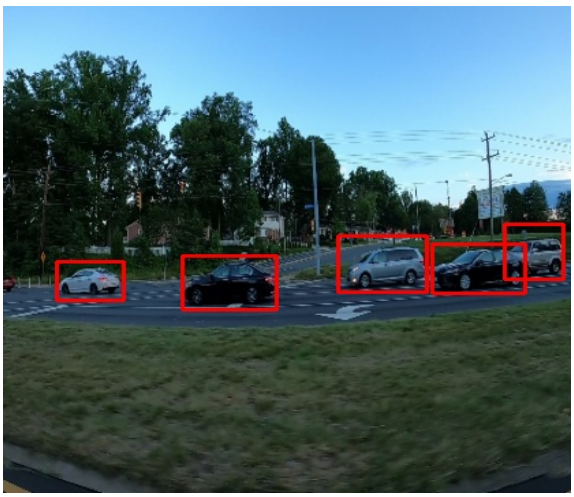


Рис. 14. График точности алгоритма YOLO

Ниже на рис. 15 и рис. 16 представлены примеры обнаружения автомобилей с помощью нейронной сети и сравнение предсказанных ограничивающих рамок с истинными.



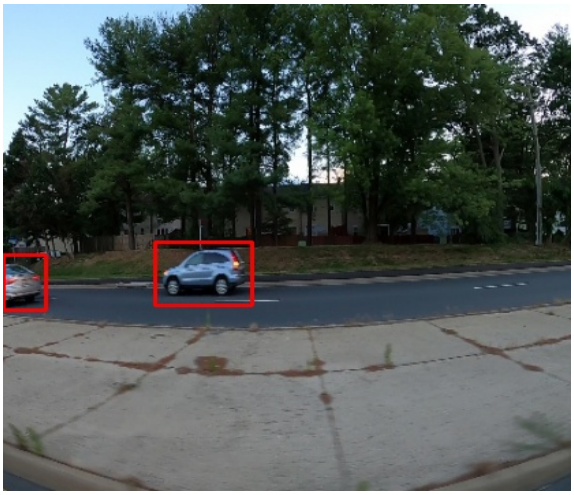
Предсказанные рамки



Истинные рамки

Рис. 15. Первый пример обнаружения объектов YOLO нейросетью

Предсказанные рамки



Истинные рамки

Рис. 16. Второй пример обнаружения объектов YOLO нейросетью

ЗАКЛЮЧЕНИЕ

В настоящей работе было дано определение задаче обнаружения объектов, позволяющая выделять и классифицировать объекты на изображении. Рассмотрены различные методы решения задачи обнаруения объектов. Среди них приведен метод скользящего окна, отличающийся простотой реализации, но долгой работой. Современные методы, основанные на машинном глубоком обучении показали значительный успех в точности обнаружения объектов.

Реализованы два подхода: сеть YOLO с применением сверточных нейронных сетей, якорных рамок и немаксимальным подавлением и сеть ViT с использванием механизма внимания.

Предоставлены графики, полученные в результате обучения данных двух моделей, а так же даны оценки качества и скорости работы.

Дальнейшие исследования в области обнаружения объектов могут быть направлены на разработку новых алгоритмов, учитывающих скорость их работы, специфику изображений и других особенностей, влияющих на результат, а также на изучение комбинаций между собой уже существующих подходов.

Список использованных источников

1. Viola, P., Jones, M. Rapid object detection using a boosted cascade of simple features. // Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. - Kauai: IEEE, 2001. - P. I-I.
2. Viola, P., Jones, M.J. Robust Real-Time Face Detection. // International Journal of Computer Vision. - 2004. - №57. - P. 137–154.
3. Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. // 2014 IEEE Conference on Computer Vision and Pattern Recognition. - Columbus: IEEE, 2014. - C. 580-587.
4. Girshick Ross. Fast R-CNN // 2015 IEEE International Conference on Computer Vision (ICCV). - Santiago: IEEE, 2015. - P. 1440-1448.
5. Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks // IEEE Transactions on Pattern Analysis and Machine Intelligence. - 2017. - Vol.39 - No.6. - P. 1137-1149.
6. J. Redmon, S. Divvala, R. Girshick and A. Farhadi You Only Look Once: Unified, Real-Time Object Detection // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). - Las Vegas: IEEE, 2016. - C. 779-788.
7. SSD: Single Shot MultiBox Detector // arXiv URL: arXiv:1512.02325 (дата обращения: 16.05.2023).
8. Attention Is All You Need // arXiv URL: arXiv:1706.03762 (дата обращения: 16.04.2023).
9. End-to-End Object Detection with Transformers // arXiv URL: arXiv:2005.12872 (дата обращения: 15.04.2023).
10. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale // arXiv URL: arXiv:2010.11929 (дата обращения: 24.04.2023).
11. Муаль М.Н.Б., Козырев Д.В., Уанкпо Г.Ж.К., Нибасумба Э. Разработка нейросетевого метода в задаче классификации и распозна- вании изобра-

жения // Современные информационные технологии и ИТ-образование. – 2021. – Т. 17, № 3. – С. 507-518.

12. Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation // ArXiv URL: https://arxiv.org/abs/1406.1078 (дата обращения: 28.05.2023).
13. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need // arXiv URL: https://arxiv.org/pdf/1706.03762.pdf (дата обращения: 26.05.2023).
14. Yoon Kim, Carl Denton, Luong Hoang, Alexander M. Rush. Structured Attention Networks // 5th International Conference on Learning Representations. - Toulon: OpenReview.net, 2017. - P. 1-21.
15. E. Zhang. Car Object Detection // Kaggle URL: https://www.kaggle.com/datasets/sshikamaru/car-object-detection (дата обращения: 26.05.2023).
16. R.E. Burkard, M. Dell'Amico, S. Martello. Assignment Problems. - Philadelphia: Society for Industrial and Applied Mathematics, 2009. - 382 p.
17. H. W. Kuhn. The Hungarian Method for the assignment problem // Naval Research Logistics Quarterly. - 1955. - T. 2. - P. 83–97.

ПРИЛОЖЕНИЕ А

Ниже представлена програмная реализация метрики точности для сети ViT на языке программирования Python

```
def Hang_p(y_true, y_pred):
    d = tf.shape(y_true)[1]
    obj_true = tf.cast(y_true[...,-1], dtype=tf.float32)
    shapes = tf.cast(tf.shape(obj_true), dtype=tf.float32)
    num_object = tf.reduce_sum(obj_true)
    y_true = tf.expand_dims(y_true, axis=1)
    y_true = tf.tile(y_true, [1, d, 1, 1])
    y_pred = tf.expand_dims(y_pred, axis=2)\
    y_pred = tf.tile(y_pred, [1, 1, d, 1])
    loss_mat = f_loss(y_true, y_pred)
    output = f_accuracy(y_true, y_pred)
    iou, p = output[0], output[1]
    iou_ = tf.zeros((), dtype=tf.float32)
    p_ = tf.zeros((), dtype=tf.float32)
    for i in tf.range(tf.shape(loss_mat)[0]):
        selectors = tf.numpy_function(linear_sum_assignment,
        [loss_mat[i]], [tf.int64, tf.int64] )
        row_ind = tf.cast(selectors[0], dtype=tf.int32)
        col_ind = tf.cast(selectors[1], dtype=tf.int32)
        for j in tf.range(0, tf.shape(row_ind)[0]):
            iou_ = tf.add(iou_, iou[i, row_ind[j], col_ind[j]])
            p_ = tf.add(p_, p[i, row_ind[j],col_ind[j]]) )[0],
            dtype=tf.float32))

    iou_ = tf.divide(iou_, num_object) if num_object > 0 //
        else tf.cast(1, dtype=tf.float32)
    p_ = tf.divide(p_, tf.reduce_prod(shapes))
    return p_
```

Далее представлена програмная реализация метрики IoU для сети ViT на языке программирования Python

```
def Hang_IoU(y_true, y_pred):
    d = tf.shape(y_true)[1]
    obj_true = tf.cast(y_true[...,-1], dtype=tf.float32)
    shapes = tf.cast(tf.shape(obj_true), dtype=tf.float32)
    num_object = tf.reduce_sum(obj_true)
    y_true = tf.expand_dims(y_true, axis=1)
    y_true = tf.tile(y_true, [1, d, 1, 1])
    y_pred = tf.expand_dims(y_pred, axis=2)
    y_pred = tf.tile(y_pred, [1, 1, d, 1])
    loss_mat = f_loss(y_true, y_pred)
    output = f_accuracy(y_true, y_pred)
    iou = output[0]
    iou_ = tf.zeros((), dtype=tf.float32)
    for i in tf.range(tf.shape(loss_mat)[0]):
        selectors = tf.numpy_function(linear_sum_assignment,
        [loss_mat[i]], [tf.int64, tf.int64] )
        row_ind = tf.cast(selectors[0], dtype=tf.int32)
        col_ind = tf.cast(selectors[1], dtype=tf.int32)
        for j in tf.range(0, tf.shape(row_ind)[0]):
            iou_ = tf.add(iou_, iou[i, row_ind[j], col_ind[j]])
    if num_object == 0:
        iou_ = tf.cast(1, dtype=tf.float32)
    else:
        iou_ = tf.divide(iou_, num_object)

    return iou_
```