

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Физико-механический институт

Работа допущена к защите

Руководитель ОП

К.Н. Козлов

«_____» _____ 2023 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
РАСПОЗНАВАНИЕ ХИМИЧЕСКОЙ СТРУКТУРЫ МОЛЕКУЛЫ
МЕТОДАМИ МАШИННОГО ОБУЧЕНИЯ**

по направлению подготовки 01.04.02 Прикладная математика и информатика
Направленность (профиль) 01.04.02_02 Математические методы анализа и визуализации данных

Выполнил

студент гр. 5040102/10201

К.А. Дамасинский

Руководитель

старший преподаватель ВШПМиВФ

В.С. Чуканов

Консультант

по нормоконтролю

Л.А. Арефьева

Санкт-Петербург

2023

**САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ПЕТРА ВЕЛИКОГО
Физико-механический институт**

УТВЕРЖДАЮ
Руководитель ОП
_____ К.Н. Козлов
«_____» _____ 2023г.

**ЗАДАНИЕ
на выполнение выпускной квалификационной работы**

студенту Дамаскинскому Константину Александровичу гр. 5040102/10201

1. Тема работы: Распознавание химической структуры молекулы методами машинного обучения.
2. Срок сдачи студентом законченной работы июнь 2023.
3. Исходные данные по работе
 - Набор молекул, сохранённых в InChI формате, полученный из kaggle соревнования BMS [3.1].
 - Обзорные статьи:
 1. OCR-решения для распознавания молекул: [3.2]
 2. ML-решения для распознавания молекул: [3.2]
 - Инstrumentальные средства:
 - Языки программирования C++, Python, bash
 - Платформа для машинного обучения Google Cloud
 - Система контроля версий git
- 3.1. Kaggle соревнование BMS. — URL: <https://www.kaggle.com/c/bms-molecular-translation> (дата обращения: 28.05.2023).
- 3.2. *Vogt M.* Using deep neural networks to explore chemical space // Expert Opinion on Drug Discovery. — 2022. — Т. 17, № 3. — С. 297—304.
4. Содержание работы (перечень подлежащих разработке вопросов):
 - 4.1. Введение. Обоснование актуальности
 - 4.2. Постановка задачи

- 4.3. Обзор существующих решений
 - 4.4. Разработка моделей машинного обучения
 - 4.5. Применение
 - 4.6. Результаты и их сравнительный анализ
 - 4.7. Выводы
 - 4.8. Заключение
5. Дата выдачи задания: 13.02.2023.

Руководитель ВКР _____ В.С. Чуканов

Задание принял к исполнению 13.02.2023

Студент _____ К.А. Дамасинский

РЕФЕРАТ

На 33 с., 17 рисунков, 2 таблицы, 1 приложение

КЛЮЧЕВЫЕ СЛОВА: ЗАДАЧА РАСПОЗНАВАНИЯ, ХИМИЯ, INCHI, МАШИННОЕ ОБУЧЕНИЕ.

Тема выпускной квалификационной работы: «Распознавание химической структуры молекулы методами машинного обучения»¹.

Целью данной работы является построение программного продукта, который позволяет распознавать структуру химических, в том числе стереоорганических, молекул. Основным требованием является способность распознавать изображения, полученные из сканов печатной литературы и электронных документов.

При выполнении работы произведён анализ программных продуктов, выполняющих задачу распознавания структуры молекулы, основанных на принципах оптического распознавания знаков (алгоритмический подход) и машинного обучения. Разработано решение в парадигме машинного обучения: построен конвейер из трёх моделей машинного обучения с архитектурами Faster R-CNN, CNN и CNN. В открытых источниках найден набор InChI-последовательностей для обучения и тестирования моделей. Реализован генератор изображений молекул с различными аугментациями, такими как изменение разрешения изображения, толщина и длина линий связей, шрифты меток атомов, гауссовский шум на изображении. Построен алгоритм, осуществляющий сборку молекулы из распознанных компонентов и валидацию построенного химического соединения. Оценено качество для всех этапов полученного решения. Произведено сравнение качества с существующими решениями. Выполнены замеры производительности программного продукта.

Работа реализована на языке программирования Python. Обучение производилось на платформе Google Cloud и на персональном ноутбуке с видеокартой.

ABSTRACT

33 pages, 17 figures, 2 tables, 1 appendices

KEYWORDS: OBJECT DETECTION, CHEMISTRY, INCHI, MACHINE LEARNING.

¹Реферат должен содержать: предмет, тему, цель ВКР; метод или методологию проведения ВКР; результаты ВКР: область применения результатов ВКР; выводы.

The subject of the graduate qualification work is «Molecular structure detection by machine learning methods».

This study aims at building a software product capable of recognizing the structure of chemical molecules, stereoorganic ones in particular. The primary feature of this product is the ability to recognize images from photocopies of print materials and electronic documents.

The study involved the analysis of the software products designed to recognize the structure of a molecule, that are based on optical character recognition (algorithmic approach) and machine learning. The solution was developed in the machine learning paradigm, and it included building a pipeline of three machine learning models with the Faster R-CNN, CNN and CNN architectures. A set of InChI sequences used for training and testing models was found in open source. A generator was developed for producing molecule images with various augmentations, such as changing image resolution, varying thickness and length of lines representing the bonds between atoms, atomic label fonts, Gaussian noise. An algorithm was designed to assemble the correctly recognized chemical components into one molecule and then validate the resulting chemical compound. The quality of image recognition and molecule assembly was assessed and compared with the existing solutions. The software product performance was measured and evaluated.

The software code is written in the Python programming language. The models were trained on the Google Cloud platform and on personal computer with a GPU.

СОДЕРЖАНИЕ

Введение	7
Глава 1. Постановка задачи распознавания молекулы. Методы решения....	10
1.1. Постановка задачи распознавания молекулы.....	10
1.1.1. Описание входных данных.....	10
1.1.2. Описание выходных данных	11
1.2. Rule-based решения.....	11
1.3. ML-based решения	12
1.4. Решение на базе нейросети-трансформера	13
1.5. Алгоритм сборки молекулы	15
Глава 2. Построение конвейера машинного обучения. Генерация данных ..	17
2.1. Object detection задача. Модель Detectron2 Faster R-CNN	17
2.1.1. Архитектура Detectron2 Faster R-CNN ResNet 50 FPN	17
2.2. Первоначальный конвейер моделей машинного обучения	22
2.2.1. Описание конвейера.....	22
2.2.2. Подготовка данных	22
2.2.3. Результаты обучения.....	23
2.3. Итоговый конвейер моделей машинного обучения	25
2.3.1. Изменения конвейера	25
2.3.2. Результаты обучения итоговой object detection модели	25
2.3.3. Архитектура вспомогательных нейронных сетей. Генерация данных	28
2.3.4. Результаты обучения классификаторов	28
Глава 3. Результаты работы конвейера моделей.....	29
3.1. Качество сборщика молекул.....	29
3.2. Качество конвейера моделей	30
3.3. Типовые ошибки конвейера. Методы устранения.....	30
3.4. Производительность решения	31
Заключение	32
Список использованных источников.....	33
Приложение 1. Архитектура Faster R-CNN.....	34

ВВЕДЕНИЕ

Задача распознавания структуры химической молекулы является разновидностью задачи компьютерного зрения. До того, как машинное обучение получило широкое распространение, такие задачи чаще всего решали алгоритмическим путём. Так, большинство решений, упомянутых в [12], состояло из этапов предобработки (сглаживание, бинаризация), поиска связей как отрезков прямых и распознавания меток атомов OCR-методами. Решение, основанное на таком подходе, оказывается уязвимым к дефектам изображения, таким как шум, получаемый при сканировании, дефекты печати, следы эксплуатации журналов и книг (мятые или надорванные страницы). Кроме того, при создании алгоритмического решения оказывается гораздо сложнее поддерживать такие параметры, как положения букв на метках атомов, шрифты, расстояние между линиями на двойных и тройных связях, поскольку OCR-методы зачастую параметризуются различного рода порогами погрешностей.

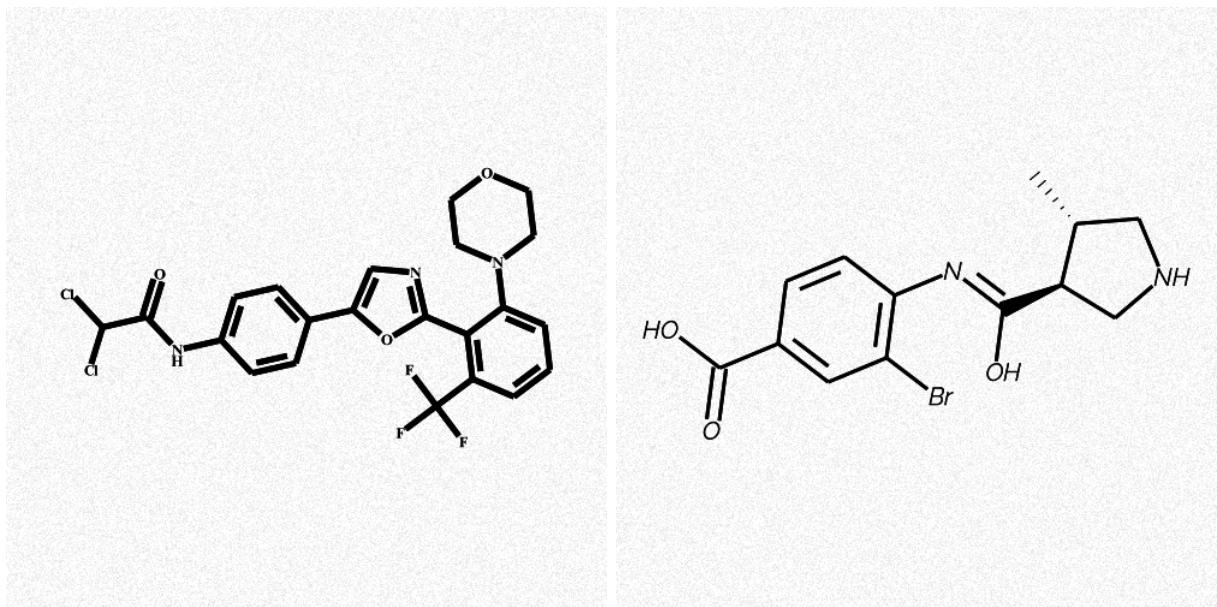


Рис.0.1. Изображения молекул, на которых метка атома азота с присоединённым атомом водорода изображена: (1) вертикально, (2) горизонтально

Таким образом, программная поддержка алгоритмических решений является крайне трудоёмким процессом, поскольку при появлении новых, пусть даже слабо отличающихся от предыдущих, стандартов рендеринга изображений, в программу-распознаватель придётся дописывать блоки кода, поддерживающие эти изменения.

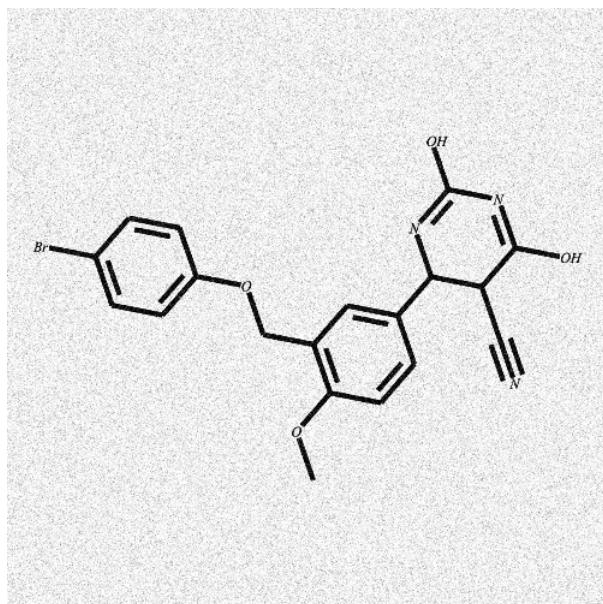


Рис.0.2. Молекула с двойными и тройными связями. Двойная связь обозначается двумя параллельными отрезками, тройная – тремя

В то же время решение, построенное с помощью методов машинного обучения, даже не видя каких-то характерных особенностей при обучении, зачастую сможет провести корректное распознавание. Если же окажется, что изменение слишком сильное, достаточно добавить некоторое количество изображений с новой особенностью в обучающую выборку и дообучить сеть, что требует весьма небольших затрат ресурсов.

Задача распознавания структуры химической молекулы является частью комплексного решения по распознаванию документа и необходима преимущественно для распознавания научных статей по химии, биологии и медицине. Распознавание документов необходимо для следующих целей:

- Хранение «идеальных» рендеров: пользователь получает возможность читать документы без дефектов. Изображения молекул и другие векторные по своей природе схемы, отпечатанные на бумаге в растром виде, сохраняются в векторном формате, что позволяет пользователю масштабировать изображение до необходимых ему пределов
- Хранение документов в plain-text формате позволяет осуществлять эффективный поиск информации. В частности, модуль распознавания структуры молекулы позволяет искать научные статьи, содержащие информацию о конкретной молекуле. Данная функциональность помогает фармацевтическим исследователям находить интересующие их свойства конкретных веществ, что значительно ускоряет процесс создания лекарственных средств

В рамках данной работы предполагается, что распознавание структуры химической молекулы может производиться как для сканов печатных документов, так и для электронных документов, не содержащих информацию о структуре молекулы в плоском виде.

Таким образом, **исследование является актуальным**, поскольку построение решения, которое позволяет распознавать структуру химической молекулы на разнообразных, поможет создать высококачественный распознаватель документов химической тематики, что в свою очередь значительно улучшит качество и производительность работы исследователей в сфере химии, биологии и фармацевтики.

Целью проводимого исследования является создание описанного ранее решения, замеры его качества, оценка производительности. Также в рамках исследования будет оценена скорость обучения выбранных моделей и *масштабируемость* решения, то есть трудозатраты, требуемые для поддержки дополнительных элементов молекул, таких как RS-стереометки атомов и обозначения смесей различных геометрических изомеров.



Рис.0.3. Изображение молекул с (E), (Z) метками [10]

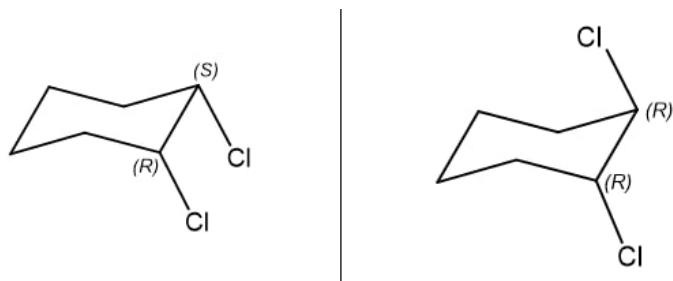


Рис.0.4. Изображение молекул с (R), (S) метками [10]

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ РАСПОЗНАВАНИЯ МОЛЕКУЛЫ. МЕТОДЫ РЕШЕНИЯ

В данной главе приводится постановка задачи распознавания молекулы. Даются определения основных понятий предметной области и описываются существующие решения.

В параграфе 1.1 приводится постановка задачи распознавания молекулы. Производится описание входных и выходных данных. В параграфе 1.2 описана общая структура программ, построенных на алгоритмах распознавания геометрических примитивов. Такие программы мы будем называть **rule-based решениями**. В параграфе 1.3 описаны различные подходы решения задачи распознавания молекулы с использованием методов машинного обучения – **ML-based системы**. В параграфе 1.5 приведён алгоритм сборки молекулы из распознанных тех или иным способом примитивов.

1.1. Постановка задачи распознавания молекулы

1.1.1. Описание входных данных

На вход задачи подаётся изображение химической молекулы. В рамках данной задачи конкретизируются следующие требования к изображениям:

- Молекула является органической. Это означает, что в молекуле присутствуют только те атомы, которые составляют основу либо присутствуют в органических соединениях: углерод (C), кислород (O), азот (N), водород (H), фосфор (P), фтор (F), кремний (Si), а также хлор (Cl) и бром (Br)
- Молекула может быть стереоизомером: на изображении допускаются сплошные и штрихованные клиновидные связи
- На изображении отрендерена молекула, являющаяся конкретным изомером, то есть не поддерживаются смеси вида (*R*), (*S*), *RS*, (*RS*). Также не поддерживаются перечёркнутые связи, обозначающие смесь геометрических (*E* и *Z*) изомеров

Изображения могут быть получены как из электронных источников литературы, так и из печатных. Допускается наличие дефектов, связанных с небрежным хранением бумажного документа, его возрастом, а также гауссовское зашумление, получаемое в результате процесса сканирования.

Не допускается наличие искажений изображения, получаемых:

- при фотографировании на длинной выдержке
- при неполном прилегании документа к стеклу сканера

1.1.2. Описание выходных данных

Выходные данные представляют из себя стандартизованный текстовый идентификатор InChI [3; 4]. Данный идентификатор разработан организациями IUPAC и NIST в течение 2000 – 2005 годов для обеспечения стандартного и читаемого способа кодирования информации о структуре молекулы и облегчить поиск этой информации в различных базах данных. Он позволяет хранить информацию о структурной формуле, стереоизомерии и наличии изотопов.

Важно отметить, что, тем не менее, InChI не всегда способен однозначно закодировать скелетную формулу. Так, при рисовании алленов зачастую вместо одной из двух клиновидных связей рисуют простую одинарную связь, подразумевая при этом клиновидную. Химики так делают, поскольку знают, что молекула со скелетом такого вида может иметь только одну конкретную геометрическую структуру, в связи с чем достаточно нарисовать только один клин. При этом InChI-представления у этих двух скелетных формул будут разными, хотя молекулы являются одинаковыми. Это явление значительно снижает качество распознавания молекул и будет более подробно показано в подразделе 1.5.

1.2. Rule-based решения

Для подготовки обзора rule-based решений была использована статья [12]. В ней описаны 14 различных программных продуктов, реализованных в рамках rule-based подхода. Каждый продукт имеет свои особенности, которые позволяют ему справляться лучше с теми или иными особенностями изображения, однако основной конвейер у них общий и состоит из следующих этапов:

- 1. Предобработка.** Данный этап предполагает бинаризацию изображения, то есть его перевод в оттенки серого, сглаживание изображения – использование антиалиасинга, фильтров Гаусса и т. п., и утончение линий.
- 2. Поиск стандартных и клиновидных связей.** Данная операция предполагает обнаружение и классификацию связей. Стандартные связи являются совокупностью от одного до трёх параллельных отрезков, поэтому они

могут быть обнаружены с применением детектора Хаффа. Штрихованные клиновидные связи могут быть обнаружены как последовательность увеличивающихся в толщине и ориентированных одинаковым образом равнобедренных трапеций. Сплошные клиновидные связи изображаются как равнобедренный треугольник, основание которого много меньше, чем две его остальные стороны.

3. Поиск ароматических соединений. Ароматическое соединение – это циклические углеводороды, обладающие определёнными химическим свойствами, такими как склонность к реакциям замещения. С точки зрения задачи распознавания важно понимать, что ароматическое соединение представляет может быть изображено в виде формулы Кекуле или формулы Полинга – таким образом, возникает необходимость поиска окружностей внутри циклов и правильного добавления этих соединений в молекулу.

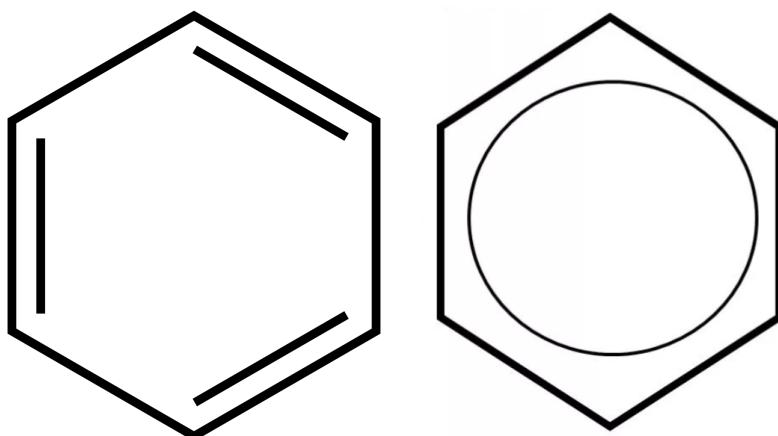


Рис.1.1. Бензол. (1) Формула Кекуле, (2) Формула Полинга

1.3. ML-based решения

Согласно [12], до 2019 года было известно два решения, использовавших машинное обучение. Первое из них, OCSR [13], использовало карты Кохонена для того, чтобы отличать изображения химических молекул от других картиночек. Программный продукт ChemOCR использовал машину опорных векторов ([14]) для классификации изображений связей.

Первый продукт, полностью построенный на технологиях машинного обучения, называется MSE-DUDL и представлен в 2019 году [15]. Известно, что в основе

этого продукта лежит свёрточная сеть на базе U-Net. Модель была натренирована на 57 миллионах изображений, сгенерированных пакетом Indigo.

Второй продукт, Chemgrapher, содержит две сети. Одна отвечает за обнаружение объектов, а вторая – за их классификацию. Обе сети являются свёрточными, и были натренированы на изображениях, сгенерированных пакетом RDKit. В результате исследований этого решения было выяснено, что сегментирующая нейронная сеть имеет гораздо более низкое качество, чем классифицирующая.

1.4. Решение на базе нейросети-трансформера

Стоит отдельно упомянуть решения, построенные на нейросетях с архитектурой «трансформер».

Отличием трансформера от классических CNN и RNN является наличие механизма внимания Multi-Head Attention.

Данная архитектура сетей чаще всего применяется при решении задачи машинного перевода, поэтому описывать принцип её работы мы здесь будем в терминах текстов. Тем не менее, как и любая нейронная сеть, трансформер может принимать на вход любые конечные последовательности данных, в частности изображения молекул.

В данной архитектуре все слова параллельно проходят через заданные слои. На пути каждого слова находится слой внимания, суть которого заключается в возможности взаимодействия с другими словами.

На вход слою внимания даются вектора Query и несколько пар Key и Value. Чаще всего при практическом использовании оказывается, что Key и Value это один и тот же вектор. Каждый из них преобразуется обучаемым линейным преобразованием, а потом вычисляется скалярное произведение Q со всеми K по очереди, результат этих скалярных произведений подаётся на вход в softmax слой, и с полученными весами все вектора V суммируются в единый вектор.

Данный подход позволяет обучать сеть искать похожие в каком-то смысле слова и таким образом «уделять внимание» наиболее значимым словами при произведении машинного перевода.

Несмотря на то, что трансформеры изначально задумывались как нейросети для машинного перевода, они нашли также применение в других задачах. Например, можно обучить трансформер таким образом, чтобы он на вход получал изображение,

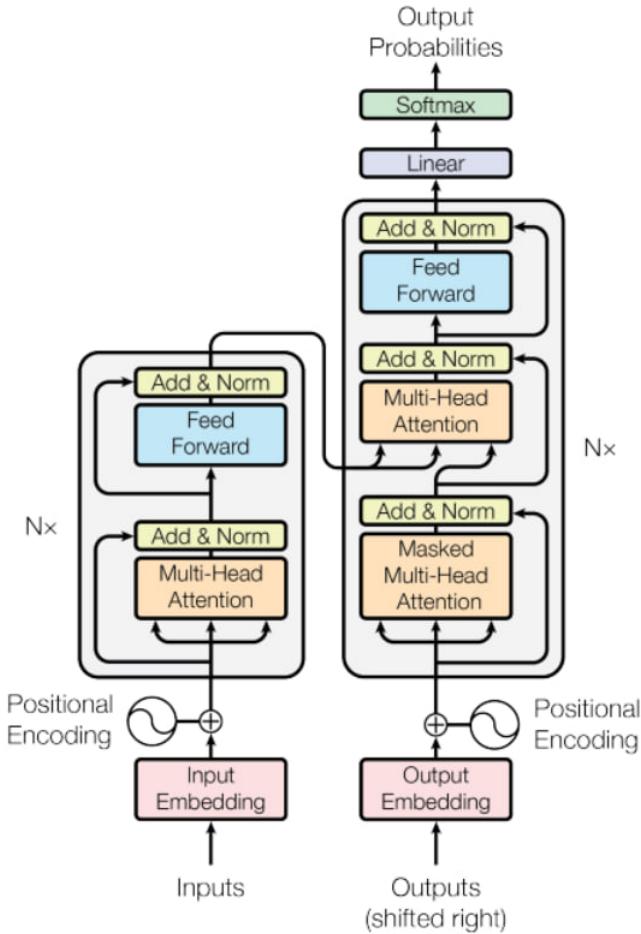


Рис.1.2. Архитектура сети «трансформер»

а на выход отдавал текстовую последовательность. Одним из примеров таких сетей является сеть DETR от FacebookResearch [2].

На базе трансформера в рамках kaggle-соревнования BMS [5] было разработано решение, которое конвертирует изображение молекулы *целиком* в InChI-последовательность. Оказалось, что трансформер решает задачу распознавания структуры молекулы даже лучше, чем object-detection модель, из чего можно сделать вывод, что механизм внимания позволяет сети обучаться искать взаимосвязь между скелетной формулой и InChI-цепочками. Данное наблюдение ещё раз показывает способность трансформеров к обучению языкам, поскольку InChI-последовательность, как и скелетная формаула, является языком описания структуры молекулы.

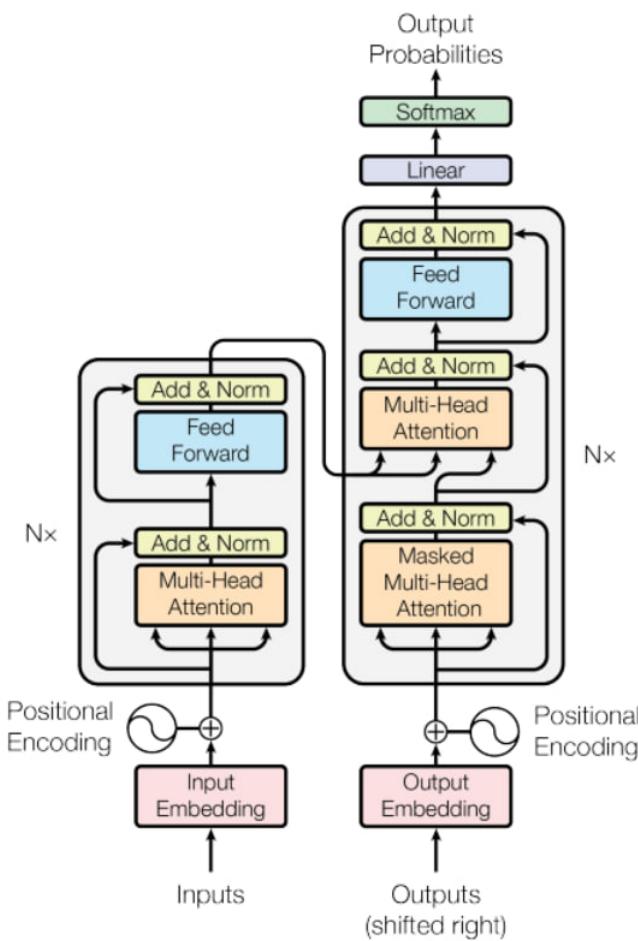


Рис.1.3. Архитектура слоя внимания

1.5. Алгоритм сборки молекулы

При решении задачи распознавания молекулы любым из описанных выше способов возникает необходимость собрать молекулу из отдельных примитивов. Приведём здесь алгоритм, который позволяет это делать.

На вход алгоритма подаётся набор изображений связей и меток атомов. Будем строить *граф молекулы* – граф, в вершинах которого находятся атомы (непомеченные атомы углерода либо другие атомы с указанием названия), а рёбрами которого являются связи.

Такой граф можно передать в один из химических пакетов. В рамках данной работы используется Python-реализация пакета RDKit. Данный пакет позволяет:

- Строить граф молекулы из таких распространённых форматов, как SMILES, SDF, InChI
- Добавлять атомы и связи в существующий граф молекулы, а также указывать категорию связи

- Генерировать текстовые идентификаторы SMILES, SDF, InChI из графа молекулы

Таким образом, для генерации InChI-идентификатора достаточно правильным образом сформировать граф молекулы.

Будем считать, что связи представлены в баундбоксах, для которых указана категория связи и её направление. В случае плоской связи достаточно указать направление диагонали, в случае стереосвязи – её начало и конец. Метки атомов также ожидаются в виде баундбоксов, для которых указан текст метки.

Граф строится в несколько этапов.

Первый этап – построение списка смежности молекулы. Для этого производится двойное вложенное итерирование по списку связей и, если у бсвязей находится общая точка (пересечение баундбоксов с учётом направления связи внутри баундбокса), то точка соединения объявляется новым атомом. Если атом уже существует в списке смежности, то текущая связь добавляется в него. Иначе, создаётся новый элемент в списке смежности и к нему присоединяется две связи.

Второй этап – поиск висячих вершин, то есть непомеченных атомов, которые связаны только с одним другим атомом.

Третий этап – сопоставление помеченных и непомеченных атомов. Для этого производится итерирование по всем возможным парам вершин графа молекулы и меток атомов и, если метка расположена достаточно близко к вершине, то считается, что метка относится к данной вершине.

Последний этап – добавление построенного списка смежности в rdkit-молекулу. После того, как атомы и связи добавлены, необходимо вызвать генератор центров хиральности. Он позволяет сгенерировать данные, необходимые для вывода InChI-идентификатор из информации о клиновидных связях. Важно заметить, что RDKit самостоятельно выводит количество присоединённых атомов водорода из того набора связей, который текущий атом имеет со своими соседями. Это обстоятельство позволяет при классификации меток атомов определять в одну категорию метки с явным указанием присоединённых атомов водорода и без него.

Неприятным моментом является тот факт, что на данном этапе происходят основные потери точности сборки, связанные с неоднозначным соответствием строения химической молекулы и необходимым набором клиновидных связей, о котором упоминалось во введении.

ГЛАВА 2. ПОСТРОЕНИЕ КОНВЕЙЕРА МАШИННОГО ОБУЧЕНИЯ. ГЕНЕРАЦИЯ ДАННЫХ

Данная глава посвящена исследованиям, проведённым для построения конвейера машинного обучения. В параграфе 2.1 приведено описание object detection задачи машинного обучения и выбранной для этого модели. В параграфе 2.2 описан первоначальный конвейер моделей, с помощью которого предполагалось решать задачу. Описан процесс генерации данных для используемой object detection модели, результаты обучения модели. В параграфе 2.3 предложены исправления, необходимые для повышения качества обучения модели и показаны результаты обучения object detection модели на итоговом варианте. Далее, приводится методика генерации данных для обучения вспомогательных сетей, решение об использовании которых было принято по результатам проведённых ранее экспериментов. Показаны результаты обучения вспомогательных моделей.

2.1. Object detection задача. Модель Detectron2 Faster R-CNN

Object detection задача состоит в необходимости найти область интереса – region of interest, ROI в виде баундбокса, то есть прямоугольника, стороны которого параллельны сторонам экрана, и который содержит объект интересов.

Для выполнения данной работы была использована модель Detectron2 Faster R-CNN ResNet 50 FPN [1].

2.1.1. Архитектура Detectron2 Faster R-CNN ResNet 50 FPN

Для того, чтобы детально описать архитектуру сети Detectron2 Faster R-CNN, необходимо сначала описать архитектуру сетей предыдущего поколения R-CNN и Fast-CNN.

2.1.1.1. Архитектура R-CNN

R-CNN, основоположник модели Faster R-CNN – один из первых подходов, применённых для определения нахождения объекта на картинке [6].

Его архитектура состоит из следующих шагов:

1. Определение набора гипотез.

2. Извлечение из предполагаемых регионов признаков с помощью сверточной нейронной сети и их кодирование в вектор.
3. Классификация объекта внутри гипотезы на основе вектора из шага 2.
4. Улучшение (корректировка) координат гипотезы.
5. Повтор шагов 2-4, пока не будут обработаны все гипотезы.

Опишем каждый этап подробнее.

- **Поиск гипотез.** Изображение на входе первым делом разбивается на маленькие гипотезы разных размеров. Для этого используется алгоритм Selective Search [9]. Таким образом выделяется примерно 2000 разных регионов, которые частично друг друга перекрывают. Для более точной последующей обработки каждая гипотеза дополнительно расширяется на 16 пикселей во всех 4 направлениях – как бы добавляя *контекст*.
- **Кодирование признаков в вектор.** Каждая гипотеза из предыдущего шага независимо и по отдельности друг от друга поступает на вход сверточной нейронной сети. В качестве ее используется архитектура AlexNet без последнего softmax-слоя. Главной задачей сети является кодирование поступающего изображения в векторное представление, которое извлекается из последнего полносвязного FC7 слоя. Так на выходе получается 4096-размерное векторное представление.
- **Классификация.** После получения характеризующего гипотезу вектора становится возможна ее дальнейшая обработка. Для определения, какой именно объект находится в предполагаемом регионе, авторы используют классический метод классификации разделяющими плоскостями на базе SVM. SVM работает по принципу OvR: One vs. Rest – один против всех, один из методов мультиклассовой классификации. Таким образом, фактически решается задача бинарной классификации – есть ли конкретный класс объекта внутри предполагаемого региона или нет. Таким образом, выходом является размерный вектор, отображающий уверенность в конкретном классе содержащегося в гипотезе объекта.
- **Уточнение гипотез.** Гипотезы, полученные на шаге 1, не всегда содержат правильные координаты, например, объект может быть неудачно «обрезан». Поэтому необходимо их дальнейшее уточнение. Так, гипотезы, содержащие какой-либо объект (наличие объекта определяется на шаге классификации), дополнительно обрабатываются линейной регрессией. То

есть гипотезы с классом «фон» не нуждаются в дополнительной обработке регионов, ведь на самом деле там нет объекта.

Каждый объект, специфично для своего класса, имеет определенные размеры и соотношения сторон, поэтому рекомендуется применять собственный регрессор на каждый класс.

В отличие от предыдущего шага, авторы для наилучшей работы используют на входе не вектор из слоя FC7, а карты признаков, извлеченные из последнего MaxPooling слоя (в AlexNet его размерность составляет $256 \times 6 \times 6$). Это объясняется тем, что вектор сохраняет информацию о наличии объекта с какими-то характеристическими подробностями, а карта признаков наилучшим образом сохраняет информацию о местоположении объектов.

2.1.1.2. Архитектура Fast R-CNN

Данная архитектура является улучшенным вариантом архитектуры R-CNN. Целью её создания было, в первую очередь, ускорение процесса обучения и инференса. Рассмотрим этапы, через которые проходит изображение при обработке Fast R-CNN [7].

- Извлечение карты признаков изображения (не для каждой гипотезы по отдельности, а для всего изображения целиком)
- Поиск гипотез – остался без изменения
- Сопоставление каждой гипотезы с местом на карте признаков, то есть использование единого набора выделенных признаков для каждой гипотезы (координаты гипотез можно однозначно сопоставить с местоположением на карте признаков)
- Классификация каждой гипотезы и исправление координат ограничивающей рамки (этот шаг стало возможным запускать параллельно, поскольку больше нет зависимости от SVM-классификации)

В изначальной концепции R-CNN каждая предложенная гипотеза по отдельности обрабатывается с помощью CNN – этот этап стал бутылочным горлышком для всей архитектуры. Для решения этой проблемы был разработан Region of Interest (RoI) слой. Этот слой позволяет единожды обрабатывать изображение целиком с помощью нейронной сети, получая на выходе карту признаков, которая далее используется для обработки каждой гипотезы.

Основной задачей RoI слоя является сопоставление координат гипотез (координаты баундбоксов) соответствующим координатам карты признаков. Делая «срез» карты признаков, RoI слой подает его на вход полно связному слою для последующего определения класса и поправок к координатам (см. следующие разделы).

В R-CNN использовались отдельные SVM-классификаторы. В Fast R-CNN они заменены одним SoftMax выходом. Отмечается, что при этом потеря точности составляет менее 1%.

Выход регрессоров обрабатывается с помощью NMS (Non-Maximum Suppression).

Multi-task loss. В одновременном обучении сети для задач регрессии баундбоксов и классификации применяется специальная лосс-функция:

$$L(p, u, t^u, v) = L_{cls}(P, u) + \lambda_{[u]} L_{loc}(t^u, v) \quad (2.1)$$

,

где:

- λ – настройка баланса между функциями потерь регрессора баундбоксов и классификатора
- u – правильный класс
- L_{cls} – функция потерь классификатора: $L_{cls}(P, u) = -\log P_u$
- L_{loc} измеряет разницу между целевым значением

необходим для настройки баланса между двумя функциями (авторы использовали $\lambda=1$); — правильный класс; представляет собой функции ошибки для классификации ; является *SmoothL1*-функцией и измеряет разницу между предсказаниями и целевыми значениями:

$$SmoothL1(x) = \begin{cases} \frac{1}{2}x^2, & |x| < 1 \\ |x| - \frac{1}{2}, & |x| \geq 1 \end{cases} \quad (2.2)$$

Здесь x обозначает разность целевого значения и предсказания $t_i^u - v_i$.

2.1.1.3. Архитектура Faster R-CNN

Следующим улучшением является изменения алгоритма поиска гипотез. Для этого представим всю систему как композицию двух модулей – определение

гипотез и их обработка. Первый модуль будет реализовываться с помощью **Region Proposal Network (RPN)**, а второй аналогично Fast R-CNN (начиная с RoI слоя).

Следовательно, в этот раз процесс работы с изображением изменился и теперь происходит таким образом:

1. Извлечение карты признаков изображения с помощью нейронной сети
2. Генерация на основе полученной карты признаков гипотез – определение приблизительных координат и наличие объекта любого класса
3. Сопоставление координат гипотез с помощью RoI с картой признаков, полученной на первом шаге
4. Классификация гипотез (уже на определение конкретного класса) и дополнительное уточнение координат

Основное улучшение произошло именно в месте генерации гипотез – теперь для этого есть отдельная небольшая нейронная сеть, которую назвали **Region Proposal Network**.

Конечной задачей данного модуля является полноценная замена Selective Search алгоритма. Для более быстрой работы необходимы общие веса с сетью, извлекающей необходимые признаки. Поэтому входом RPN является карта признаков, полученная после этой сети. Авторы оригинальной статьи используют для извлечения признаков сеть VGG16, выходом которой считают последний сверточный слой – *conv5_3*. Такая сеть имеет следующие характеристики рецептивного поля:

- Эффективное сжатие (effective strides,): 16
- Размер рецептивного поля (receptive field size,): 196

Это значит, что карта признаков будет в 16 раз меньше изначального размера изображения (количество каналов равно 512), а на каждое значение в ее ячейках оказывают влияние пиксели изначального изображения, лежащие в прямоугольнике размером 196×196 . Таким образом выходит, что если использовать стандартный вход VGG16 224×224 , то на формирование значения центральной ячейки карты признаков $(14, 14)$ будет влиять почти все изображение целиком. На основе полученной карты признаков RPN для каждой ячейки выдает гипотез (в изначальной реализации) разного размера и соотношения сторон. Так, для стандартного размера это $14 \cdot 14 \cdot 9 = 1764$ гипотезы!

Полную архитектуру сети Faster R-CNN с описанием слоёв можно найти в Приложении 1 1.

2.2. Первоначальный конвейер моделей машинного обучения

2.2.1. Описание конвейера

Первоначально планировалось построить конвейер из двух моделей машинного обучения:

1. Faster R-CNN для поиска и классификации связей и поиска меток атомов (без распознавания текста меток). Предполагалось использовать следующий набор категорий:
 - Порядок связи: 1-3
 - Для связи первого порядка – категория: плоская, клиновидная сплошная или клиновидная штрихованная
 - Направление связи: одна из диагоналей для стандартной плоской связи, одно из четырёх возможных направлений для клиновидных стереосвязей
 - Метка атома
2. Свёрточная нейронная сеть для классификации текста меток атомов. Поскольку в условиях задачи сказано, что производится распознавание только стереоорганических молекул, количество возможных меток весьма сильно ограничено. Во всём рассматриваемом датасете присутствует 45 различных меток, из них 16 составляют 97% всего датасета.

2.2.2. Подготовка данных

Для генерации данных используется открытый датасет – список InChI-идентификаторов из kaggle-соревнования «BMS» [5].

С помощью пакета RDKit [8] генерируются изображения молекул в формате png размером 500x500 пикселей. Применяются следующие аугментации:

- Изменение длины линии связи
- Изменение толщины линии связи
- Применение разных типов шрифтов, их жирности, курсива и размера
- Повороты молекул (при сохранении ориентации меток атомов)
- Зашумление изображения гауссовским шумом

При подготовке данных необходимо произвести **кекулезацию**. Смысл процедуры состоит в том, чтобы заставить RDKit определять категории связей, являющихся частью ароматической системы, как одинарные и двойные, а не аро-

матические, как это происходит по умолчанию. Категория ароматической связи присваивается по умолчанию, поскольку RDKit, исходя из InChI-представления, определяет набор связей как ароматическое соединение. Название процедуры связано с тем, что молекула, записанная в виде формулы Кекуле, выглядит как обычная молекула с одинарными и двойными связями, а в формуле Полинга всем связям в рамках ароматического соединения присваивается одна и та же категория и они в этом смысле неразличимы между собой.

2.2.3. Результаты обучения

2.2.3.1. Метрика AP

Для анализа качества обучения object detection модели используется метрика AP. Данная метрика показывает, насколько точно модель предсказала баундбокс и одновременно с этим угадала категорию объекта интереса.

Метрика вычисляется следующим образом. Если IoU распознанного i -го баундбокса больше некоторого значения x , строятся графики *precision*(*confidence score*) и *recall*(*confidence score*). Из этих двух параметрически заданных графиков получается график *precision*(*recall*). Тогда положим:

$$AP_i[@IoU = x] = \int_0^1 precision(recall) drecall \quad (2.3)$$

$$AP[@IoU = x] = \frac{1}{n} \sum_{i=1}^n AP_i[@IoU = x] \quad (2.4)$$

2.2.3.2. Графики качества и их анализ

Обучение производилось на 200000 изображениях. Соотношение размеров обучающей и валидационной выборки – 80/20. Было произведено 140000 итераций. На одну итерацию приходится обработка 1536 ROI. На одном изображении находится в среднем около 50 ROI, что означает, что была произведена 21 эпоха обучения.

В результате обучения был получен следующий график AP50 для валидационной выборки:

После того, как был получен такой результат, автор в ручном режиме просмотрел большое количество предсказанных данных и заметил, что модель

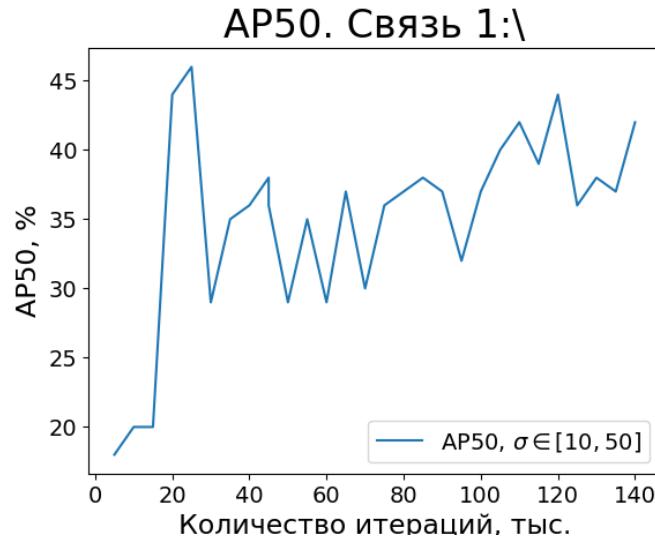


Рис.2.1. AP50. Первоначальный вариант обучения Faster R-CNN. Валидационная выборка. Здесь σ – параметр рассеяния гауссовского шума. При генерации датасета изображение зашумливалось в 2 из 3 случаях и, если зашумливалось, то параметр положения μ гауссовского шума присваивался нулю, а параметр рассеяния σ выбирался случайным образом равномерно из диапазона [10; 50]

практически всегда очень точно определяет границы баундбоксов, но часто путает категории. Отсюда такие низкие показатели метрики AP.

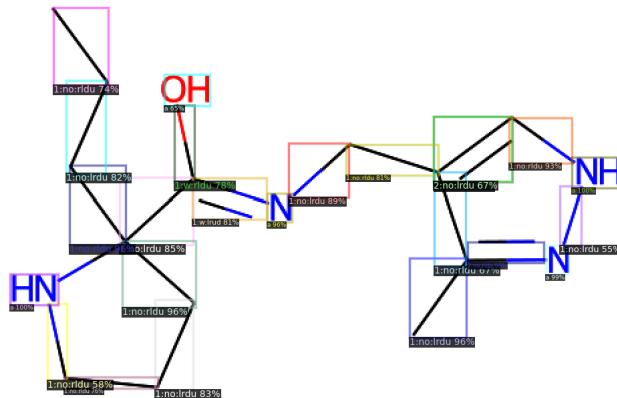


Рис.2.2. Пример результата предсказания. Видно, что двойная связь помечена категорией $I:w:lrud$, что означает сплошную клиновидную связь

2.3. Итоговый конвейер моделей машинного обучения

2.3.1. Изменения конвейера

По итогам обучения первоначального варианта было замечено, что модель особенно плохо справляется с определением направления связи внутри баундбокса, в связи с чем было принято свести категориальный набор к следующему:

- Порядок связи: 1-3
- Метка атома

Ожидается, что уменьшение количества категорий снизит нагрузку на модуль классификации, и тем самым получится повысить качество работы object detection модели.

Для классификации направления связи используется отдельная свёрточная сеть, архитектура которой описана в разделе 2.3.3.

2.3.2. Результаты обучения итоговой object detection модели

Приведём подробные графики всех loss функций.

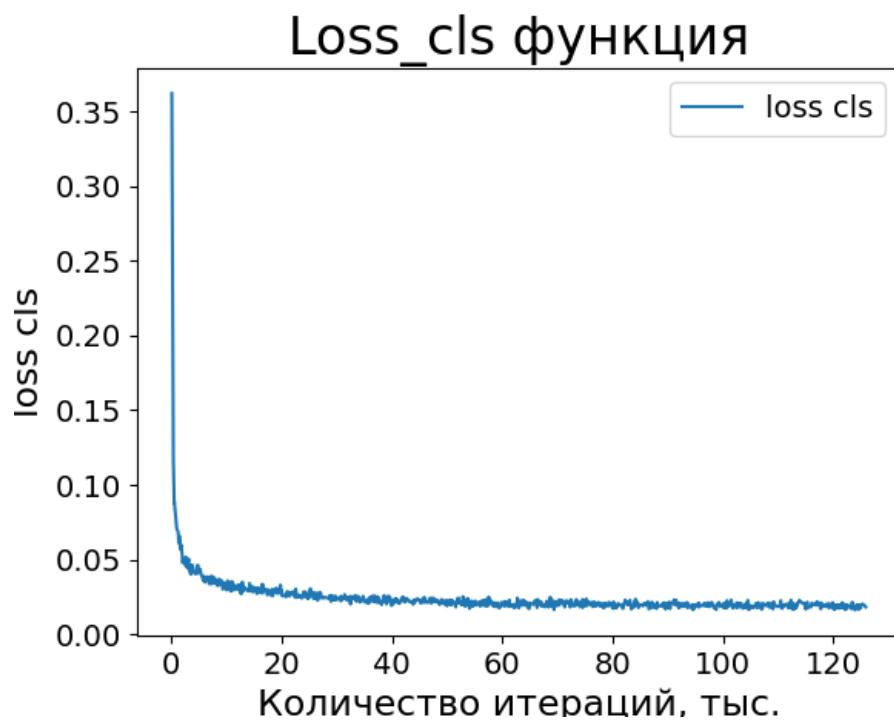


Рис.2.3. График функции loss_cls

Обучение производилось до тех пор, пока все loss-функции не выйдут на асимптоту, поскольку это является критерием окончания обучения модели.

Сопоставим графикам loss-функций графики валидационных метрик AP.

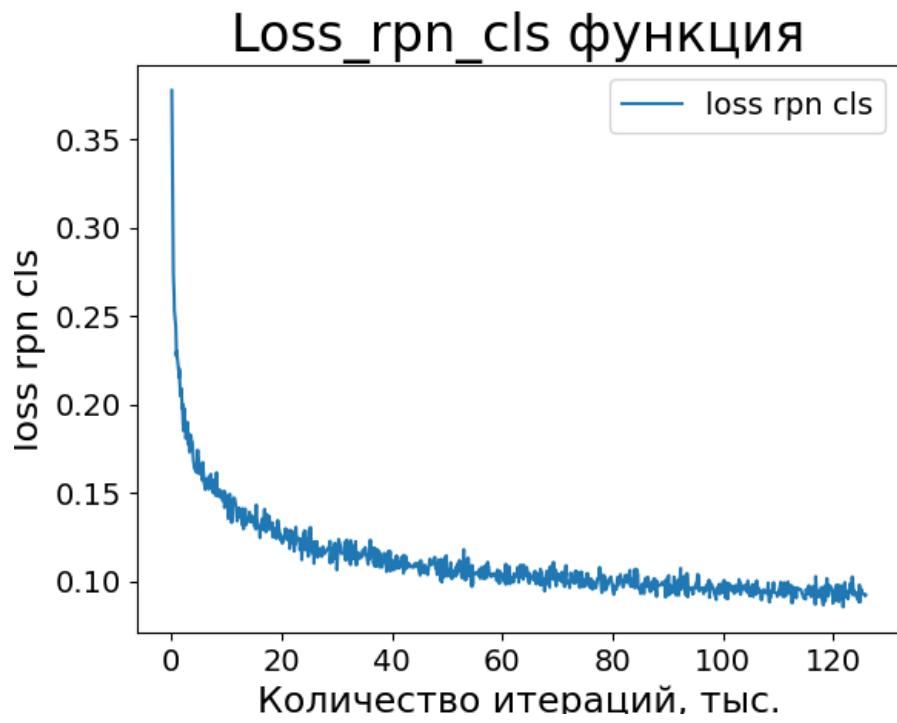


Рис.2.4. График функции loss_rpn_cls

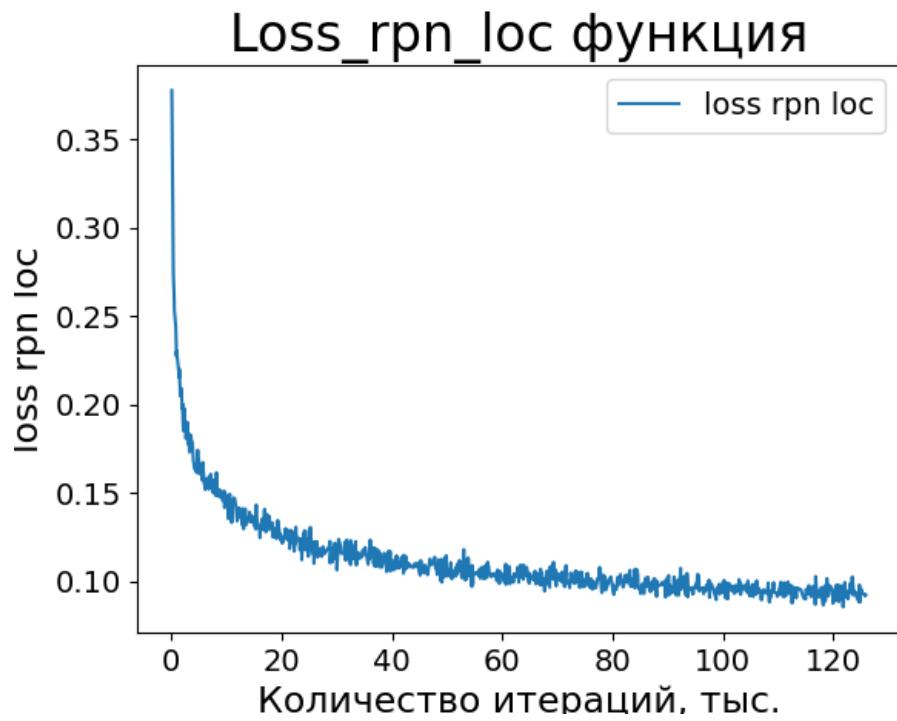


Рис.2.5. График функции loss_rpn_loc

На изображении 2.7 приведено два графика AP50 для валидационной выборки. Графики соответствуют разной степени зашумленности изображения. Видно, что, во-первых, качество приблизилось к идеальному: метрика AP50 превосходит 95% в обоих, а, во-вторых, сеть устойчива к шуму: повышение амплитуды шума в два раза снизило качество обучения не более, чем на 1.5 процентных пункта.

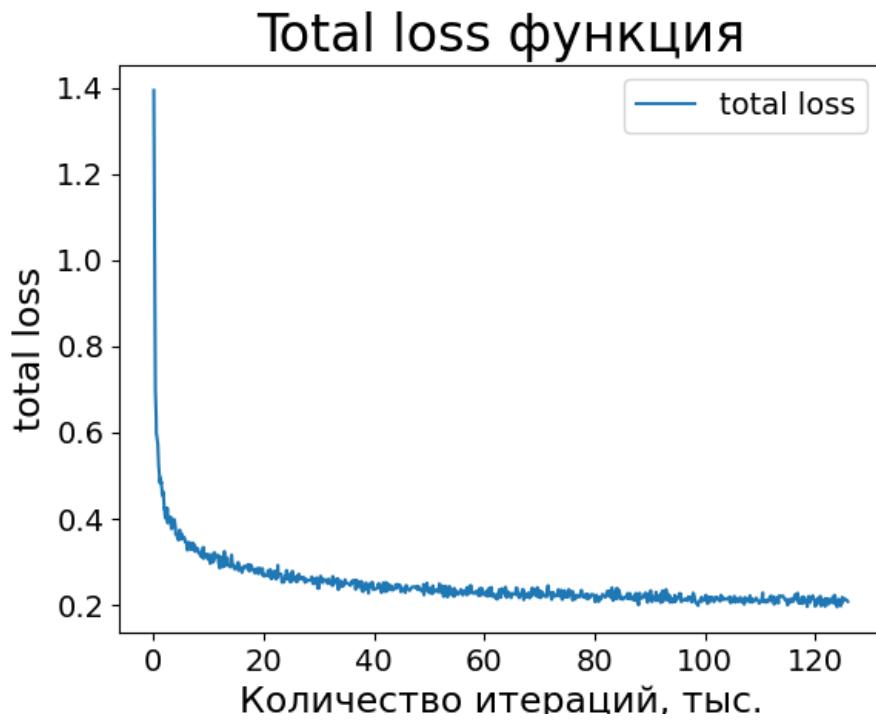
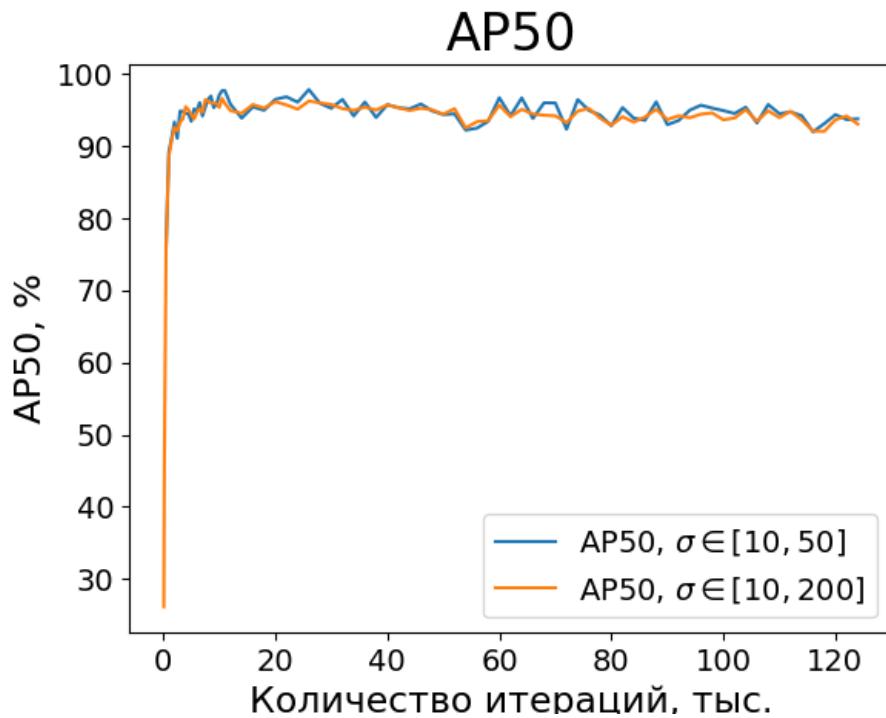


Рис.2.6. График функции total_loss

Графики функций потерь приведены для $\sigma \in [10; 200]$.

Рис.2.7. AP50. Итоговый вариант обучения Faster R-CNN. Валидационная выборка. Приведено два графика для разных степеней зашумленности изображения: $\sigma \in [10; 50]$, $\sigma \in [10; 200]$

Стоит обратить внимание на то, что несмотря на достаточно медленную сходимость сети (уровень асимптоты по всем функциям потерь достигается

примерно к 80-100 тысячам итераций), метрики качества выходят на асимптоту гораздо раньше.

2.3.3. Архитектура вспомогательных нейронных сетей. Генерация данных

Для классификации направлений связи и меток атомов были обучены две свёрточные нейронные сети с одинаковой архитектурой: использовались два свёрточных слоя, flatten, dropout и dense слои.

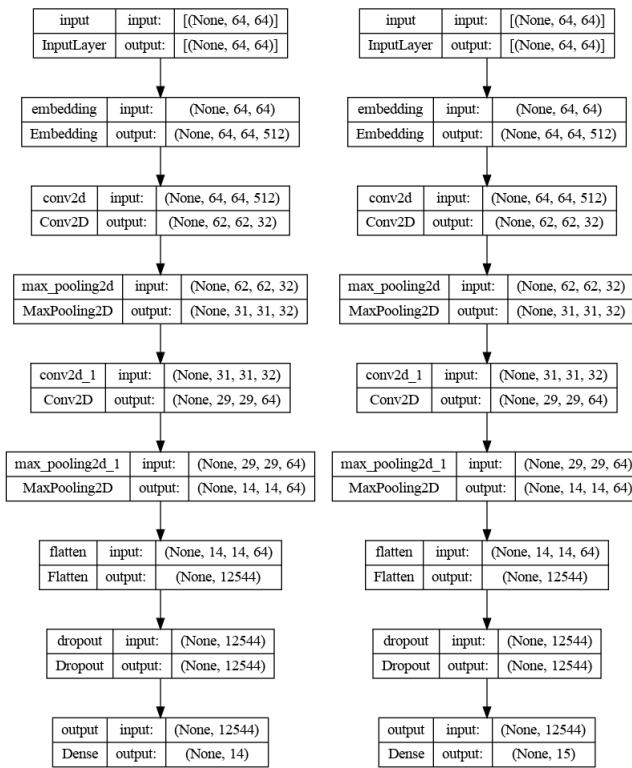


Рис.2.8. Архитектура CNN для распознавания связей (слева) и меток атомов (справа)

Для генерации обучающих данных использовались ранее сгенерированные для обучения object detection модели изображения. Был сформирован идеально сбалансированный датасет: на каждую категорию сгенерировано по 1000 изображений.

2.3.4. Результаты обучения классификаторов

Обучение производилось в обоих случаях на 50 эпохах.

Для классификатора категорий связи были получены следующие данные:

Для классификатора меток атомов были получены следующие данные:

$F_{1\min}$	$F_{1\max}$	$F_{1\text{avg}}$
0.97	0.99	0.98

Таблица 2.1

Качество обучения классификатора связей

$F_{1\min}$	$F_{1\max}$	$F_{1\text{avg}}$
0.96	0.98	0.9

Таблица 2.2

Качество обучения классификатора меток атомов

ГЛАВА 3. РЕЗУЛЬТАТЫ РАБОТЫ КОНВЕЙЕРА МОДЕЛЕЙ

В данной главе рассматриваются результаты работы конвейера моделей и сборщика графа молекул в совокупности. В параграфе 3.1 приведено качество работы сборщика молекул. В параграфе 3.2 приведены следующие метрики: среднее расстояние Левенштейна и процент точных совпадений InChI-идентификаторов. В параграфе 3.3 показаны характерные примеры, на которых конвейер ошибается, и предложены меры по исправлению некорректных распознаваний. В параграфе 3.4 приведены замеры производительности построенного продукта.

3.1. Качество сборщика молекул

Сборщик молекул тестировался следующим образом. Из InChI-идентификатора генерировался идеальный вывод конвейера нейросетей. Это делалось с помощью функциональности RDKit, позволяющей генерировать векторные изображения молекул. Таким образом, удалось установить точные границы ожидаемых баундбоксов и растеризовать их вместе с самими связями и метками атомов.

Далее полученные данные были переданы на вход сборщика. На выходе был получен InChI-идентификатор, который, в соответствии с ожиданиями, должен точно совпадать со входным идентификатором.

В результате оказалось, что генератор позволяет получить только 92.8% точных совпадений. При анализе проблемы было выяснено, что 0.8% несовпадений вызвано некачественной генерацией изображений, когда изображения связей и меток накладываются друг на друга, а остальные 6.4% вызваны неоднозначностью изображения связи и её метаданных либо неоднозначным соответствием изображения химической молекулы и её физической структурой.

Ярким примером последней ситуации являются изображения алленов. Это соединения, представляющие из себя углеводороды и их замещённые производные с двумя двойными связями у одного атома углерода общей формулы $RR'C=C=CR''R'''$ [11]. Известно, что у таких соединений плоскости, в которых находятся атомы R, R', C и C, R'', R''' соответственно, являются ортогональными. Для того, чтобы отразить эту ортогональность на рисунке, необходимо нарисовать две клиновидных связи. Однако из определения алленов понятно, что молекула, состоящая из двух двойных соединяющих атомы углеродов связей, из концов которых выходят ещё по две одинарные связи, обязательно будет являться именно алленом, а не каким-то другим соединением. Поэтому достаточно указывать только один клин – для того, чтобы определить, какой из атомов – R или R' – находится ближе к наблюдателю, а какой дальше. Таким образом одну и ту же молекулу можно изобразить как минимум тремя способами².

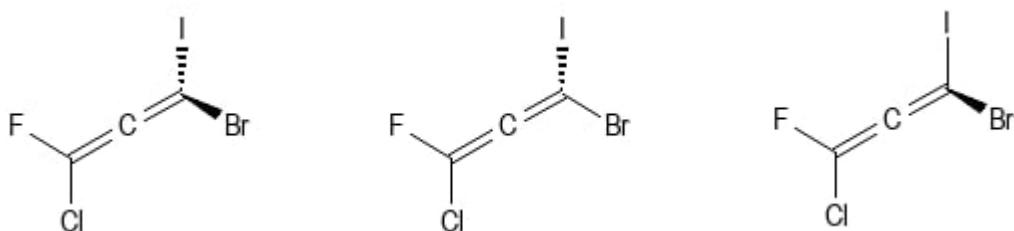


Рис.3.1. Три разных изображения одного и того же аллена

3.2. Качество конвейера моделей

Для проведения качественного тестирования моделей были использованы данные, ранее не задействованные при обучении. Было взято 20000 разнообразных молекул и посчитано среднее расстояние Левенштейна. Оно составило 4.32 редакционных изменения.

Процент точных совпадений молекул составил 81.8%. Учитывая, что точность генератора составляет 92.8%, можно сделать вывод, что истинная точность конвейера моделей составляет около 81.8%, что является отличным показателем для object detection задачи, поскольку задача распознавания при таком подходе состоит из большого числа распознаваний, в каждом из которых есть шанс ошибиться и, таким образом, предоставить неточное предсказание.

²На самом деле таких способов шесть. Можно указывать клиновидность связей, находящихся в правой части изображения – CR'' и CR'''

3.3. Типовые ошибки конвейера. Методы устранения

Наиболее часто встречающиеся ошибки связаны с распознаванием меток атомов кислорода и азота с указанием атомов водорода – O , NH , NH_2 . Причина заключается в том, что обучающая выборка является несбалансированной: количество меток атомов в несколько раз меньше, чем количество меток связей. Для решения данной проблемы необходимо генерировать набор данных, содержащий большее количество меток атомов при том же количестве связей. Реализация данной функциональности предполагает разработку алгоритма, который сможет менять атомы углерода на другие атомы, соблюдая при этом требования, касающие валентности: количество связей, которые исходят из данного атома, должно быть в списке возможных валентностей. Такой алгоритм может быть реализован при дальнейших исследованиях в рамках данной работы.

Подобные ошибки происходят приблизительно в одном из 15-20 случаев.

3.4. Производительность решения

Были произведены замеры производительности на ноутбуке следующей конфигурации:

- CPU: AMD Ryzen 5 5600H
- GPU: Nvidia GeForce RTX3050
- RAM: 16 Gb

Для данной конфигурации была производительность решения составила 3.11 изображений в секунду. Из них 45% уходит на операции RDKit по добавлению элементов во внутреннее представление молекулы и её валидацию. Также около 8% уходит на обмен данными между CPU и GPU, а оставшиеся 47% занимает инференс моделей и сборка графа молекулы.

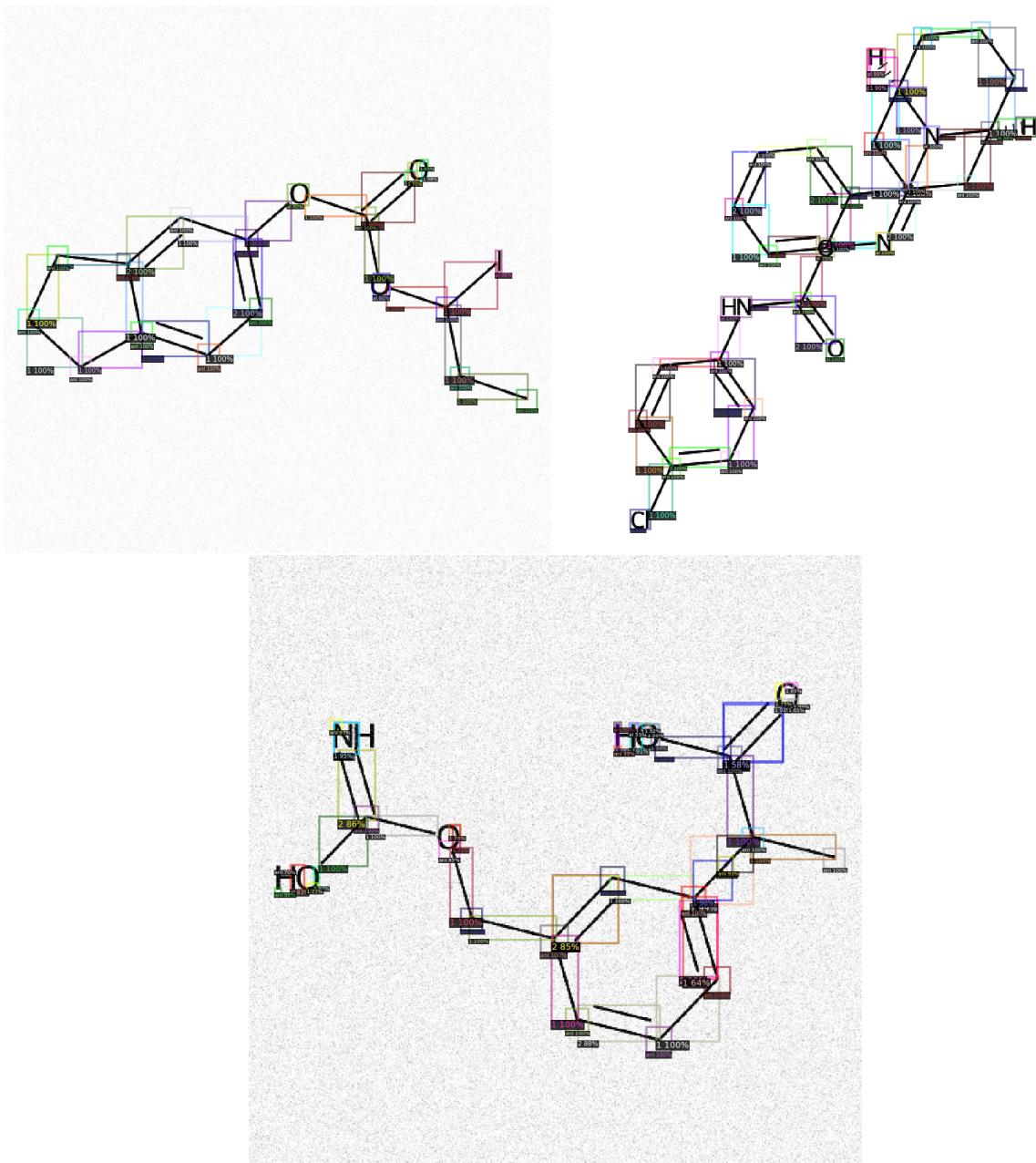


Рис.3.2. Типовые ошибки распознавания Faster R-CNN: некорректно построенные баундбоксы атомов (1) кислорода, (2) водорода, (3) азота

ЗАКЛЮЧЕНИЕ

В результате проделанной работы был разработан программный продукт, который позволяет распознавать структуру химической молекулы с применением object detection подхода с точностью 81.8%. Было проведено исследование, результатом которого стало понимание особенностей работы object detection модели Detectron2 Faster R-CNN на изображениях химических молекул.

Учитывая хорошую разделимость этих данных, можно сделать вывод, что модуль классификации Detectron2 достаточно плохо справляется с возложенными

на него обязанностью. Однако регрессор баундбоксов работает очень качественно, в связи с чем можно дать общую рекомендацию по применению таких сетей.

Данная сеть обучается очень быстро и не склонна существенно улучшать качество работы после первых нескольких тысяч итераций, несмотря на то, что loss-функция выходит на асимптоту только приблизительно к ста тысячной итерации. Стоит обучать такую сеть на поиск баундбоксов, а задачу классификации отдавать в следующую сеть, скорее всего простую по своей структуре. Таким образом, удастся качественно выполнить поставленную задачу и, несмотря на кажущуюся громоздкость и дублирование функциональности двух моделей, практически не потерять в производительности инференса.

Кроме того, найдены проблемные места в построенном решении: сеть иногда распознаёт очень большое количество баундбоксов на метке атома кислорода и в целом склонна к некорректному распознаванию меток атомов. Это связано с тем, что образцов меток атомов у сети во много раз меньше, чем образцов связей, в связи с чем сеть недоучивается распознавать метки атомов, и вместо них видит связи. Для исправления этой ситуации необходимо реализовать генератор химических соединений, который позволит создавать химически корректные молекулы с большим числом атомов с указываемыми метками (преимущественно азот и кислород).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Detectron2 official docs. — URL: <https://detectron2.readthedocs.io/en/latest/> (дата обращения: 28.05.2023).
2. DETR: End-to-End Object Detection with Transformers. — URL: <https://github.com/facebookresearch/detr> (дата обращения: 28.05.2023).
3. International Chemical Identifier. Википедия. — URL: https://ru.wikipedia.org/wiki/International_Chemical_Identifier (дата обращения: 28.05.2023).
4. International Chemical Identifier. Сайт организации. — URL: <https://www.inchi-trust.org/> (дата обращения: 28.05.2023).
5. Kaggle соревнование BMS. — URL: <https://www.kaggle.com/c/bms-molecular-translation> (дата обращения: 28.05.2023).
6. Object Detection. Распознавай и властвуй. Часть 1. — URL: <https://habr.com/ru/companies/jetinfosystems/articles/498294/> (дата обращения: 28.05.2023).
7. Object Detection. Распознавай и властвуй. Часть 2. — URL: <https://habr.com/ru/companies/jetinfosystems/articles/498652/> (дата обращения: 28.05.2023).
8. RDKit: Open-Source Cheminformatics Software. — URL: <https://www.rdkit.org/> (дата обращения: 28.05.2023).
9. Selective Search Algorithm. — URL: <https://www.geeksforgeeks.org/selective-search-for-object-detection-r-cnn/> (дата обращения: 28.05.2023).
10. Stereochemistry. Cis-trans and E-Z Isomerism. — URL: <https://blogs.ntu.edu.sg/cy1101-1819s1-g09/2018/10/cis-trans-and-e-z-isomerism/> (дата обращения: 28.05.2023).
11. Аллены. Википедия. — URL: <https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%BB%D0%B5%D0%BD%D1%8B> (дата обращения: 28.05.2023).
12. A review of optical chemical structure recognition tools / K. Rajan [и др.] // Journal of Cheminformatics. — 2020. — Т. 12, № 1. — С. 1—13.
13. Chemical machine vision: automated extraction of chemical metadata from raster images / G. V. Gkoutos [и др.] // Journal of chemical information and computer sciences. — 2003. — Т. 43, № 5. — С. 1342—1355.
14. Cortes C., Vapnik V. Support-vector networks // Machine learning. — 1995. — Т. 20. — С. 273—297.
15. Molecular structure extraction from documents using deep learning / J. Staker [и др.] // Journal of chemical information and modeling. — 2019. — Т. 59, № 3. — С. 1017—1029.

Приложение 1

Архитектура Faster R-CNN

Полная архитектура Faster R-CNN выглядит следующим образом:

```

1 GeneralizedRCNN(
2     (backbone): FPN(
3         (fpn_output1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
4         (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
5         (fpn_output3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
6         (fpn_output4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), padding=(1, 1))
7         (fpn_output5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1), padding=(1, 1))
8         (fpn_output6): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
9         (fpn_output7): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
10        (fpn_output8): Conv2d(2048, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
11        (top_block): BottleneckMuxpool()
12    (bottom_up): ResNet(
13        (stem): BasicStem(
14            (conv1): Conv2d(
15                3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False
16                (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
17            )
18        )
19        (res2): Sequential(
20            (0): BottleneckBlock(
21                (shortcut): Conv2d(
22                    64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
23                    (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
24                )
25                (conv1): Conv2d(
26                    64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
27                    (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
28                )
29                (conv2): Conv2d(
30                    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
31                    (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
32                )
33                (conv3): Conv2d(
34                    64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
35                    (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
36                )
37            )
38            (1): BottleneckBlock(
39                (conv1): Conv2d(
40                    256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
41                    (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
42                )
43                (conv2): Conv2d(
44                    64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
45                    (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
46                )
47            )
48            (conv3): Conv2d(
49                64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
50                (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
51            )
52        )
53        (2): BottleneckBlock(
54            (conv1): Conv2d(
55                256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
56                (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
57            )
58            (conv2): Conv2d(
59                64, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
60                (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
61            )
62            (conv3): Conv2d(
63                64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
64                (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
65            )
66        )
67        (res3): Sequential(
68            (0): BottleneckBlock(
69                (shortcut): Conv2d(
70                    256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
71                    (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
72                )
73                (conv1): Conv2d(
74                    256, 128, kernel_size=(1, 1), stride=(2, 2), bias=False
75                    (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
76                )
77                (conv2): Conv2d(
78                    128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
79                    (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
80                )
81                (conv3): Conv2d(
82                    128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
83                    (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
84                )
85            )
86            (1): BottleneckBlock(
87                (conv1): Conv2d(
88                    512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
89                    (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
90                )
91            )
92        )
93    )
94)

```

Рис.П1.1. Архитектура Faster R-CNN (1)

```

91     (conv2d):
92       128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
93       (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
94   )
95   (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
96     (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
97   )
98 )
99 (2): BottleneckBlock(
100   (conv1): Conv2d(
101     512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
102     (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
103   )
104   (conv2): Conv2d(
105     128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
106     (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
107   )
108   (conv3): Conv2d(
109     128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
110     (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
111   )
112 )
113 (3): BottleneckBlock(
114   (conv1): Conv2d(
115     512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
116     (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
117   )
118   (conv2): Conv2d(
119     128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
120     (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
121   )
122   (conv3): Conv2d(
123     128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
124     (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
125   )
126 )
127 )
128 (res4): Sequential(
129   (0): BottleneckBlock(
130     (shortcut): Conv2d(
131       512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False
132       (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
133     )
134     (conv1): Conv2d(
135       512, 256, kernel_size=(1, 1), stride=(2, 2), bias=False
136       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
137     )
138   )
139   (conv3): Conv2d(
140     256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
141     (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
142   )
143   (4): BottleneckBlock(
144     (conv1): Conv2d(
145       1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
146       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
147     )
148     (conv2): Conv2d(
149       256, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
150       (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
151     )
152   )
153   (conv3): Conv2d(
154     1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
155     (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
156   )
157   (5): BottleneckBlock(
158     (conv1): Conv2d(
159       256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
160       (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
161     )
162     (conv2): Conv2d(
163       1024, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
164       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
165     )
166   )
167   (conv3): Conv2d(
168     256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
169     (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
170   )
171   (6): BottleneckBlock(
172     (conv1): Conv2d(
173       1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
174       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
175     )
176     (conv2): Conv2d(
177       256, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
178       (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
179     )
180     (conv3): Conv2d(
181       1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
182       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
183     )
184 )
185   (conv3): Conv2d(
186     256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
187     (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
188   )
189   (4): BottleneckBlock(
190     (conv1): Conv2d(
191       1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
192       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
193     )
194     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
195       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
196     )
197     (conv3): Conv2d(
198       256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
199       (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
200     )
201   )
202   (5): BottleneckBlock(
203     (conv1): Conv2d(
204       1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
205       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
206     )
207     (conv2): Conv2d(
208       256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
209       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
210     )
211     (conv3): Conv2d(
212       256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
213       (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
214     )
215   )
216 )
217 (res5): Sequential(
218   (0): BottleneckBlock(
219     (shortcut): Conv2d(
220       1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False
221       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
222     )
223     (conv1): Conv2d(
224       1024, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
225       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
226     )
227     (conv2): Conv2d(
228       512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
229       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
230     )
231   )
232   (conv3): Conv2d(
233     512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
234     (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
235   )
236   (1): BottleneckBlock(
237     (conv1): Conv2d(
238       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
239       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
240     )
241     (conv2): Conv2d(
242       512, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
243       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
244     )
245     (conv3): Conv2d(
246       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
247       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
248   )
249   (2): BottleneckBlock(
250     (conv1): Conv2d(
251       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
252       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
253     )
254     (conv2): Conv2d(
255       512, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
256       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
257     )
258     (conv3): Conv2d(
259       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
260       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
261   )
262   (3): BottleneckBlock(
263     (conv1): Conv2d(
264       512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
265       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
266     )
267     (proposal_generator): RPN(
268       (rpn_head): StandardRPNHead(
269         (conv): Conv2d(
270           256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
271           (activation): ReLU()
272         )
273         (objectness_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
274         (anchor_deltas): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
275       )
276     )
277   )
278   (cell_anchors): BufferList()
279 )
280   (roi_heads): StandardROIHeads(
281     (box_pooler): ROIPooler(
282       (level_poolers): ModuleList(
283         (0): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0, aligned=True)
284         (1): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.125, sampling_ratio=0, aligned=True)
285         (2): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.0625, sampling_ratio=0, aligned=True)
286         (3): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.03125, sampling_ratio=0, aligned=True)
287       )
288     )
289     (box_head): FastRCNNConvFCHead(
290       (flatten): Flatten(start_dim=1, end_dim=-1)
291       (fc1): Linear(in_features=12544, out_features=1024, bias=True)
292       (fc_relu1): ReLU()
293       (fc2): Linear(in_features=1024, out_features=1024, bias=True)
294     )
295     (fc_relu2): ReLU()
296   )
297   (box_predictor): FastRCNNOutputLayers(
298     (cls_score): Linear(in_features=1024, out_features=6, bias=True)
299     (bbox_pred): Linear(in_features=1024, out_features=20, bias=True)
300   )
301 )

```

Рис.П1.2. Архитектура Faster R-CNN (2)

```

184   (conv3): Conv2d(
185     256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
186     (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
187   )
188   (4): BottleneckBlock(
189     (conv1): Conv2d(
190       1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
191       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
192     )
193     (conv2): Conv2d(
194       256, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
195       (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
196     )
197     (conv3): Conv2d(
198       1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
199       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
200     )
201   )
202   (5): BottleneckBlock(
203     (conv1): Conv2d(
204       1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
205       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
206     )
207     (conv2): Conv2d(
208       256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
209       (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
210     )
211     (conv3): Conv2d(
212       256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
213       (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
214     )
215   )
216 )
217 (res5): Sequential(
218   (0): BottleneckBlock(
219     (shortcut): Conv2d(
220       1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False
221       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
222     )
223     (conv1): Conv2d(
224       1024, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
225       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
226     )
227     (conv2): Conv2d(
228       512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
229       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
230     )
231   )
232   (conv3): Conv2d(
233     512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
234     (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
235   )
236   (1): BottleneckBlock(
237     (conv1): Conv2d(
238       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
239       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
240     )
241     (conv2): Conv2d(
242       512, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
243       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
244     )
245     (conv3): Conv2d(
246       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
247       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
248   )
249   (2): BottleneckBlock(
250     (conv1): Conv2d(
251       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
252       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
253     )
254     (conv2): Conv2d(
255       512, 2048, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False
256       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
257     )
258     (conv3): Conv2d(
259       2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
260       (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
261   )
262   (3): BottleneckBlock(
263     (conv1): Conv2d(
264       512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False
265       (norm): FrozenBatchNorm2d(num_features=2048, eps=1e-05)
266     )
267     (proposal_generator): RPN(
268       (rpn_head): StandardRPNHead(
269         (conv): Conv2d(
270           256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
271           (activation): ReLU()
272         )
273         (objectness_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
274         (anchor_deltas): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
275       )
276     )
277   )
278   (cell_anchors): BufferList()
279 )
280   (roi_heads): StandardROIHeads(
281     (box_pooler): ROIPooler(
282       (level_poolers): ModuleList(
283         (0): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0, aligned=True)
284         (1): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.125, sampling_ratio=0, aligned=True)
285         (2): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.0625, sampling_ratio=0, aligned=True)
286         (3): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.03125, sampling_ratio=0, aligned=True)
287       )
288     )
289     (box_head): FastRCNNConvFCHead(
290       (flatten): Flatten(start_dim=1, end_dim=-1)
291       (fc1): Linear(in_features=12544, out_features=1024, bias=True)
292       (fc_relu1): ReLU()
293       (fc2): Linear(in_features=1024, out_features=1024, bias=True)
294     )
295     (fc_relu2): ReLU()
296   )
297   (box_predictor): FastRCNNOutputLayers(
298     (cls_score): Linear(in_features=1024, out_features=6, bias=True)
299     (bbox_pred): Linear(in_features=1024, out_features=20, bias=True)
300   )
301 )

```

Рис.П1.3. Архитектура Faster R-CNN (3)

```

276   (anchor_generator): DefaultAnchorGenerator(
277     (cell_anchors): BufferList()
278   )
279 )
280   (roi_heads): StandardROIHeads(
281     (box_pooler): ROIPooler(
282       (level_poolers): ModuleList(
283         (0): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.25, sampling_ratio=0, aligned=True)
284         (1): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.125, sampling_ratio=0, aligned=True)
285         (2): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.0625, sampling_ratio=0, aligned=True)
286         (3): ROIAxisAlign(output_size=(7, 7), spatial_scale=0.03125, sampling_ratio=0, aligned=True)
287       )
288     )
289     (box_head): FastRCNNConvFCHead(
290       (flatten): Flatten(start_dim=1, end_dim=-1)
291       (fc1): Linear(in_features=12544, out_features=1024, bias=True)
292       (fc_relu1): ReLU()
293       (fc2): Linear(in_features=1024, out_features=1024, bias=True)
294     )
295     (fc_relu2): ReLU()
296   )
297   (box_predictor): FastRCNNOutputLayers(
298     (cls_score): Linear(in_features=1024, out_features=6, bias=True)
299     (bbox_pred): Linear(in_features=1024, out_features=20, bias=True)
300   )
301 )

```

Рис.П1.4. Архитектура Faster R-CNN (4)