



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 _____ **«Информатика и системы управления»**
КАФЕДРА _____ **«Программное обеспечение ЭВМ и информационные технологии»**

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
НА ТЕМУ:**

**«Метод распознавания челюстно–лицевых костей черепа по
томографическим снимкам головы человека на основе
обратного распространения ошибки и опорных векторов»**

Студент группы ИУ7–83Б

(Подпись, дата)

С. Кононенко

(И.О. Фамилия)

Руководитель

(Подпись, дата)

М.В. Филиппов

(И.О. Фамилия)

Консультант

(Подпись, дата)

Ю.М. Гаврилова

(И.О. Фамилия)

Нормоконтроллер

(Подпись, дата)

(И.О. Фамилия)

Москва — 2022г.

РЕФЕРАТ

Расчетно–пояснительная записка 83 с., 45 рис., 2 табл., 30 ист., 3 прил.

В работе представлена разработка метода распознавания зубочелюстных и челюстно–лицевых костей черепа по томографическим снимкам головы человека.

Рассмотрена задача распознавания образов на изображении. Рассмотрены методы и технологии распознавания образов при помощи нейронных сетей, их применимость при работе с томографическими снимками. Рассмотрены сопутствующие методы и подходы при реализации метода распознавания образов. Представлена реализация метода распознавания челюстно–лицевых костей черепа человека, проведены исследования эффективности разработанного метода на различных типах томографических снимков.

СОДЕРЖАНИЕ

РЕФЕРАТ	5
ВВЕДЕНИЕ	9
1 Аналитический раздел	11
1.1 Томография в медицине	11
1.1.1 Компьютерная томография	11
1.1.2 Магнитно–резонансная томография	12
1.2 Задача распознавания образов	13
1.3 Задача нахождения объектов на изображении	15
1.4 Нейронные сети	16
1.4.1 Принцип работы нейронных сетей	17
1.4.2 Метод обратного распространения ошибки	20
1.4.3 Виды нейронных сетей	23
1.4.4 Сравнение нейронных сетей и глубокого обучения	24
1.4.5 Особенности применения нейронных сетей в качестве классификаторов	25
1.4.6 Машина опорных векторов	27
1.5 Методы и технологии распознавания образов	30
1.5.1 R–CNN	30
1.5.2 Fast R–CNN	32
1.5.3 Faster R–CNN	33
1.5.4 Сравнение рассмотренных методов	34
1.6 Методы и технологии выделения объектов на изображении	35
1.6.1 Mask R–CNN	35
1.6.2 UNet	36
1.6.3 Сравнение рассмотренных методов	38
1.7 Выбор данных для обучения модели	38
2 Конструкторский раздел	40
2.1 Требования к разрабатываемому методу	40
2.2 Требования к разрабатываемому программному комплексу	40
2.3 Проектирование метода распознавания	40
2.3.1 Физиология зубочелюстного сустава	40
2.3.2 Структура вектора особенностей	41
2.3.3 IDEF0–диаграмма	42
2.4 Структура разрабатываемого программного комплекса	42
2.4.1 Модуль UNet модели	43
2.4.2 Модуль пользовательского приложения	43
2.5 Данные для обучения модели	44

3 Технологический раздел	46
3.1 Средства реализации программного комплекса	46
3.1.1 Выбор языка программирования	46
3.1.2 Выбор библиотеки глубокого обучения	46
3.1.3 Выбор средства реализации машины опорных векторов	46
3.2 Реализация программного комплекса	46
3.2.1 Модель UNet	46
3.2.2 Тренировка модели	47
3.2.3 Классификация зубов при помощи машины опорных векторов	50
3.2.4 Выделение сегментированных участков	50
3.3 Результаты обучения модели	52
3.4 Примеры использования разработанного программного комплекса	53
3.4.1 Пример использования модуля модели UNet	53
3.4.2 Пример использования модуля пользовательского приложения	54
4 Исследовательский раздел	56
4.1 Сравнение применимости программного комплекса для снимков в разных проекциях	56
4.2 Сравнение применимости программного комплекса для снимков в разных оттенках	58
4.3 Сравнение времени работы приложения при различном разрешении исходного снимка	60
4.4 Оценка разработанного программного комплекса	62
ЗАКЛЮЧЕНИЕ	63
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	64
ПРИЛОЖЕНИЕ А	67
ПРИЛОЖЕНИЕ Б	74
ПРИЛОЖЕНИЕ В	81

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

- 1) КТ (компьютерная томография) — метод неразрушающего послойного исследования внутренней структуры объекта посредством сканирующего просвечивания его рентгеновским излучением [1].
- 2) МРТ (магнитно–резонансная томография) — способ получения томографических медицинских изображений для исследования внутренних органов и тканей с использованием явления ядерного магнитного резонанса [2].
- 3) MLP (от англ. Multilayer perceptron) — многослойный персепtron.
- 4) CNN (от англ. Convolutional neural network) — сверточная нейронная сеть.
- 5) RNN (от англ. Recurrent neural network) — рекуррентная нейронная сеть.
- 6) SVM (от англ. Support vector machine) — метод машинного обучения, применяемый в задачах классификации, основанный на построении гиперплоскости и ее анализе [3].

ВВЕДЕНИЕ

Согласно исследованиям [4][5], на момент 2018 года 10% всех несчастных случаев и обращений в отделения неотложной помощи приходятся на травмы челюстно–лицевой области головы человека, а 1% всех пластических операций — на контурную пластику (изменение формы) лица.

Для лечения лицевых повреждений требуется диагностика типа травмы, а также причины ее появления. Для диагностики обычно используются КТ и МРТ снимки головы. На основе полученных снимков можно выбрать план лечения.

Нередко при лечении челюстно–лицевых повреждений требуется хирургическое вмешательство, а проведение безоперационной контурной пластики дает непостоянный результат, требующий повторной процедуры спустя некоторое время. Поэтому пациенты часто прибегают к операции, чтобы добиться долгосрочных изменений [5].

Проведение операций по лечению дефектов челюстно–лицевого сустава может быть затруднительным ввиду недостаточных данных, полученных при сборе анамнеза [6]. Для уточнения и расширения показаний пациента можно воспользоваться компьютерными технологиями для распознавания челюстно–лицевых костей и их повреждений. Использование нейронных сетей глубокого обучения позволяет повысить эффективность определения травм лица, что в свою очередь снижает риск замедления лечения и выбора неправильного подхода к реабилитации [7].

Компьютерная диагностика также позволяет выявить дефекты в строении зубочелюстной системы или предотвратить развитие патологических отклонений. Кроме того, она может использоваться как вспомогательный инструмент при проектировании зубочелюстных протезов.

Цель работы — разработать метод распознавания челюстно–лицевых костей черепа по томографическим снимкам головы человека.

Для достижения поставленной цели потребуется:

- Описать термины предметной области и обозначить проблему;
- Описать технологии, с помощью которых можно реализовать метод распознавания челюстно–лицевых костей;
- Разработать метод распознавания челюстно–лицевых костей черепа по

томографическим снимкам головы человека.

- Разработать программный комплекс, реализующий интерфейс для взаимодействия с разработанным методом.
- Исследовать разработанный метод на применимость при работе с различными типами томографических снимков и при работе с различными проекциями одного снимка.

1. Аналитический раздел

В данном разделе описываются медицинские методы томографии. Рассматриваются задачи распознавания образов и нахождения объектов на изображении. Рассматривается технология нейронных сетей, приводится обзор принципа работы нейронных сетей. Описываются виды нейронных сетей, их применение в задачах классификации и сегментации. Даётся обзор существующих методов распознавания и нахождения объектов на изображении.

1.1 Томография в медицине

Томография основана на получении послойных изображений объекта исследований. Метод томографии был разработан Н.И. Пироговым. Современная томография также основана на получении послойных изображений. Однако она позволяет проводить прижизненные исследования и с помощью математической обработки производить трехмерную реконструкцию изображения исследуемого объекта.

В медицине используются различные методы томографии: компьютерная томография (КТ), магнитно–резонансная томография (МРТ), однофотонная эмиссионная компьютерная томография (ОЭФКТ) и позитронная эмиссионная томография (ПЭТ).

В контексте работы будут рассмотрены магнитно–резонансная и компьютерная томографии, как наиболее информативные и высокоточные методы инструментальной диагностики [8].

1.1.1 Компьютерная томография

Компьютерная томография (КТ) — способ получения послойных срезов тела человека или другого объекта с помощью рентгеновских лучей. Этот метод для диагностических целей был предложен к использованию в 1972 году. Основателями компьютерной томографии принято считать Годфри Хаунсфилда и Алана Кормака, получившими за свои разработки Нобелевскую премию. В основе компьютерной томографии лежит измерение разницы ослабления рентгеновского излучения различными тканями, обработка полученных данных компьютером с помощью математических алгоритмов и формирование графической визуализации органов человека на экране с последующей их интерпретацией врачом–радиологом.

Для получения среза трубка оборачивается вокруг пациента на 360 гра-

дусов, толщина среза при этом задается заранее. В обычном КТ–сканере трубка вращается постоянно, излучение расходится веерообразно. Рентгеновская трубка и принимающее устройство (детектор) спарены, их вращение вокруг сканируемой зоны происходит синхронно: рентгеновское излучение улавливается детекторами, расположенными на противоположной стороне. Веерообразное расхождение происходит под углом от 40 до 60 градусов в зависимости от конкретного аппарата [9].

На рисунке 1.1 представлен принцип работы компьютерного томографа.

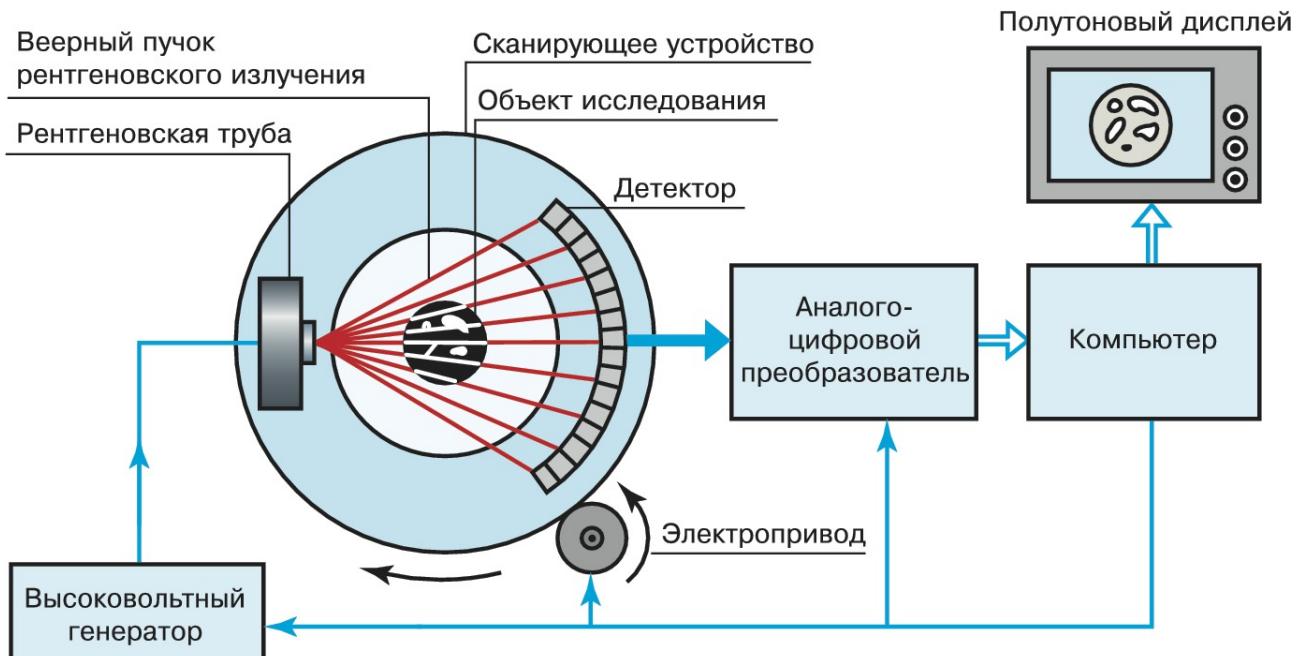


Рис. 1.1: Принцип работы компьютерного томографа

1.1.2 Магнитно–резонансная томография

Магнитно–резонансная томография (МРТ) — способ исследования химических процессов в организме человека и его мягких тканей.

Принцип работы аппарата МРТ основан на воздействии радиосигналов на атомы водорода в теле человека. Будучи помещенными в сильное магнитное поле, атомы резонируют и выдают различные сигналы, на основе анализа которых можно сделать вывод о характере заболевания. Снимки МРТ, которые получаются в результате исследования, позволяют увидеть даже минимальные опухоли и небольшие воспалительные процессы во всех тканях, содержащих воду.

На монитор компьютера выводится трехмерная проекция изучаемого органа, которую можно рассмотреть в любых плоскостях. МР–томограф дает ре-

зультаты, существенно отличающиеся от получаемых при помощи компьютерной томографии или рентгена. Компьютерная томография ориентирована на изучение физических процессов в мягких тканях, а рентген предназначен для исследования твердых тканей (костей). С помощью МРТ можно изучать химическое строение тканей и функционирование систем организма в динамике, что позволяет получать полную картину состояния того или иного органа [10].

На рисунке 1.2 представлен принцип работы магнитно–резонансного томографа.

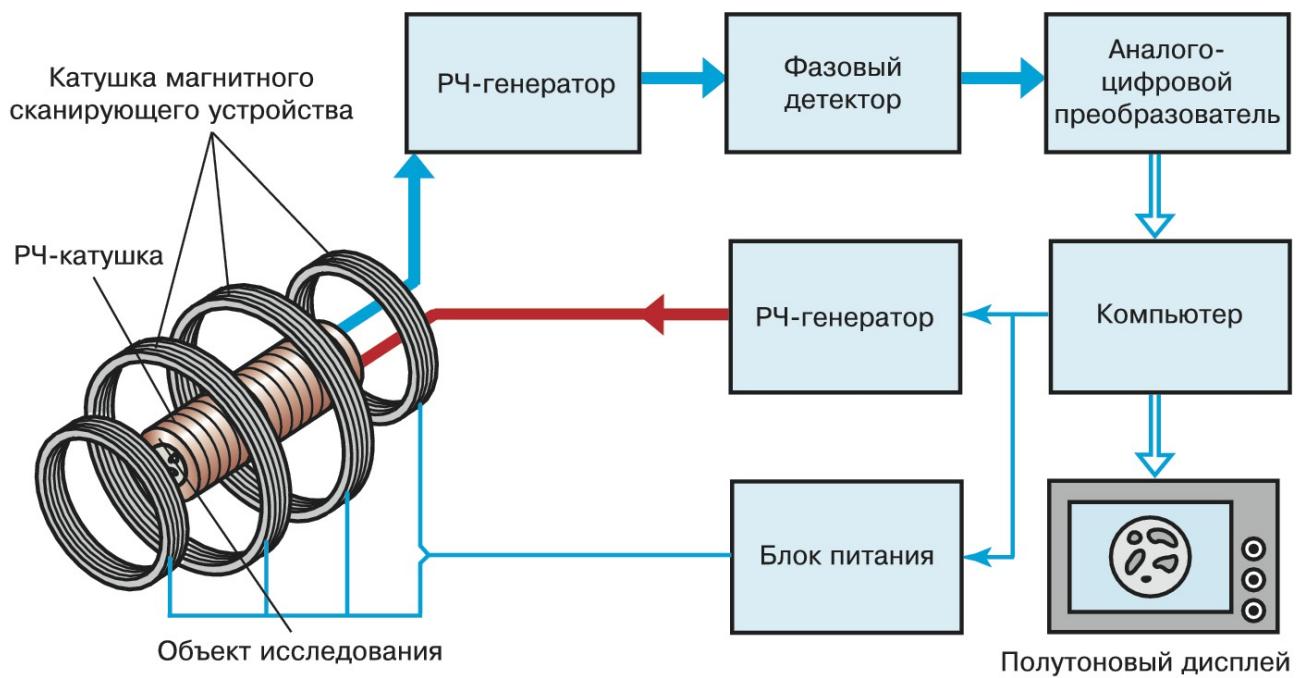


Рис. 1.2: Принцип работы магнитно-резонансного томографа

1.2 Задача распознавания образов

Задачу распознавания образов можно отнести к задаче классификации [11]. В контексте работы под образами понимаются кости челюстно–лицевого и зубочелюстного отделов черепа человека. Классификация является одной из важнейших задач интеллектуального анализа данных. Она решается с помощью аналитических моделей, называемых классификаторами. Актуальность классификации обусловлена сравнительной простотой алгоритмов и методов ее реализации, и высокой интерпретируемостью результатов по сравнению с другими технологиями анализа данных.

В настоящее время разработано большое количество различных видов классификаторов, для построения которых используются как статистические методы (логистическая регрессия, дискриминантный анализ), так и методы ма-

шинного обучения (нейронные сети, деревья решений, метод k–ближайших соседей, машины опорных векторов) [12].

Необходимость использования в анализе данных большого числа разнообразных методов классификации обусловлена тем, что решаемые с ее помощью задачи могут иметь свои особенности, связанные, например, с числом классов (бинарная классификация или классификация с несколькими классами) или с представлением исходных данных — их объемом, размерностью и качеством, что требует выбора адекватного классификатора. Выбор классификатора, соответствующего особенностям решаемой задачи анализа, является важным фактором получения правильного решения [13].

Различные виды классификаторов имеют свои преимущества и недостатки. Так, классификаторы, в которых используются методы статистики имеют хорошую математическую обоснованность, но при этом сложны в использовании и требуют знания вероятностного распределения исходных данных и оценки его параметров, а также имеют фиксированную структуру модели. Такие классификаторы называются параметрическими. Статистические методы оценивают только вероятность принадлежности объекта классу, но не причину этой принадлежности.

Классификаторы, основанные на машинном обучении, не требуют оценки параметров распределения исходных данных, а мера сходства в них формализуется с помощью функции расстояния (обычно, евклидова) [14]. Такие классификаторы называются метрическими. Они проще в реализации и использовании, чем параметрические, а их результаты удобнее для интерпретации. При этом метрические классификаторы представляют собой эвристические модели. Они обеспечивают решение только в ограниченном числе практически значимых случаев, могут дать неточное или не единственное решение.

Компромиссом между параметрическим и метрическими методами является использование для решения задачи классификации нейронных сетей. Нейронные сети являются непараметрическими моделями, которые не требуют предположений о вероятностном распределении данных, но при этом и не используют меры расстояний. Это делает их универсальными классификаторами, позволяя получать результаты даже в случаях, когда параметрические и метрические классификаторы не обеспечивают приемлемого решения.

1.3 Задача нахождения объектов на изображении

Задача нахождения объектов на изображении — это задача, в рамках которой выполняется определение наличия или отсутствия объекта определённого домена на изображении, нахождение границ этого объекта в системе координат пикселей исходного изображения.

Данная задача решается при помощи алгоритмов машинного обучения с применением нейронных сетей. В зависимости от алгоритма обучения, объект может характеризоваться координатами ограничивающей рамки, ключевыми точками, контуром объекта.

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

В контексте работы будут рассмотрены задачи семантической сегментации и сегментации экземпляров.

Решением задачи семантической сегментации является признак принадлежности пикселя изображения к определенной категории. Например, если в исходном изображении представлено море, над которым пролетают чайки, то для каждого пикселя необходимо вывести, является ли этот пиксель частью чайки, моря, неба, или какого-то другого типа. Недостаток применения только семантической сегментации относительно задач, связанных с распознаванием объектов — маркировка пикселей по принадлежности только к типу объекта, что не создаёт различия между объектами как таковыми. Если назвать «объектом» связную область пикселей, характеризующих одинаковый тип, то два объекта, перегораживающих друг друга на исходном изображении, будут определены как один объект. В некоторых случаях это приемлемо (как например в задаче данной работы при распознавании зубов в зубочелюстной области). Задача семантической сегментации изображения с дифференцированием объектов называется задачей сегментации экземпляров. Модели, решающие задачу сегментации экземпляров, применяются при необходимости поэкземплярно отделить распознанные объекты, например, разделить распознанных чаек из представленного примера.

1.4 Нейронные сети

Нейронные сети, известные также как *искусственные нейронные сети* или *смоделированные нейронные сети*, являются подмножеством алгоритмов машинного обучения и служат основой для алгоритмов глубокого обучения. Понятие «нейронные сети» возникло при попытке смоделировать процессы, происходящие в человеческом мозге при передаче сигналов между биологическими нейронами [15].

Искусственные нейронные сети состоят из образующих слои узлов: слой входных данных, один или несколько скрытых слоев и слой выходных данных. Каждый узел (искусственный нейрон) связан с другими узлами с определенным весом и пороговым значением. Если вывод какого-либо узла превышает пороговое значение, то этот узел активируется и отправляет данные на следующий уровень сети. В противном случае данные на следующий уровень сети не передаются.

На рисунке 1.3 представлена модель работы нейронной сети, где x_i — входные данные, а Y_j — выходные данные.

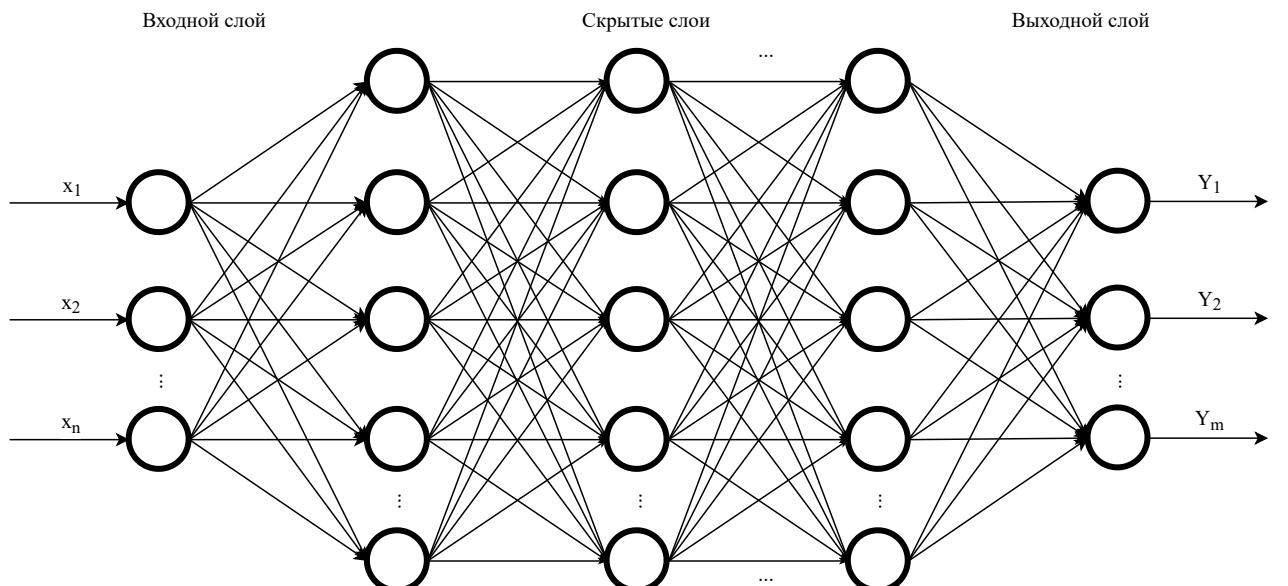


Рис. 1.3: Модель нейронной сети

Для обучения и постепенного повышения точности нейронных сетей применяются обучающие данные. При достижении требуемой точности алгоритмы обучения превращаются в мощные инструменты для вычислений и искусственного интеллекта, что позволяет использовать их для классификации и кластеризации данных с высокой скоростью. Задачи из области распознавания

речи или изображений можно выполнить за несколько минут, а не за несколько часов, как при распознавании вручную.

1.4.1 Принцип работы нейронных сетей

Каждый отдельный узел нейронной сети можно представить в виде модели линейной регрессии [16], состоящей из входных данных, весовых коэффициентов, смещения (или порогового значения) и выходных данных. Эту модель можно описать следующей формулой:

$$\hat{y} = \sum_{i=1}^m w_i x_i + bias, \quad (1)$$

где \hat{y} — прогнозируемое значение модели;

m — количество факторов;

w — весовой коэффициент фактора;

x — единица входных данных;

$bias$ — значение ошибки.

Выходное значение для узла описывается по формуле:

$$f(x_i) = \begin{cases} 1 & \text{если } \hat{y} \geq 0 \\ 0 & \text{иначе.} \end{cases} \quad (2)$$

После определения слоя входных данных необходимо назначить весовые коэффициенты. Они помогают определить важность фактора: чем выше весовой коэффициент, тем существеннее его вклад в выходные данные по сравнению с другими входными данными. Затем произведения входных данных и соответствующих им весовых коэффициентов суммируются. Наконец, выходные данные передаются через функцию активации 2, которая вычисляет результат. Если полученный результат превышает установленное пороговое значение, узел срабатывает (активируется), передавая данные на следующий слой сети. Выходные данные одного узла становятся входными данными для следующего узла. Такой последовательный процесс передачи данных между слоями характерен для нейронных сетей прямого распространения.

Для более наглядной демонстрации этой концепции рассмотрим реальный пример: допустим, нужно принять решение, стоит ли идти на серфинг (да (1), нет (0)). Решение «идти» или «не идти» — прогнозируемый результат \hat{y} .

Предположим, существует три фактора, которые влияют на принятие решения:

- 1) Хорошие ли сегодня волны? (да (1), нет (0))
- 2) Свободна ли зона для плавания? (да (1), нет (0))
- 3) Были ли случаи нападения акул в последнее время? (да (1), нет (0))

Предположим, что имеются следующие входные данные:

- $x_1 = 1$, так как сегодня хорошие волны для серфинга;
- $x_2 = 0$, так как уже собралось много серферов;
- $x_3 = 1$, так как в последнее время не было нападений акул.

Теперь нужно присвоить весовые коэффициенты для определения важности. Чем выше значение весового коэффициента, тем большим будет влияние конкретной переменной на решение или результат.

- $w_1 = 5$, так как большие волны — редкость;
- $w_2 = 2$, так как скопление серферов не является проблемой;
- $w_3 = 4$, так как присутствует страх акул.

Приняв пороговое значение (*bias*) равным 3 (случайно выбранное значение для примера), можно подставить значения в формулу 1 и получить результат.

$$\hat{y} = (1 * 5) + (0 * 2) + (1 * 4) - 3 = 6. \quad (3)$$

С помощью функции активации из формулы 2, можно вычислить выходные данные для этого узла: результат равен 1, так как 6 больше 0. Это означает, что в примере стоит идти на серфинг. Если же изменить весовые коэффициенты или пороговое значение, результат вычисления для данной модели может отличаться.

Из примера следует, что нейронная сеть способна принимать решения с возрастающей степенью сложности, в зависимости от выходных данных прошлых решений или слоев. Для иллюстрации математических понятий были использованы персептроны, в то время как в нейронных сетях применяются сигмоидальные нейроны, значения которых могут находиться в диапазоне от 0 до 1. По своему принципу работы нейронные сети схожи с деревьями принятия решений, поэтому в результате передачи данных от одного узла к другому, при x значений от 0 до 1, влияние того или иного изменения отдельной переменной на выходные данные любого узла и, следовательно, выходные данные нейронной сети уменьшается.

В более практических сценариях использования нейронных сетей, например распознавание или классификация изображений, для обучения алгоритма используется контролируемое обучение или маркированные наборы данных. В ходе обучения модели потребуется оценить точность с помощью функции стоимости (среднеквадратическая ошибка). Функция стоимости выражается с помощью формулы:

$$CostFunction = \frac{1}{2m} \sum_{i=1}^m (\hat{y} - y)^2, \quad (4)$$

где i — индекс выборки;
 \hat{y} — прогнозируемое значение;
 y — фактическое значение;
 m — число выборок.

Конечная цель — минимизировать функцию стоимости, чтобы обеспечить корректность для каждого отдельно взятого наблюдения. В процессе корректировки весовых коэффициентов и смещения модель использует функцию стоимости и обучение с подкреплением для достижения точки сходимости или локального минимума. Корректировка весовых коэффициентов происходит с помощью алгоритма градиентного спуска, что позволяет определить стратегию уменьшения количества ошибок (или минимизации функции стоимости). С каждым шагом обучения параметры модели корректируются, пока не будет достигнут минимум. На рисунке 1.4 изображен процесс минимизации функции стоимости.

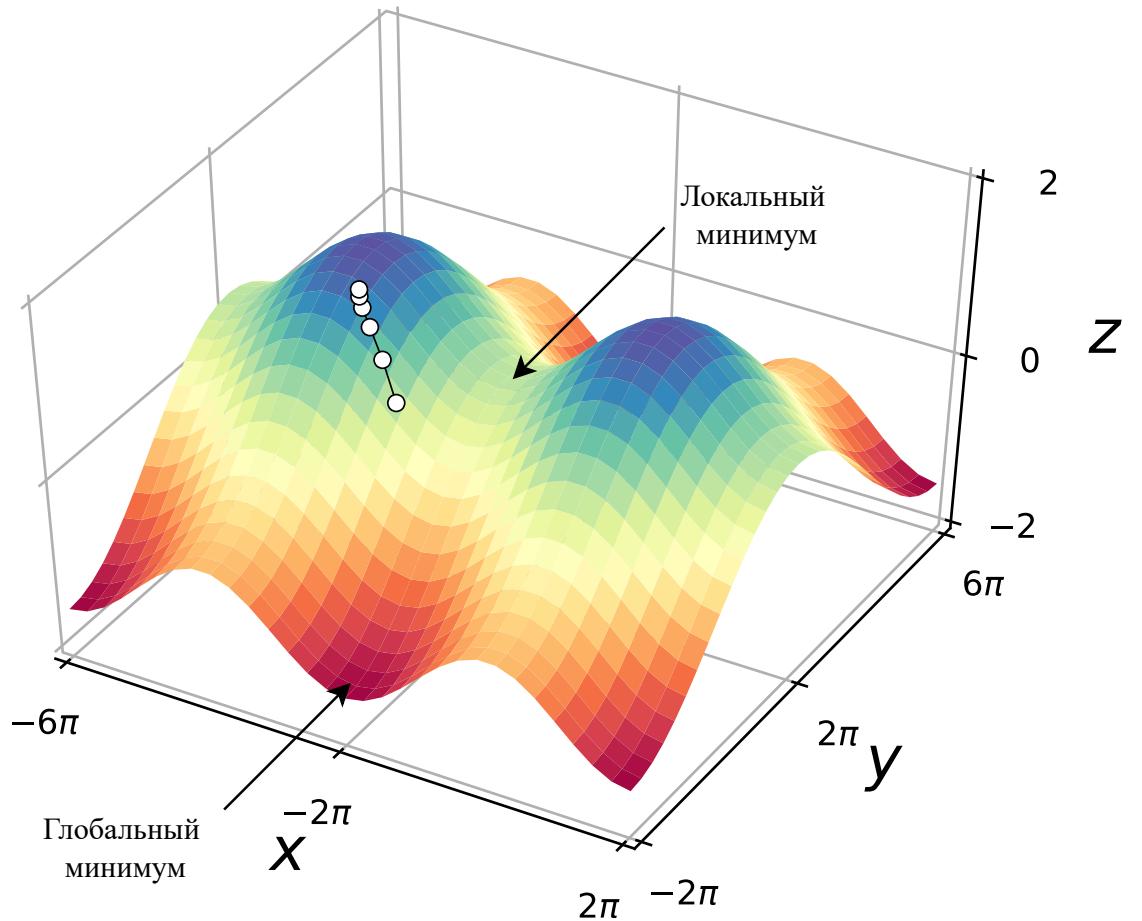


Рис. 1.4: Минимизация функции стоимости

Большинство глубоких нейронных сетей относятся к алгоритмам прямого распространения, т. е. данные передаются только в одном направлении — от входа к выходу. Однако для обучения моделей может также применяться метод обратного распространения ошибки, когда данные передаются в противоположном направлении — от выхода к входу. Метод обратного распространения ошибки позволяет вычислить и объяснить ошибки, связанные с каждым нейроном, что позволяет скорректировать и адаптировать параметры модели соответствующим образом.

1.4.2 Метод обратного распространения ошибки

Метод обратного распространения ошибки (от англ. backpropagation) — метод вычисления градиента, который используется при обновлении весов в нейронной сети.

Рассмотрим простую нейронную сеть без скрытых слоев (рисунок 1.5), с двумя входными вершинами и одной выходной, в которых каждый нейрон использует линейную функцию активации, которая является взвешенной суммой

входных данных.

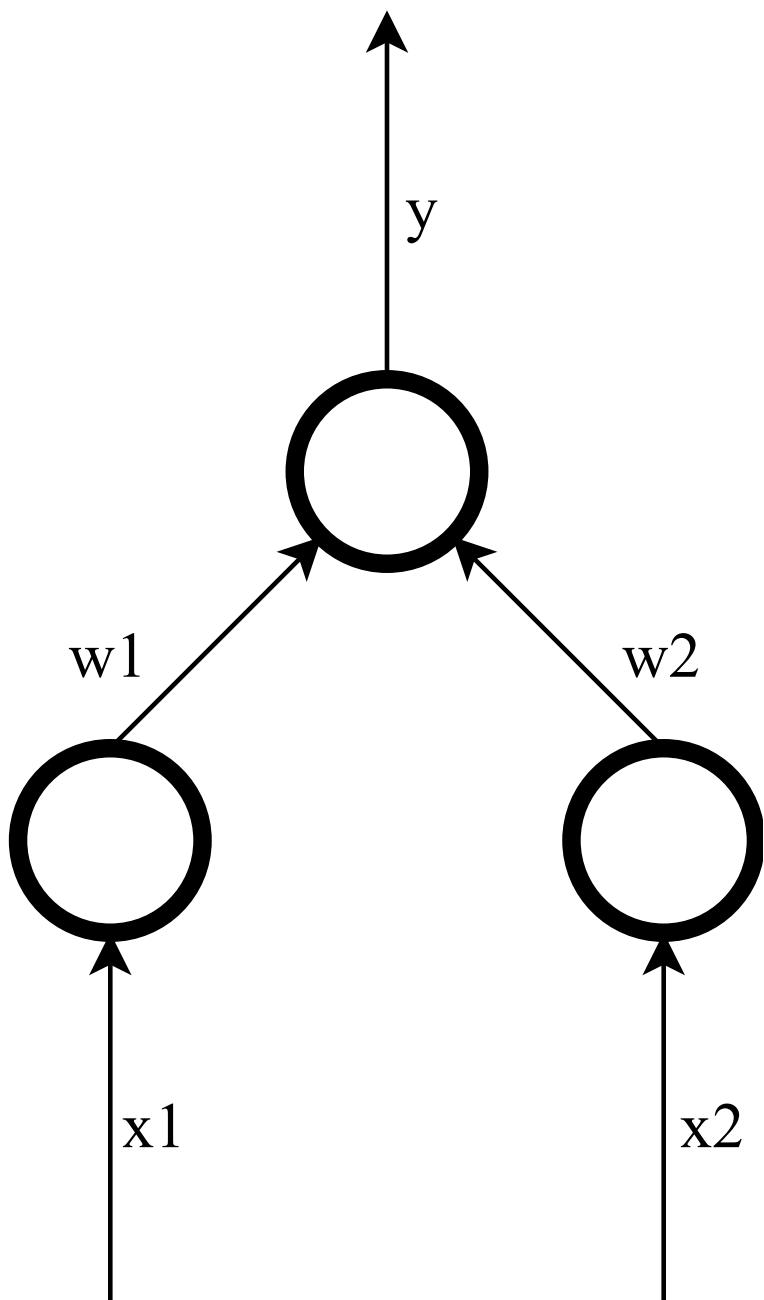


Рис. 1.5: Пример нейронной сети

Изначально веса задаются случайно. Затем, нейрон обучается с помощью тренировочного множества, которое в этом случае состоит из множества троек (x_1, x_2, \hat{y}) , где x_1 и x_2 — это входные данные сети и \hat{y} — правильный ответ. Начальная сеть, приняв на вход x_1 и x_2 , вычислит ответ y , который вероятно отличается от \hat{y} . Общепринятый метод вычисления несоответствия между

ожидааемым \hat{y} и получившимся y ответом — квадратичная функция потерь:

$$E = (\hat{y} - y)^2, \quad (5)$$

где E — ошибка.

В качестве примера, обучим сеть на объекте $(1, 1, 0)$, таким образом, значения x_1 и x_2 равны 1, а \hat{y} равно 0. Построим график зависимости ошибки E от действительного ответа y , его результатом будет парабола (рисунок 1.6).

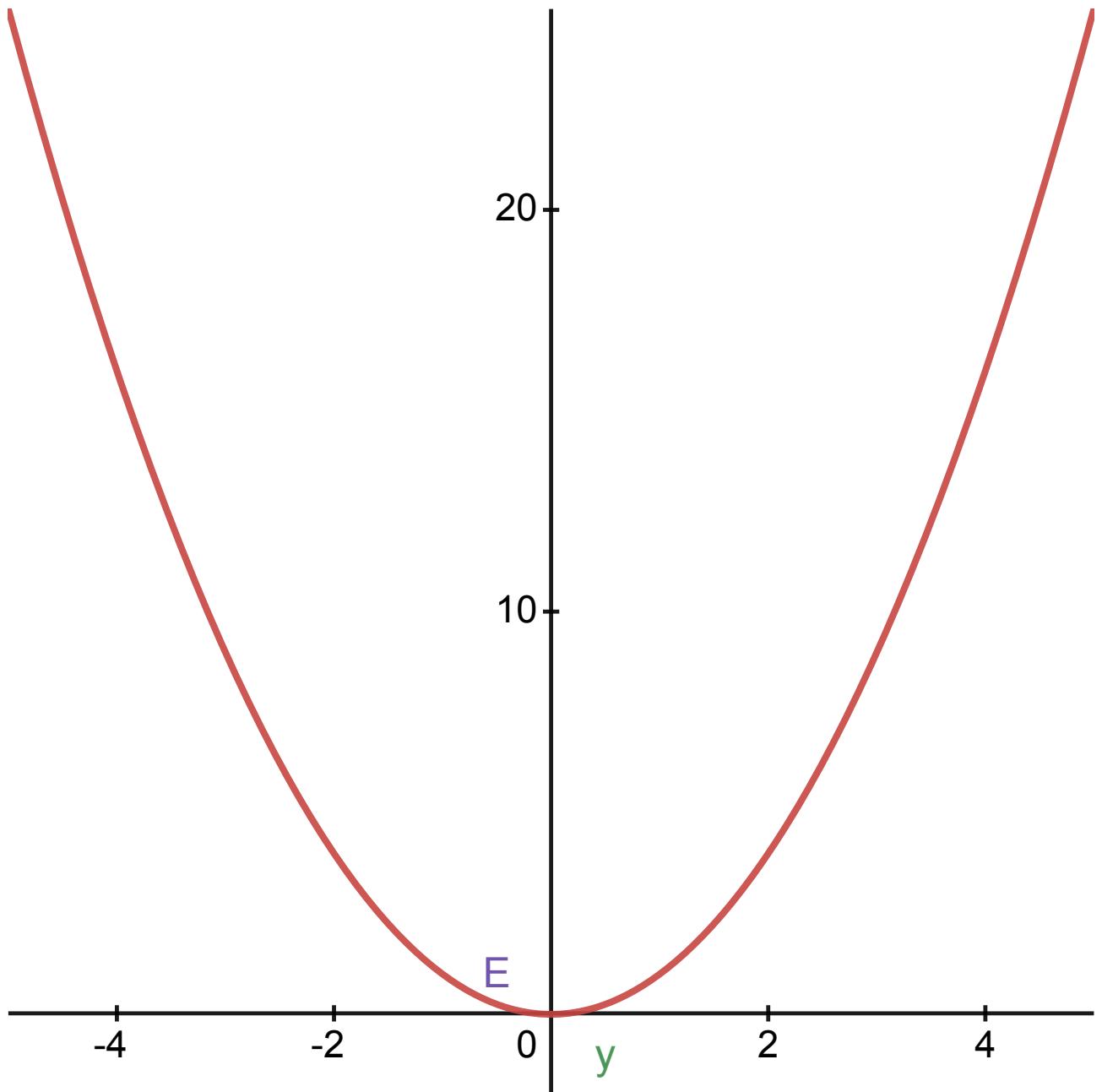


Рис. 1.6: Зависимость ошибки от действительного ответа

Минимум параболы соответствует ответу y , минимизирующему E . Если

тренировочный объект один, минимум касается горизонтальной оси, следовательно ошибка будет нулевая и сеть может выдать ответ y , равный ожидаемому ответу \hat{y} . Следовательно, задача преобразования входных значений в выходные может быть сведена к задаче оптимизации, заключающейся в поиске функции, которая даст минимальную ошибку.

В таком случае, выходное значение нейрона — взвешенная сумма всех его входных значений:

$$\hat{y} = x_1w_1 + x_2w_2, \quad (6)$$

где w_1, w_2 — веса на ребрах, соединяющих входные вершины с выходной.

Следовательно, ошибка зависит от весов ребер, входящих в нейрон. Это и нужно менять в процессе обучения. Распространенный алгоритм для поиска набора весов, минимизирующего ошибку — градиентный спуск, описанный ранее. Метод обратного распространения ошибки используется для вычисления самого крутого направления для спуска.

1.4.3 Виды нейронных сетей

Нейронные сети можно разделить на несколько видов, в зависимости от целевого назначения.

Персепtron — первая нейронная сеть, созданная Фрэнком Розентлаттом в 1958 году. Она содержит один нейрон и представляет собой простейшую форму нейронной сети. На рисунке 1.7 изображен пример такой нейронной сети.

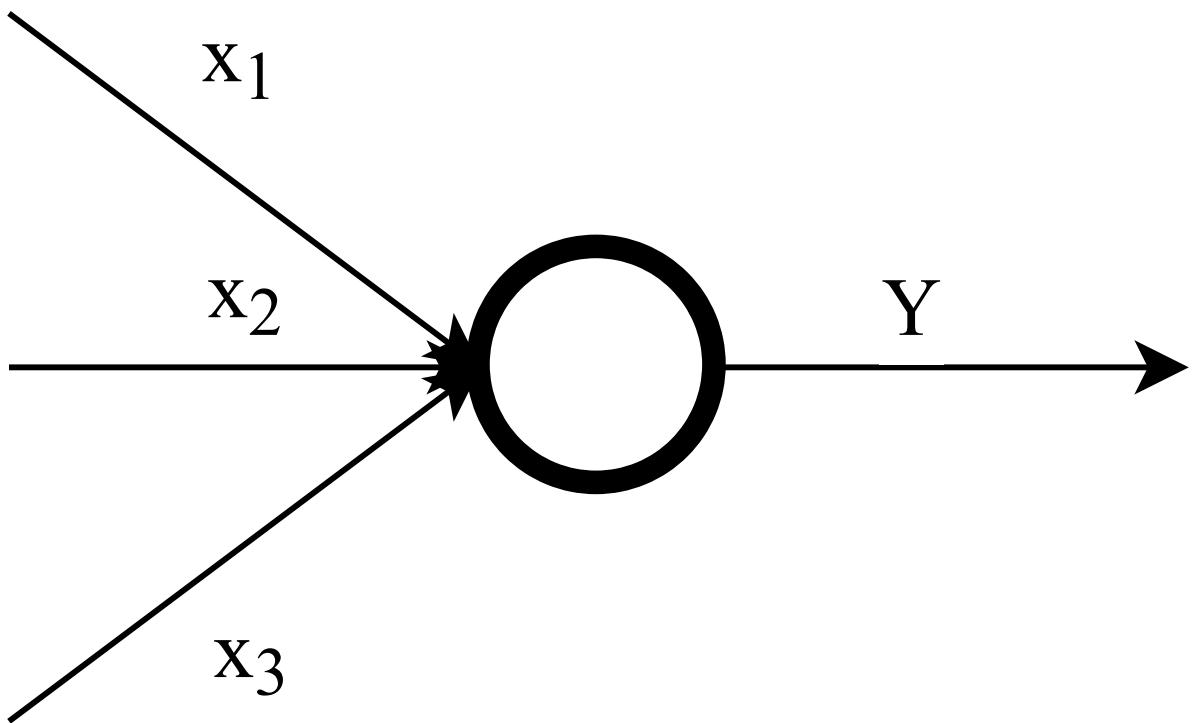


Рис. 1.7: Персептрон

Нейронные сети прямого распространения (многослойные персептроны (MLP)) — нейронные сети, состоящие из слоев входных данных, одного или нескольких скрытых слоев и выходных данных. Данные, поступающие в эти модели, используются для обучения. Эти сети лежат в основе алгоритмов компьютерного зрения, обработки данных на естественном языке и других нейронных сетей.

Рекуррентные нейронные сети (RNN) имеют в своем составе обратные связи. Такие алгоритмы обучения используются в основном для временных рядов данных с целью прогнозирования будущих событий, например стоимости акций на фондовых биржах или объема продаж.

1.4.4 Сравнение нейронных сетей и глубокого обучения

Часто термины «глубокое обучение» и «нейронные сети» могут использоваться как синонимы, что не совсем верно. Стоит отметить, что понятие «глубина» в «глубоком обучении» характеризует лишь количество слоев нейронной сети. Нейронную сеть, в составе которой более трех слоев (включая слой входных данных и слой выходных данных), можно отнести к алгоритмам глубокого обучения. Нейронная сеть с двумя–тремя уровнями считается простой нейронной сетью.

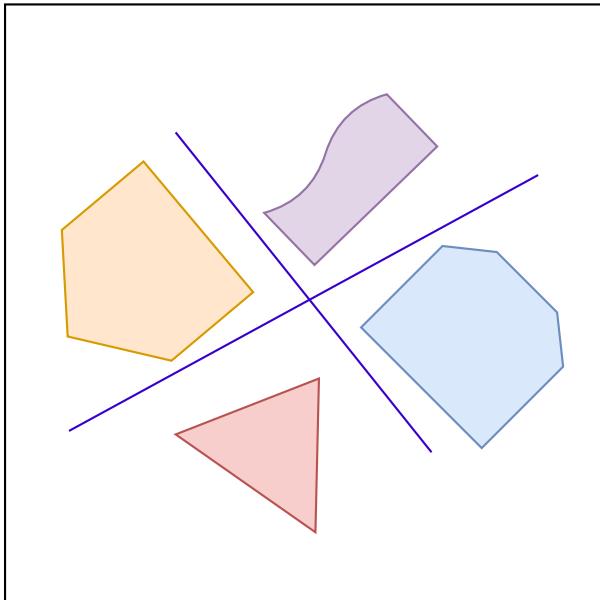
1.4.5 Особенности применения нейронных сетей в качестве классификаторов

Задача классификации для нейронных сетей не является основной. Основной задачей для нейронных сетей является численное предсказание (как было показано на примере с серфингом).

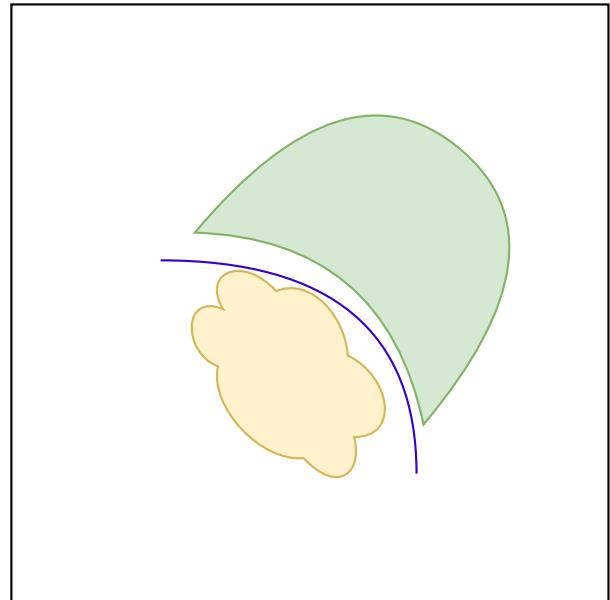
Используя специальные способы представления данных, можно адаптировать нейронные сети для работы с категориальными данными, т.е. получать на вход и формировать на выходе категориальные значения. Для этого категориальные признаки соответствующим образом кодируются с помощью числовых значений.

Нейронные сети имеют ряд преимуществ при использовании в качестве классификаторов. Например:

- нейронные сети являются самообучаемыми моделями, работа которых практически не требует вмешательства пользователя;
- нейронные сети являются универсальными аппроксиматорами, позволяющими аппроксимировать любую непрерывную функцию с достаточной точностью;
- нейронные сети являются нелинейными моделями, что позволяет эффективно решать задачи классификации даже при отсутствии линейной разделимости классов. На рисунке 1.8 приведен пример линейной разделимости и неразделимости классов.



Линейно разделимые классы



Линейно неразделимые классы

Рис. 1.8: Линейная разделимость классов

Следует отметить, что каких–либо специальных нейросетевых архитектур для классификации не существует. Наиболее часто используемой для классификации архитектурой нейронных сетей являются сети прямого распространения, на входные нейроны которых подаются значения признаков классифицируемого объекта, а на выходе формируется метка или числовой код класса.

Последующие слои, таким образом, разделяют объекты на классы в пространстве признаков более высокой размерности, чем исходное. Например, если размерность вектора признаков исходных данных равна 4, и скрытый слой содержит 6 нейронов, то выходной слой осуществит разбиение объектов на классы в 6–мерном пространстве.

Это позволяет сделать процесс более эффективным: правильно подобрав конфигурацию и параметры нейронной сети, можно получить хорошие результаты классификации даже в тех случаях, когда классификаторы других типов, работающие только в размерности обучающих данных, не обеспечивают приемлемых результатов. Недостатком является то, что конфигурация сети, наилучшим образом аппроксимирующая функцию разделения классов в пространстве признаков, заранее неизвестна. Поэтому приходится подбирать её экспериментально, либо использовать опыт аналогичных решений.

Если распределение классов таково, что для их разделения требуется сложная функция, размерность нейронной сети может оказаться неприемлемо

большой. В этом случае проблему можно решить с помощью специальной предобработки исходных данных.

1.4.6 Машина опорных векторов

SVM (от англ. Support vector machine) — метод машинного обучения, применяемый в задачах классификации, основанный на построении гиперплоскости и ее анализе [3].

Главная цель SVM как классификатора — найти уравнение разделяющей гиперплоскости $w_1x_1 + w_2x_2 + \dots + w_nx_n + w_0 = 0$ в пространстве R^n , которая бы разделила два класса неким оптимальным образом. Общий вид преобразования F объекта x в метку класса Y представлен в формуле 7:

$$F(x) = \text{sign}(w^T x - b). \quad (7)$$

При рассмотрении обозначим $w = (w_1, w_2, \dots, w_n)$, $b = -w_0$. После настройки весов алгоритма w и b , все объекты, попадающие по одну сторону от построенной гиперплоскости, будут предсказываться как первый класс, а объекты, попадающие по другую сторону — второй класс.

Внутри функции $\text{sign}()$ стоит линейная комбинация признаков объекта с весами алгоритма, именно поэтому SVM относится к линейным алгоритмам. Разделяющую гиперплоскость можно построить разными способами, но в SVM веса w и b настраиваются таким образом, чтобы объекты классов лежали как можно дальше от разделяющей гиперплоскости. Другими словами, алгоритм максимизирует зазор (англ. margin) между гиперплоскостью и объектами классов, которые расположены ближе всего к ней. Такие объекты и называют опорными векторами.

На рисунке 1.9 представлен принцип работы машины опорных векторов.

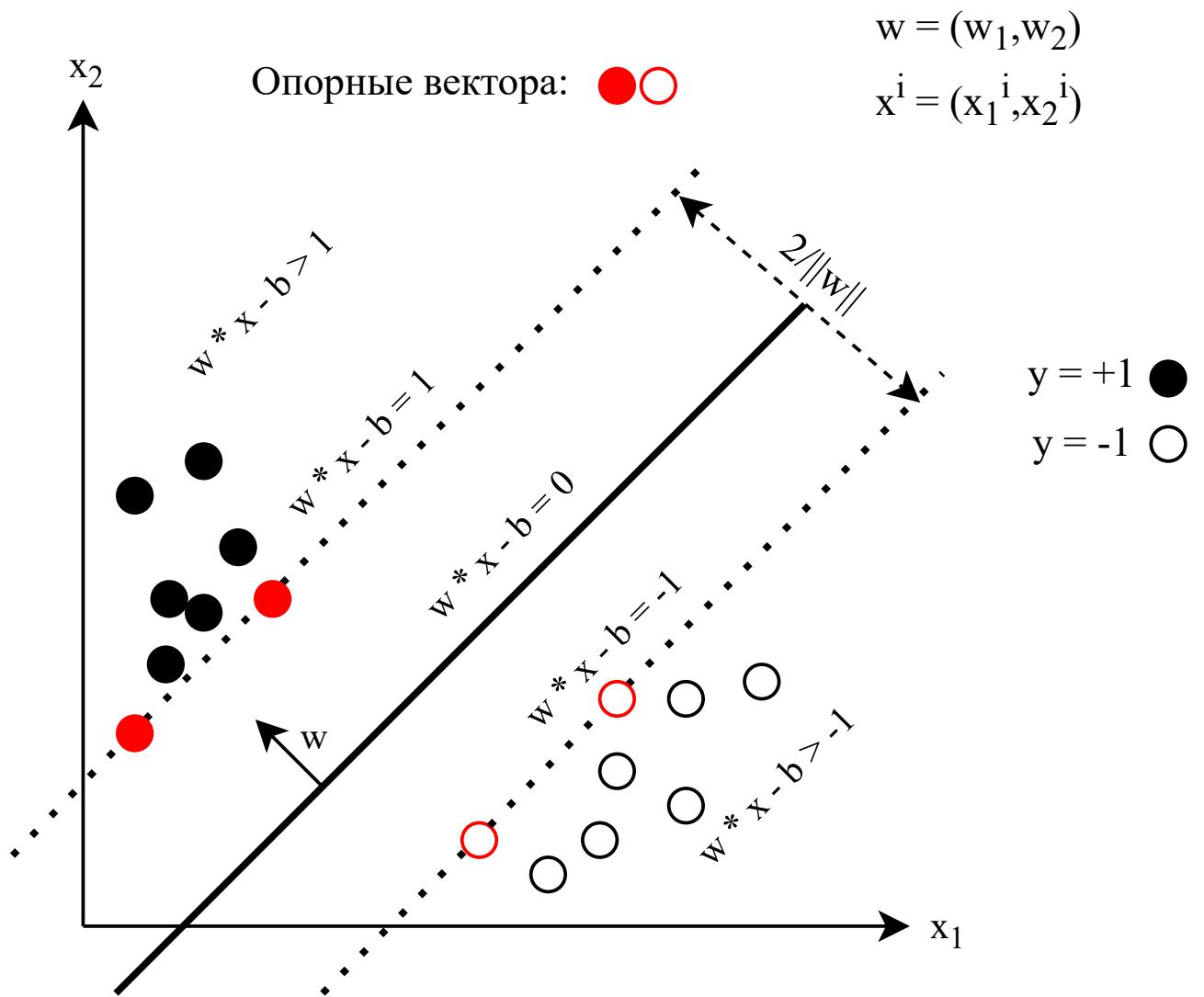


Рис. 1.9: Визуализация работы SVM

Чтобы разделяющая гиперплоскость как можно дальше отстояла от точек выборки, ширина полосы должна быть максимальной. Вектор w — вектор нормали к разделяющей гиперплоскости. Здесь и далее скалярное произведение двух векторов будет обозначаться как $\langle a, b \rangle$ или $a^T b$. Проекция вектора w , концами которого являются опорные вектора разных классов, будет показывать ширину разделяющей полосы.

На рисунке 1.10 представлен вывод вывод правил настройки весов.

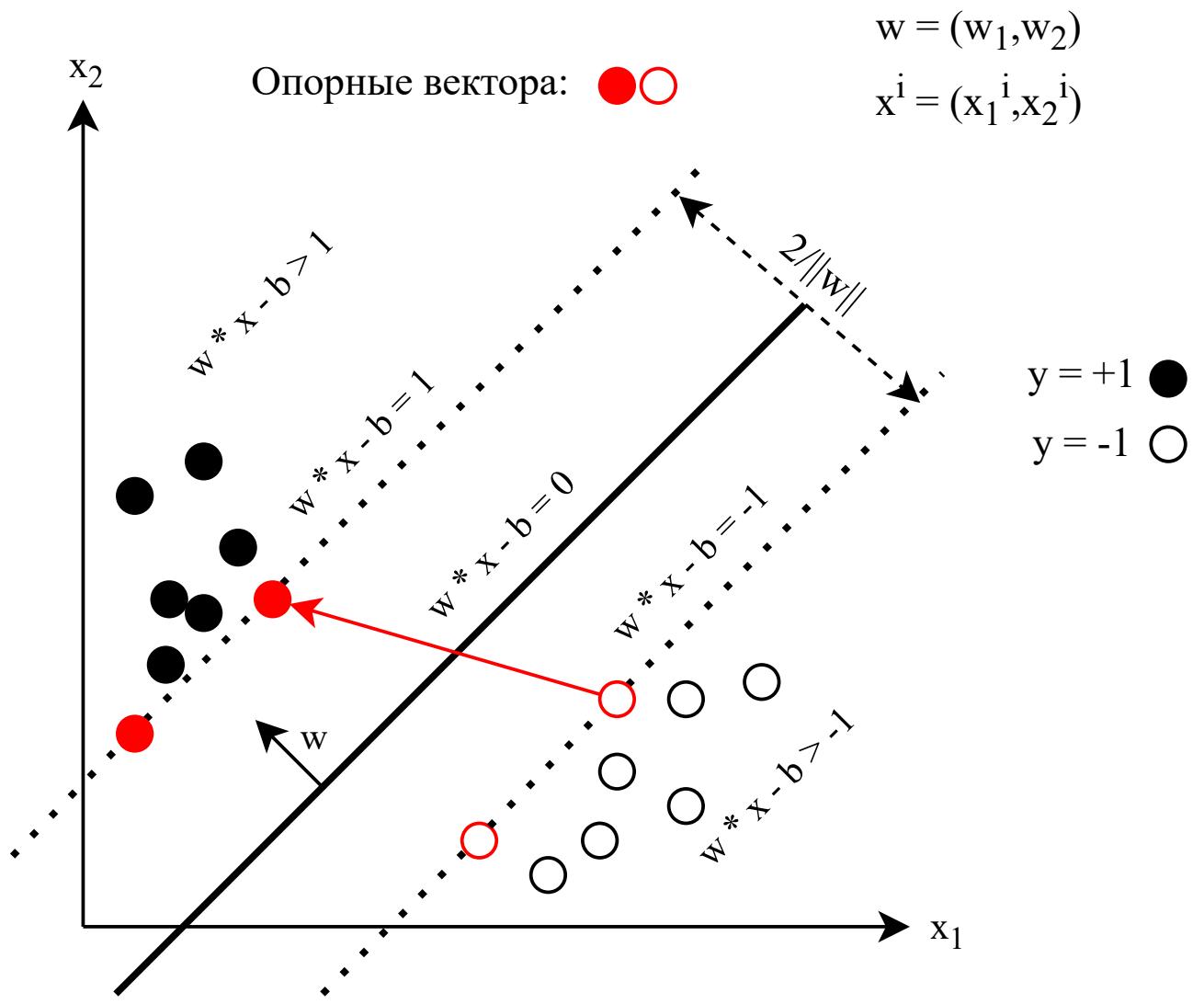


Рис. 1.10: Вывод правил настройки весов

В формулах 8 — 11 представлен вывод правил настройки весов.

$$\langle (x_+ - x_-), \frac{w}{\|w\|} \rangle = \frac{\langle x_+, w \rangle - \langle x_-, w \rangle}{\|w\|} = \frac{(b+1) - (b-1)}{\|w\|} = \frac{2}{\|w\|}, \quad (8)$$

$$\frac{2}{\|w\|} \rightarrow \max, \quad (9)$$

$$\|w\| \rightarrow \min, \quad (10)$$

$$\frac{w^T w}{2} \rightarrow \min. \quad (11)$$

Отступом (англ. margin) объекта x от границы классов называется величина $M = y(w^T x - b)$. Алгоритм допускает ошибку на объекте тогда и только тогда, когда отступ M отрицателен (когда y и $w^T x - b$ разных знаков). Если $M(0, 1)$, то объект попадает внутрь разделяющей полосы. Если $M > 1$, то объект x классифицируется правильно, и находится на некотором удалении от разделяющей полосы. Алгоритм будет правильно классифицировать объекты, если выполняется условие, представленное в формуле 13.

$$y(w^T x - b) \geq 1. \quad (12)$$

1.5 Методы и технологии распознавания образов

Для решения задачи распознавания образов с помощью нейронных сетей применяются такие способы, как:

- R-CNN (от англ. Region-based Convolutional Network);
- Fast R-CNN;
- Faster R-CNN.

1.5.1 R-CNN

R-CNN имеет следующую схему работы:

- 1) При помощи *selective search* (с англ. избирательный поиск) [17] на изображении выделяются регионы, которые предположительно содержат объект (кости лица на КТ или МРТ снимке).
- 2) Регион-претендент при помощи аффинного преобразования отображается в квадрат 227×227 .
- 3) Полученный прямоугольник подается на вход сети [18], имеющую структуру, представленную на рисунке 1.11.

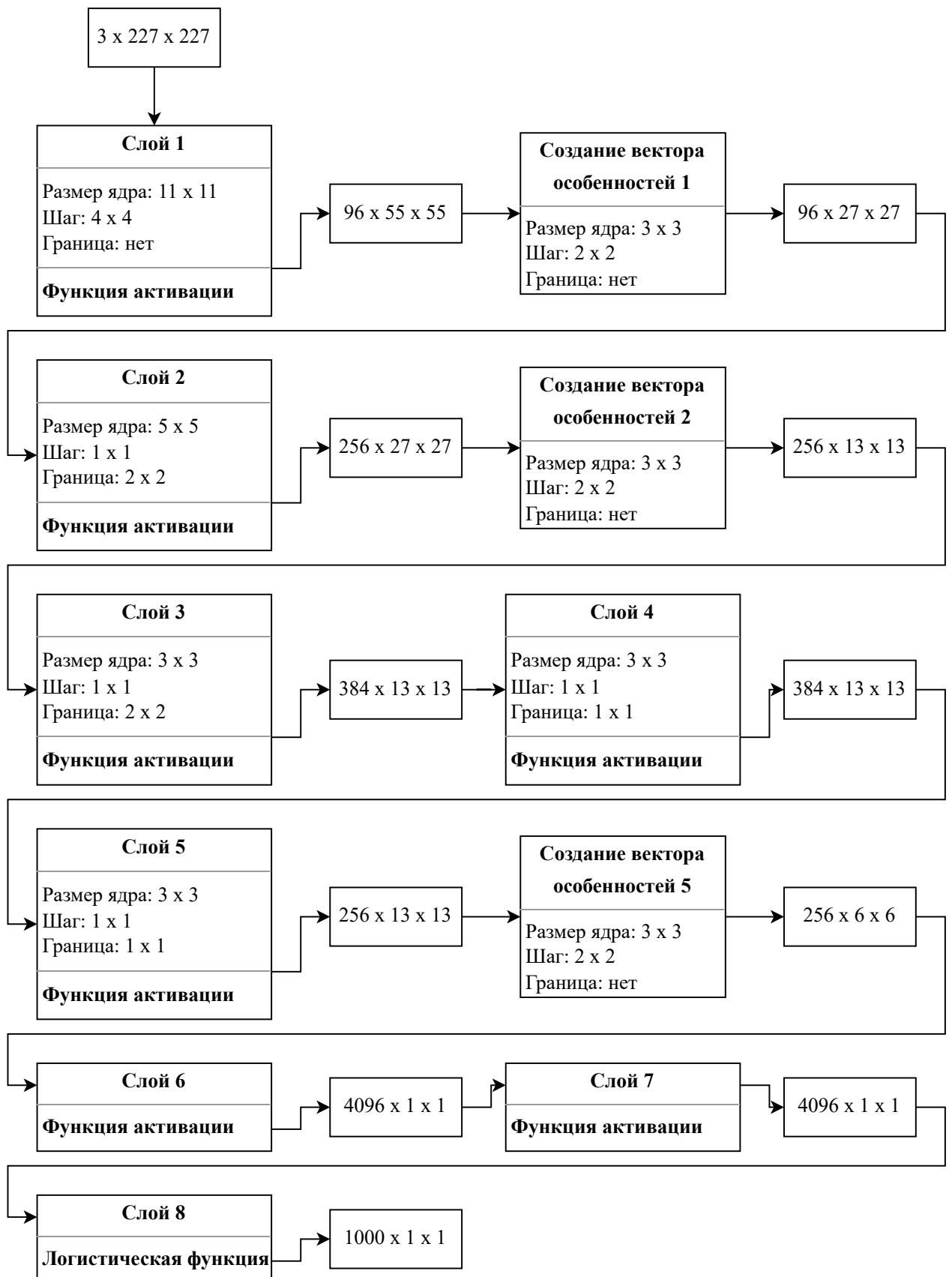


Рис. 1.11: Принцип работы R-CNN

- 4) С седьмого слоя сети снимается вектор размерности 4096. Этот вектор используется в машине опорных векторов, натренированной для

каждого класса, для классификации.

- 5) Вектор подается в линейный регрессор (логистическая функция), соответствующий классу объекта для восстановления позиции объекта.

1.5.2 Fast R-CNN

R-CNN имеет несколько недостатков, в основном связанных с высокими временными затратами:

- 1) Требуется натренировать сверточную нейронную сеть (в два этапа: тренировка без точной разметки и оптимизация для 20 классов).
- 2) Требуется натренировать набор машин опорных векторов по одной для каждого класса. При этом на вход машина опорных векторов принимает вектора особенностей, полученных от натренированной сверточной сети. Эти вектора нужно изначально вычислить на наборе данных, используемых для тренировки.
- 3) Требуется натренировать линейные регрессоры (по одному для каждого класса) для восстановления позиций объекта.
- 4) Процесс детектирования и классификации требует существенного времени (до 49 секунд на одно изображение) [19].

Существенные временные затраты на этапе детектирования сильно снижают практическую пользу R-CNN. Причина, из-за которой детектирование работает так медленно, заключается в том, что при помощи избирательного поиска генерируется примерно 2000 претендентов на изображении, и каждый претендент должен быть пропущен через сверточную сеть, чтобы вычислить для него вектор особенностей. Процесс этот достаточно затратный, особенно для сетей с большим числом сверточных слоев. В Fast R-CNN предлагают подавать на вход сети полное изображение, но при этом последний слой заменить на слой типа *RoI pooling* (Region of Interest Pooling) [19].

RoI pooling слой принимает на вход карту особенностей, полученную от последнего сверточного слоя нейронной сети, и RoI претендента (в координатах изображения). RoI преобразуется из координат изображения в координаты на карте особенностей и на полученный прямоугольник накладывается сетка $W \times H$ с наперед заданными размерами (например, $W = H = 7$) [19]. *RoI pooling* слой преобразует вектор особенностей произвольного прямоугольника из исходного изображения в вектор особенностей фиксированной размерности.

После RoI pooling слоя данные через два полно связанных слоя подаются параллельно на слой для оценки принадлежности претендента одному из классов объектов и слой, реализующий регрессию для уточнения границы объекта.

Итоговый принцип работы Fast R-CNN представлен на рисунке 1.12.

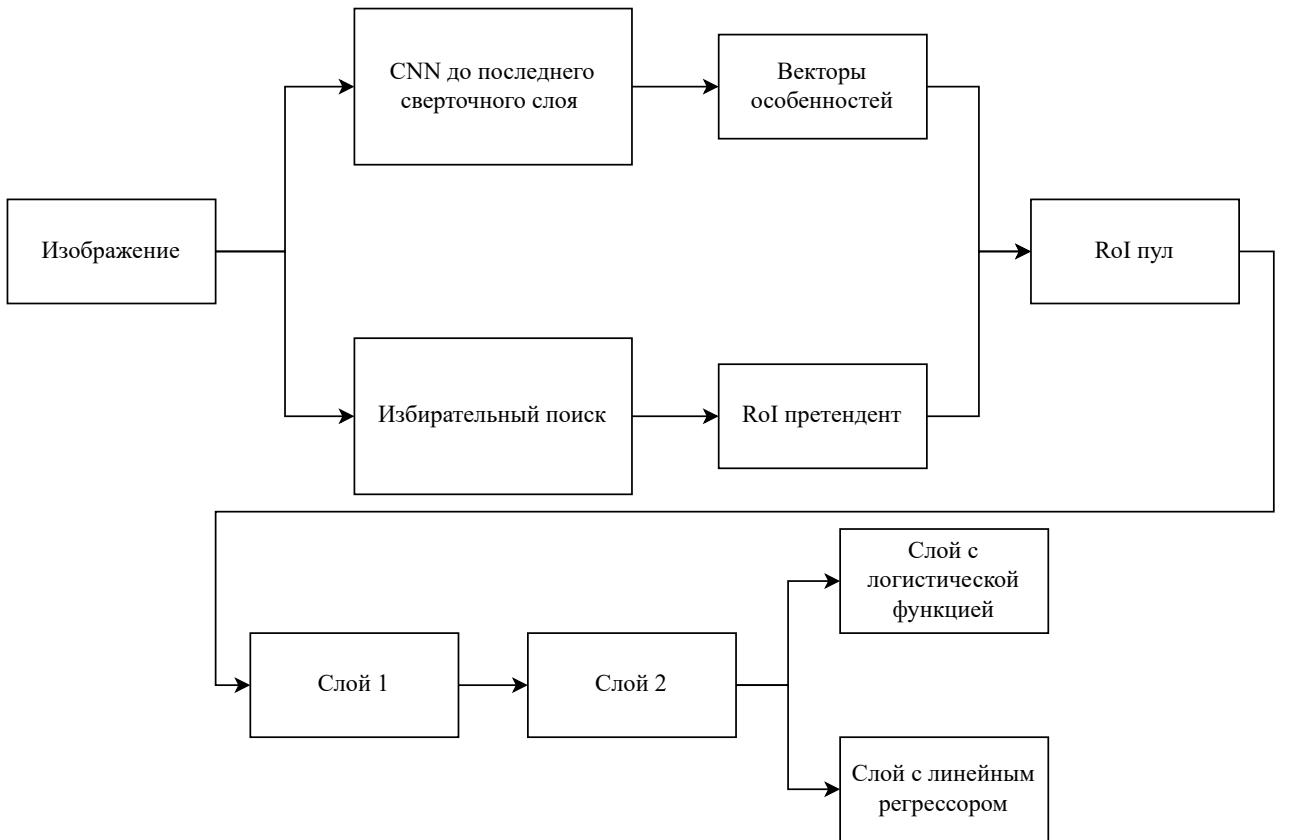


Рис. 1.12: Принцип работы Fast R-CNN

В отличии от R-CNN, где карта особенностей генерировалась для каждого претендента на изображении, в Fast R-CNN генерируется карта особенностей для всего изображения, а затем при помощи специального слоя из нее вычисляются карты для каждого из претендентов, что позволяет существенно сократить время вывода.

1.5.3 Faster R-CNN

В качестве улучшения Fast R-CNN предлагается замена процедуры генерации претендентов избирательным поиском на нейронную сеть, которая использует имеющуюся карту особенностей (например, VGG16) [20]. Для изображения размера $W_1 \times H_1$ на выходе последнего сверточного слоя сеть VGG16 выдает карту особенностей с размерами $W_1/16 \times H_1/16$, вектор особенностей для каждой точки будет иметь размерность 512. При этом вектор особенностей в точке (x_f, y_f) вносят вклад точки изображения, лежащие внутри квадрата с

центром в $(16x_f, 16y_f)$ и размера 196×196 .

Для каждой точки карты особенностей (x_f, y_f) проверяется k претендентов разных размеров на изображении в регионах с центром в $(16x_f, 16y_f)$. Рассматривают 9 претендентов, варьируя три масштаба и три отношения сторон ($1 \div 1, 1 \div 2, 2 \div 1$). Для решения задачи используется *Region Proposal Network (RPN)* (от англ. сеть для предложения регионов) [21].

Итоговый принцип работы Faster R–CNN представлен на рисунке 1.13.

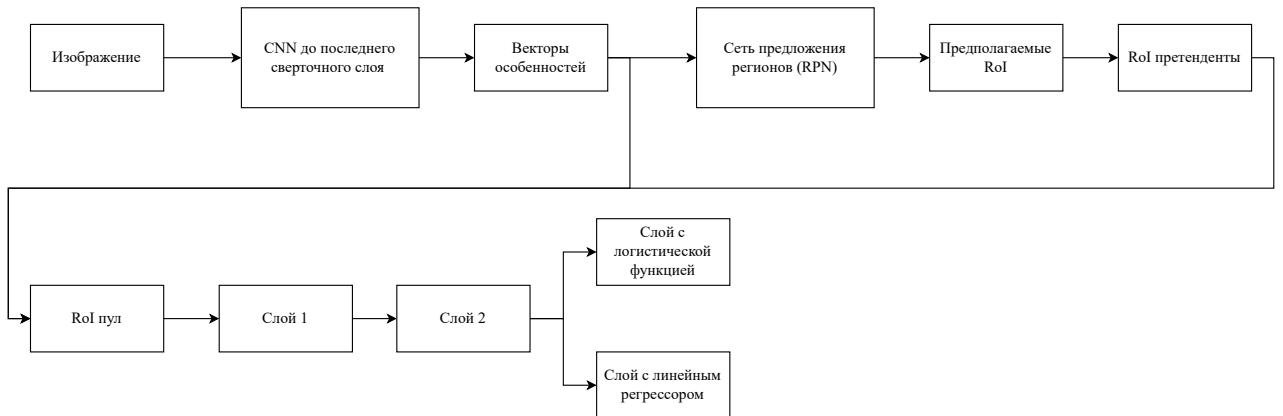


Рис. 1.13: Принцип работы Faster R–CNN

Замена генерации претендентов избирательным поиском на нейронную сеть позволяет повысить быстродействие распознавания образов на изображений до десяти раз [20].

1.5.4 Сравнение рассмотренных методов

Сравнительная характеристика рассмотренных методов представлена в таблице 1.

Таблица 1: Сравнение методов распознавания образов

Критерий	R–CNN	Fast R–CNN	Faster R–CNN
Время распознавания, сек*	~49	2.32	0.2
Время вычисления	Высокое	Высокое	Низкое
mAP** на датасете Pascal VOC 2007, %	58.5	66.9	69.9
mAP** на датасете Pascal VOC 2012, %	53.3	65.7	67.0

* — меньше — лучше.

** — mAP (от англ. mean average precision) — метрика измерения точности обнаруживаемых объектов. Больше — лучше.

1.6 Методы и технологии выделения объектов на изображении

Для решения задачи распознавания образов с помощью нейронных сетей применяются такие способы, как:

- Mask R–CNN;
- UNet.

1.6.1 Mask R–CNN

Mask R–CNN развивает архитектуру Faster R–CNN путём добавления ещё одной ветки, которая предсказывает положение маски, покрывающей найденный объект, и, таким образом решает уже задачу сегментации экземпляров. Мaska представляет собой просто прямоугольную матрицу, в которой 1 на некоторой позиции означает принадлежность соответствующего пикселя объекту заданного класса, 0 — что пиксель объекту не принадлежит.

Архитектура Mask R–CNN условно разделяется на CNN–сеть вычисления признаков изображения, называемую *backbone*, и *head* — объединение частей, отвечающих за предсказание охватывающей рамки, классификацию объекта и определение его маски. Функция потери для них общая и включает три компонента:

$$L = L_{cls} + L_{box} + L_{mask}, \quad (13)$$

где L_{cls} — функция потери для этапа классификации;

L_{box} — функция потери для этапа выделения охватывающей рамки;

L_{mask} — функция потери для этапа определения маски;

Маски предсказываются отдельно для каждого класса, без предварительного знания, что изображено в регионе, и потом выбирается маска класса, победившего в независимом классификаторе. Утверждается, что такой подход более эффективен, чем опирающийся на априорное знание класса.

Одна из основных модификаций, возникших из–за необходимости предсказывать маску — изменение процедуры RoIPool (вычисляющей матрицу признаков для региона–кандидата) на процедуру RoIAlign. Дело в том, что карта признаков, полученная из CNN, имеет меньший размер, чем исходное изображение, и регион, охватывающий на изображении целочисленное количество пикселей, не получается отобразить в пропорциональный регион карты с целочисленным количеством признаков (рисунок 1.14).

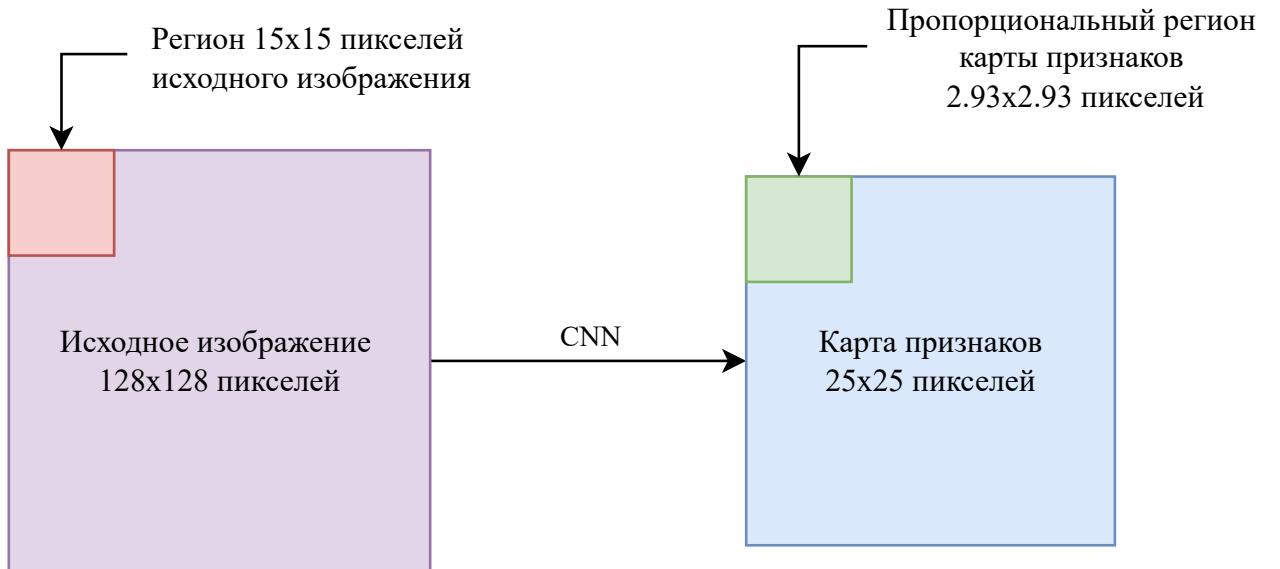


Рис. 1.14: Проблема отображения региона изображения в регион карты

В RoIPool проблема решалась округлением дробных значений до целых. Такой подход нормально работает при выделении охватывающей рамки, но вычисленная на основе таких данных маска получается неточной.

В RoIAlign не используется округление, все числа остаются действительными, а для вычисления значений признаков используется билинейная интерполяция по четырём ближайшим целочисленным точкам.

1.6.2 UNet

UNet считается одной из стандартных архитектур CNN для задач сегментации изображений, когда нужно не только определить класс изображения целиком, но и сегментировать его области по классу (сегментация экземпляров). Архитектура сети состоит из стягивающего пути для захвата контекста и симметричного расширяющегося пути, который позволяет осуществить точную локализацию [22].

На рисунке 1.15 представлена архитектура UNet.

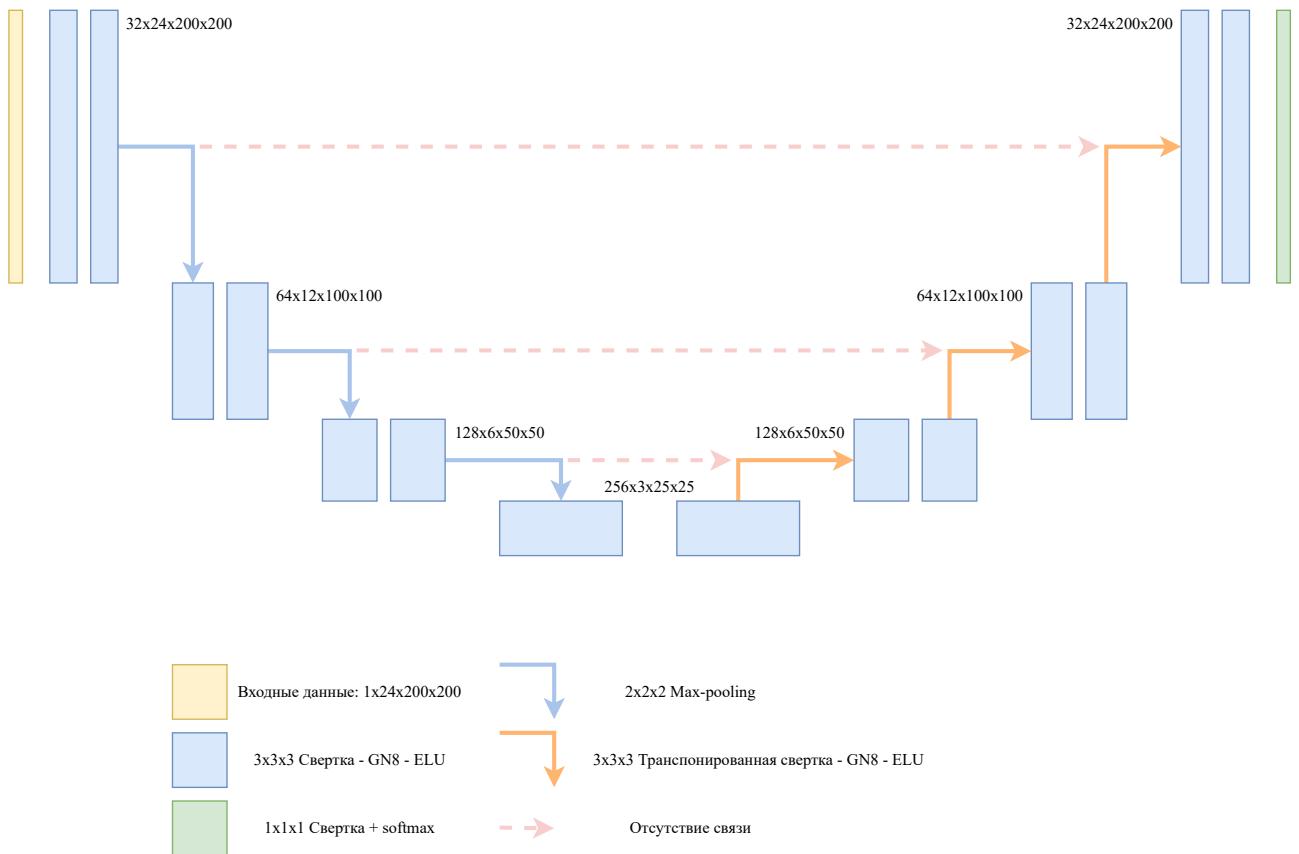


Рис. 1.15: Принцип работы UNet

Сеть обучается сквозным способом на небольшом количестве изображений. Сегментация изображения размером 512×512 пикселей занимает менее секунды на современном графическом процессоре (например, Nvidia Tesla).

Модель UNet состоит из сужающегося пути (слева) и расширяющегося пути (справа). Сужающийся путь состоит из повторного применения двух сверток 3×3 , за которыми следует функция активации ReLU (от англ. Rectified Linear Unit) и операция максимального объединения для понижения разрешения.

На каждом этапе понижающей дискретизации каналы свойств удваиваются. Каждый шаг в расширяющемся пути состоит из операции повышающей дискретизации карты свойств, за которой следуют:

- свертка 2×2 , которая уменьшает количество каналов свойств;
- объединение с обрезанной картой свойств из сужающегося пути;
- две 3×3 свертки, за которыми следует функция активации ReLU.

Обрезка нужна из-за потери граничных пикселей при каждой свертке.

На последнем слое используется свертка 1×1 для сопоставления каждого 64-компонентного вектора свойств с желаемым количеством классов. Всего

сеть содержит 23 сверточных слоя.

1.6.3 Сравнение рассмотренных методов

Согласно исследованиям [23] при сравнении Mask R–CNN и UNet Mask R–CNN показала большую точность в сравнении с UNet (0.76 ± 0.03 против 0.43 ± 0.12). Несмотря на это, UNet применяется в большей части биомедицинских задач. Это означает, что существует большее число моделей и возможных подходов при тренировке модели.

1.7 Выбор данных для обучения модели

Данные для обучения модели должны соответствовать нескольким важным критериям, на основе которых можно сделать вывод, что обучение на выходе даст корректную модель.

Среди таких факторов можно выделить:

- Полнота. В выборке должны быть представлены снимки различных состояний челюстно–лицевого сустава. Например полностью здоровая челюсть или отсутствие каких–либо из зубов.
- Количество. Снимков в выборке должно быть достаточно для того, чтобы модель на одном и том же изображении не давала разных результатов.
- Единообразность. Снимки в выборке должны быть схожего формата для исключения неоднозначных результатов.

Стоит отметить особенности томографических снимков: снимки могут быть сделаны с разных ракурсов (анфас, в профиль), возрастная категория пациента может существенно влиять на топологию снимка (количество зубов у младенцев и у взрослых в зубочелюстной области различно).

При конечном выборе датасета стоит принять во внимание вышеперечисленные особенности и скорректировать процесс обучения модели для получения удовлетворяющих задаче результатов.

Вывод

Были рассмотрены медицинские методы томографии.

Была рассмотрена задача распознавания образов (классификации) и выделения объектов на изображении (сегментации), виды классификаторов и их применимость при распознавании образов.

Было дано определение понятия нейронной сети, описаны виды нейронных сетей и принцип их работы.

Были рассмотрены особенности применения нейронных сетей в качестве классификаторов.

Были рассмотрены и проанализированы технологии (R–CNN, Fast R–CNN и Faster R–CNN) для распознавания образов при помощи нейронных сетей и для выделения объектов на изображений (Mask R–CNN, UNet), приведены преимущества и недостатки рассмотренных технологий.

В качестве базовых технологий для разрабатываемого метода выбраны модель нейронной сети UNet, так как она широко используется в биомедицинских задачах и дает приемлемый результат по затраченному времени как при обучении, так и при сегментации изображения. Для постклассификации распознанных костей выбрана технология машины опорных векторов. При полученной сегментации можно составить векторы свойств сегментированных изображений и применить метод опорных векторов для дальнейшей классификации.

2. Конструкторский раздел

В данном разделе описываются требования к разрабатываемому методу и программному комплексу, реализующему интерфейс для метода. Рассматривается архитектура метода и структура программного комплекса.

2.1 Требования к разрабатываемому методу

Метод распознавания челюстно–лицевых костей черепа по томографическим снимкам головы человека (далее — метод распознавания) должен:

- Принимать на вход изображения в форматах PNG, JPG, JPEG.
- Производить семантическую сегментацию зубочелюстных костей и костей челюстно–лицевого сустава.
- Производить сегментацию экземпляров распознанных зубов.

2.2 Требования к разрабатываемому программному комплексу

Программный комплекс, реализующий интерфейс для разработанного метода, должен предоставлять:

- Возможность загрузки изображений через графический интерфейс.
- Возможность выбора этапов работы: сегментация зубочелюстного сустава или челюстно–лицевого, семантическая сегментация или сегментация экземпляров.
- Создание итогового снимка с выделенными сегментированными зонами.

2.3 Проектирование метода распознавания

В качестве основы для метода распознавания будет применяться нейронная сеть UNet.

UNet при обучении использует готовые маски, поэтому необходимо будет произвести предварительную разметку данных для выделения масок костей зубочелюстного и черепно–лицевого суставов.

После обучения при работе модели UNet составляет маску области, по которой происходит дальнейшее выделение объектов. Для сегментации зубов по экземплярам стоит на данном этапе составить вектора особенностей для каждого зуба, что поможет произвести классификацию по типу.

2.3.1 Физиология зубочелюстного сустава

Существует четыре типа зубов:

- Резцы — служат для захвата и откусывания пищи.
- Клыки — необходимы для разрывания и удерживания пищи.

- Премоляры — раздавливают и передавливают пищу.
- Моляры — измельчают и перемалывают пищу.

На рисунке 3.4 представлена классификация зубов.

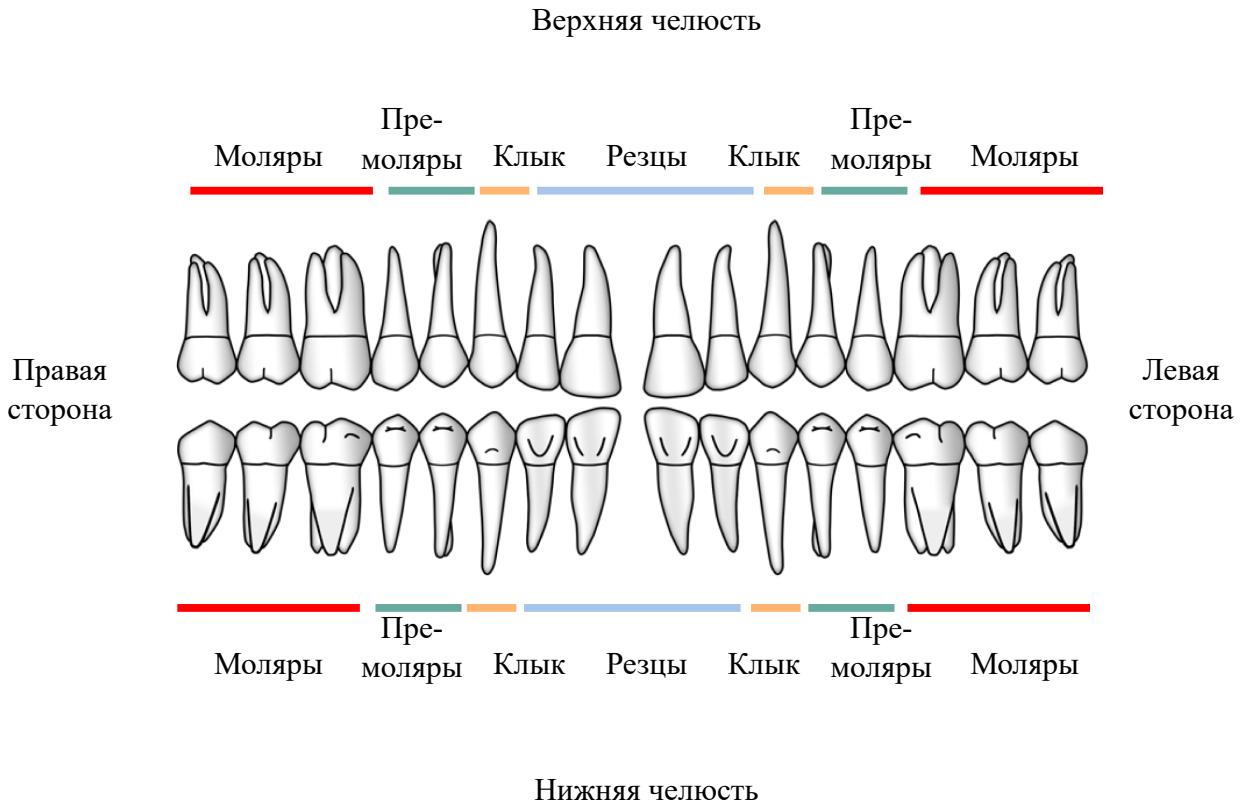


Рис. 2.1: Классификация зубов

Зубной состав сустава можно представить в виде формулы:

$$I \frac{2}{2} C \frac{1}{1} P \frac{2}{2} M \frac{3}{3} = 32, \quad (14)$$

где I — резцы;

C — клыки;

P — премоляры;

M — моляры.

2.3.2 Структура вектора особенностей

Для классификации распознанных зубов можно выделить ряд особенностей, по которым можно определить принадлежность зуба к тому или иному типу.

Среди таких особенностей можно выделить:

- размер (площадь в пикселях);

- расположение (порядковый номер если считать слева направо);

На основе анализа особенностей будет составлена модель, использование которой в машине опорных векторов позволит классифицировать выделенные зубы по типу.

2.3.3 IDEF0–диаграмма

На рисунках 2.2 и 2.3 представлена диаграмма IDEF0 метода распознавания.

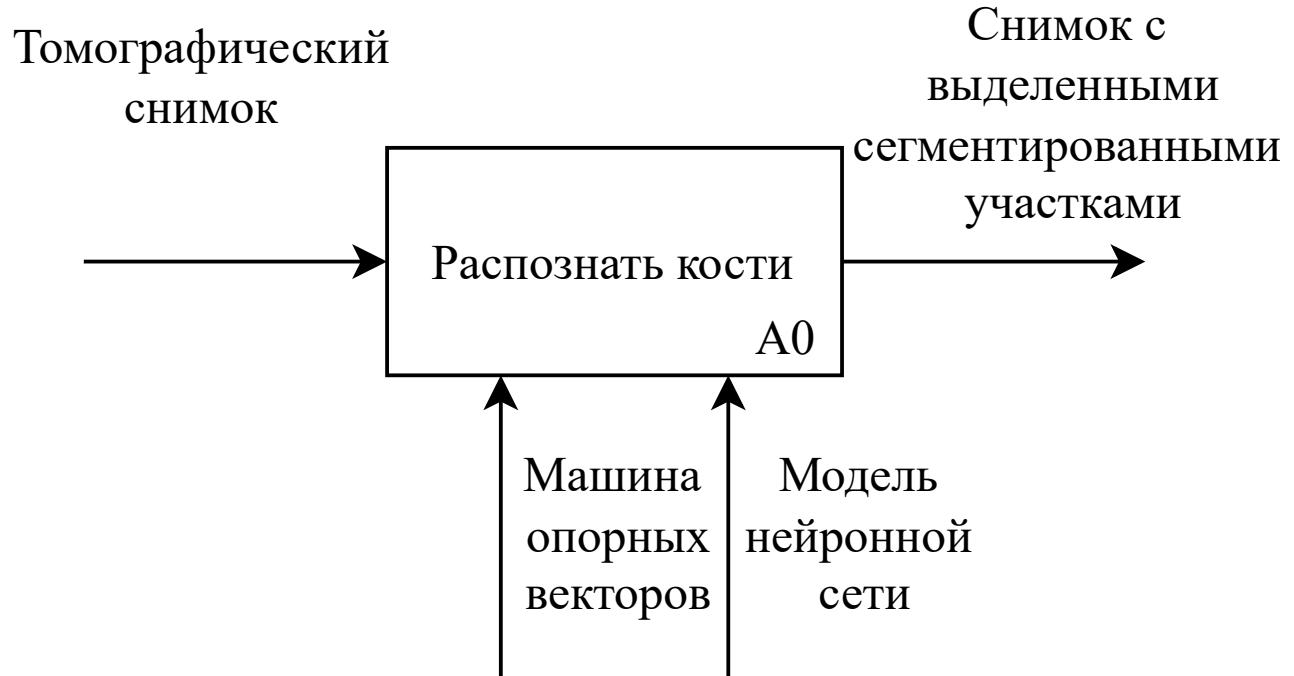


Рис. 2.2: IDEF0–диаграмма уровня A0

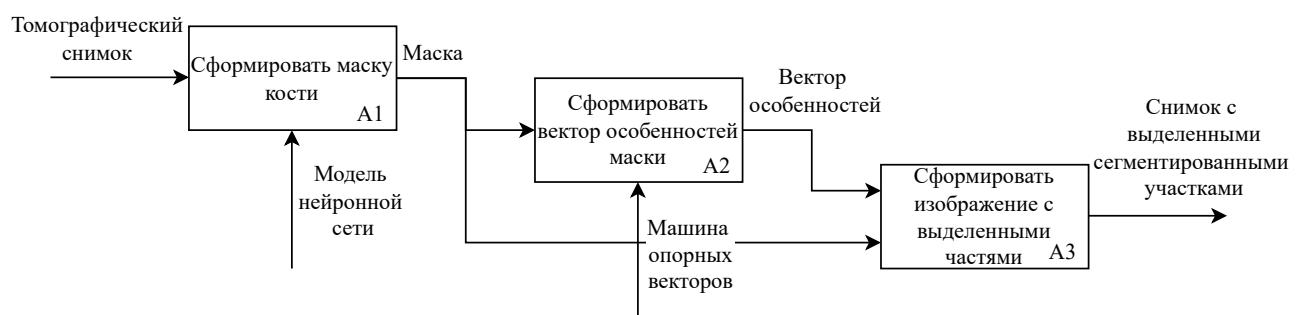


Рис. 2.3: IDEF0–диаграмма уровня A1

2.4 Структура разрабатываемого программного комплекса

Программный комплекс состоит из двух модулей:

- Модуль, реализующий модель UNet сети для семантической сегментации;

- Пользовательское приложение, производящее сегментацию на основе полученной модели и машины опорных векторов.

2.4.1 Модуль UNet модели

На рисунке 2.4 представлена схема работы с модулем UNet модели.

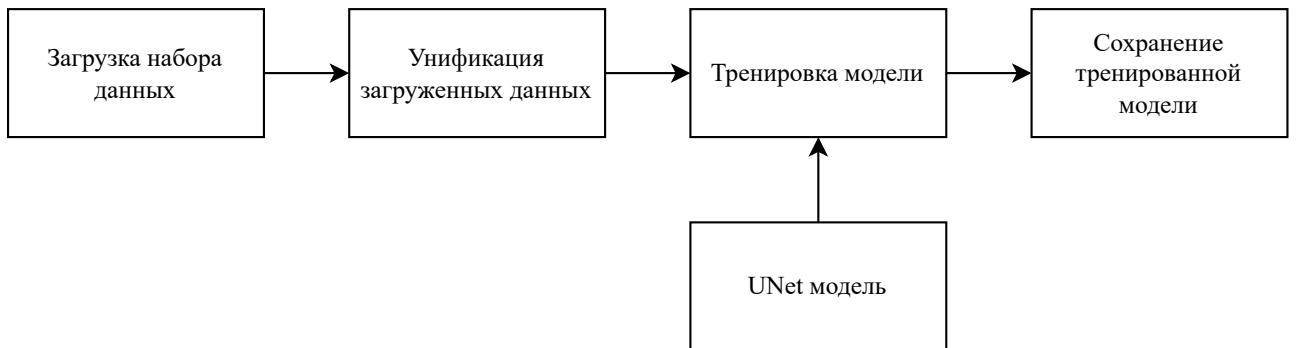


Рис. 2.4: Схема работы с модулем UNet модели

В данном модуле происходит только тренировка и сохранения модели. Модуль должен быть использован только один раз при первой тренировке модели или когда в модель требуется внести изменения. Вся последующая работа с моделью ведется через файл, который содержит в себе натренированную модель, так как процесс тренировки занимает продолжительное время.

2.4.2 Модуль пользовательского приложения

На рисунке 2.5 представлена схема работы с модулем UNet модели.

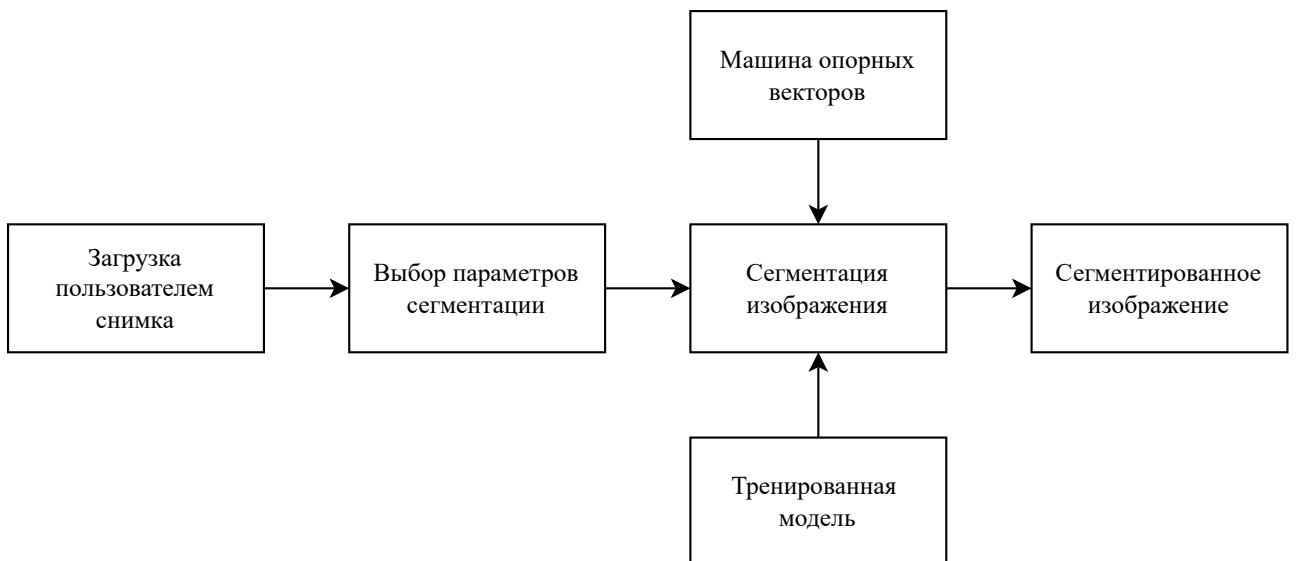


Рис. 2.5: Схема работы с модулем пользовательского приложения

При работе с данным модулем пользователь может загрузить изображение для дальнейшего анализа и выбрать какие функции нужно применить в

процессе распознавания (только выделение костей или также классификация зубов). Результатом работы данного модуля будет являться изображение с выделенными (и классифицированными, при необходимости) костями лица.

2.5 Данные для обучения модели

В качестве данных для обучения модели был выбран датасет, состоящий из КТ снимков 116 взрослых пациентов [24]. В состав датасета входят снимки как полностью здоровых зубочелюстных суставов, так и суставов с полностью отсутствующим зубным составом. Все снимки сделаны анфас.

На рисунке 2.6 и 2.7 представлены пример КТ снимков из используемого датасета с полностью здоровым зубным составом и с отсутствующим соответственно.

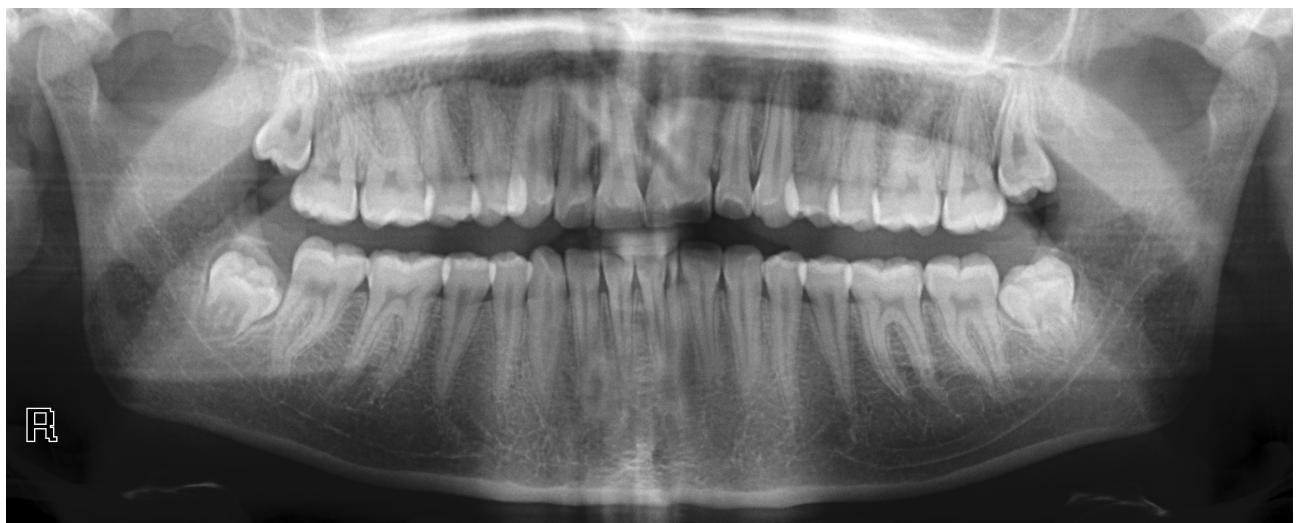


Рис. 2.6: Пример снимка из датасета (полный зубной состав)

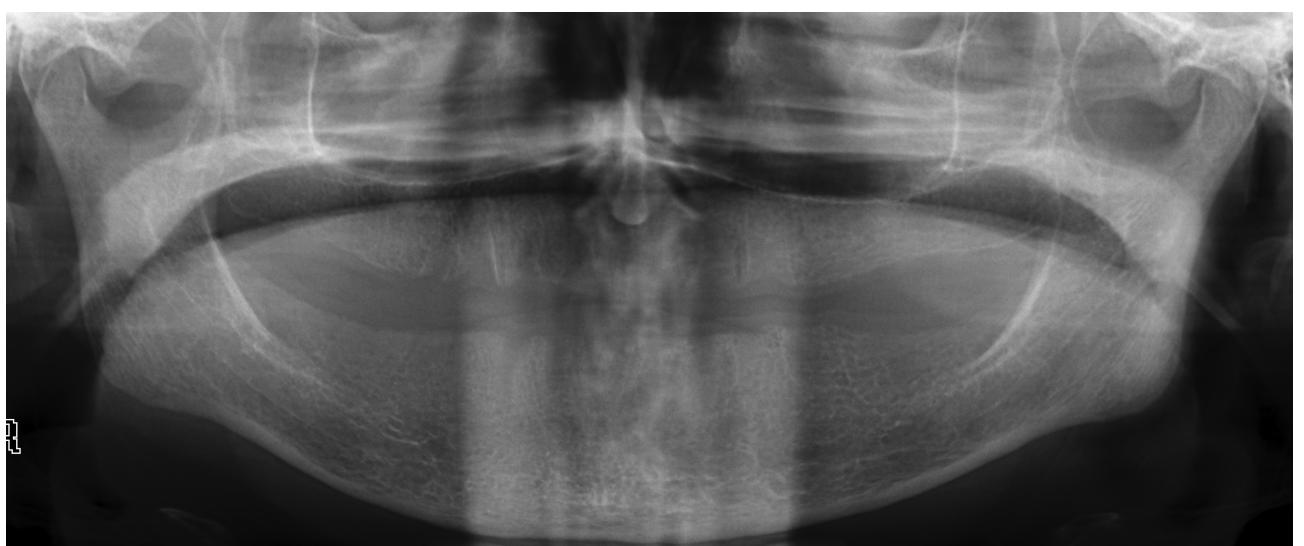


Рис. 2.7: Пример снимка из датасета (отсутствующий зубной состав)

Данные размечены ортодонтами клиники, предоставившей данный датасет. Для каждого снимка размечены нижняя челюсть и зубной состав.

На рисунке 2.8 представлен пример разметки нижней челюсти.

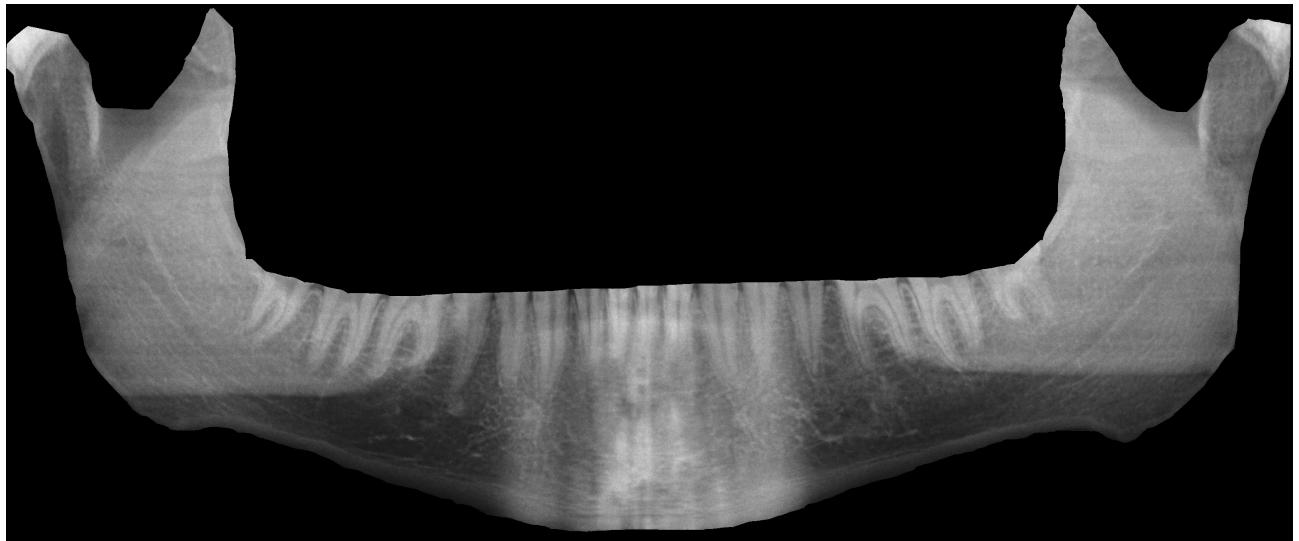


Рис. 2.8: Пример разметки нижней челюсти

Вывод

Были представлены требования к разрабатываемому методу распознавания и программному комплексу, реализующему интерфейс взаимодействия с методом.

Обозначены особенности разрабатываемого программного комплекса, показано применение машины опорных векторов при классификации распознанных костей.

Представлены схемы работы с разрабатываемыми программными модулями.

Представлен выбор датасета для обучения модели и примеры из него.

3. Технологический раздел

В данном разделе описываются средства реализации программного комплекса. Приводятся детали реализации программных компонентов и процесс обучения разрабатываемой нейронной сети.

3.1 Средства реализации программного комплекса

3.1.1 Выбор языка программирования

Для написания программного комплекса будет использоваться язык программирования Python [25].

Данный выбор обусловлен следующими факторами:

- широкий набор библиотек для работы с нейронными сетями;
- возможность тренировать нейронную сеть на графическом процессоре с использованием технологии CUDA [26];

3.1.2 Выбор библиотеки глубокого обучения

Для создания и обучения модели нейронной сети была выбрана библиотека tensorflow [27] версии 2.3.0. Выбор данной версии обусловлен тем, что версия 2.3.0 является последней версией с поддержкой CUDA 10, которая предоставляется на высоконагруженном кластере NVIDIA DGX2, на котором будет обучаться нейронная сеть.

Кроме того, tensorflow показал себя производительнее, чем pytorch, что является плюсом, так как тренировка модели с tensorflow займет меньше времени, чем с pytorch [28].

3.1.3 Выбор средства реализации машины опорных векторов

Для реализации классификатора машины опорных векторов будет использована библиотека Scikit Learn. Она предоставляет реализацию машины опорных векторов и требует от пользователя только данные для обучения [29].

3.2 Реализация программного комплекса

3.2.1 Модель UNet

Реализация модели построена на классической UNet модели с 23 сверточными слоями. В качестве функции активации на всех слоях, кроме последнего, используется ReLU. На последнем слое используется сигмоидальная функция активации. Она является более точной, чем ReLU, но менее быстрой. Именно поэтому для небольшого увеличения точности она используется именно на последнем слое.

В листинге 5 (приложение А) приведена реализация модели, отвечающей за сегментацию изображения.

На выходе модель предоставляет маску для изображения, по которой происходит дальнейшее выделение. Примеры таких масок приведены на рисунках 3.1 и 3.2 для зубов и для нижней челюсти соответственно.

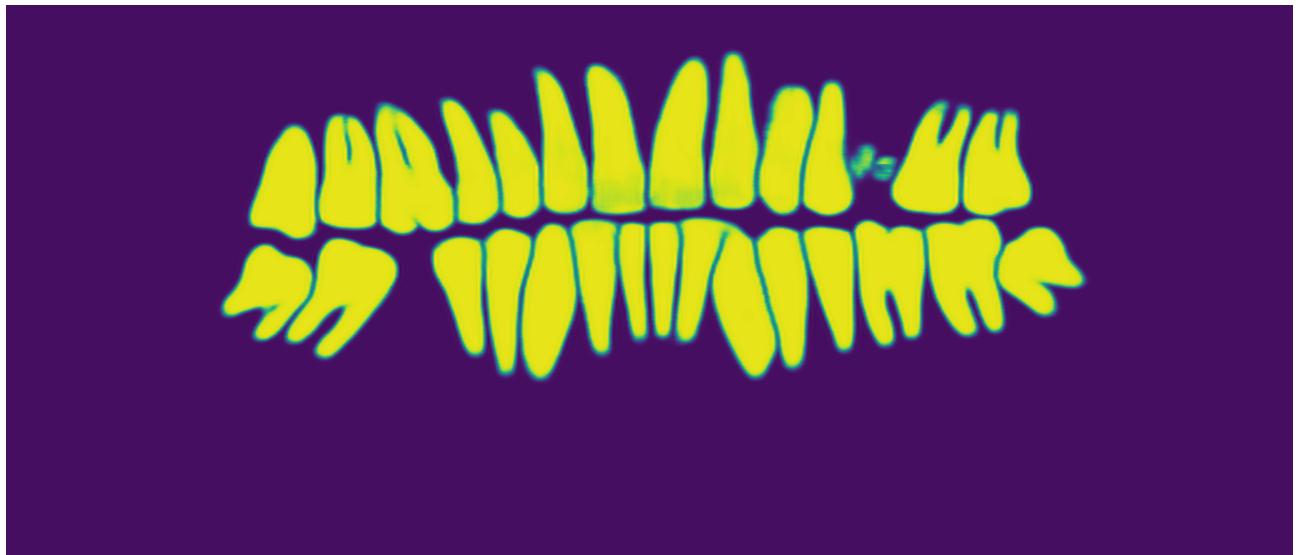


Рис. 3.1: Результат работы модели (зубной состав)



Рис. 3.2: Результат работы модели (нижняя челюсть)

3.2.2 Тренировка модели

Перед тренировкой модели необходимо произвести конвертацию исходных изображений и их масок к размеру 512×512 пикселей, как этого требует модель.

Кроме того, в процессе обучения, тренировочные данные подвергаются преобразованиям, таким как обрезка, изменение контраста, поворот, отражения, применение Гауссова шума и прочие. Это нужно для расширения набора тестовых данных и повышения точности модели.

В качестве функции оптимизации для модели будет использована функция Adam (адаптивная оценка момента). Adam — один из самых эффективных алгоритмов оптимизации в обучении нейронных сетей. Он сочетает в себе идеи среднеквадратичного распространения корня (RMSProp) и оптимизатора импульса. Вместо того чтобы адаптировать скорость обучения параметров на основе среднего первого момента (среднего значения), как в RMSProp, Adam также использует среднее значение вторых моментов градиентов. В частности, алгоритм вычисляет экспоненциальное скользящее среднее градиента и квадратичный градиент.

Обучение проводилось на графическом процессоре NVIDIA Tesla V100 с 32 гигабайтами видеопамяти, что позволило существенно ускорить процесс обучения. Обучение одной эпохи заняло 21 секунду по сравнению с 32 минутами при обучении на процессоре Intel Core I7 6-th Gen.

В листинге 6 (приложение А) приведена реализация процесса обучения.

На рисунках 3.3 — 3.5 приведены примеры тренировочных данных для модели: снимок лица, маска зубов и маска нижней челюсти соответственно.



Рис. 3.3: Пример тренировочных данных. Томографический снимок

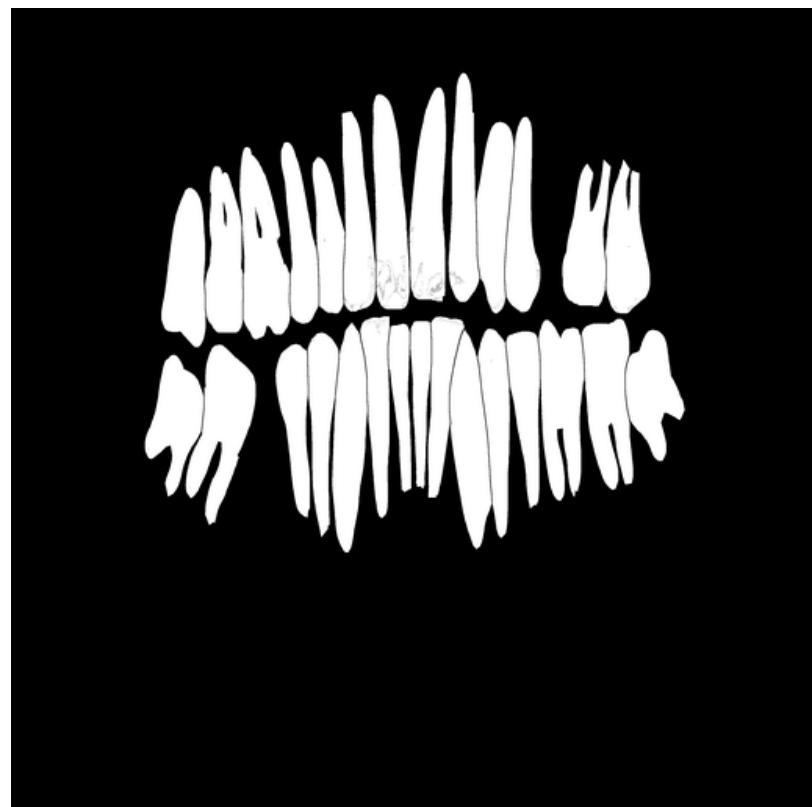


Рис. 3.4: Пример тренировочных данных. Маска зубов



Рис. 3.5: Пример тренировочных данных. Маска нижней челюсти

Тренировка проходила в течение 200 эпох (итераций) для обеспечения стабильного результата точности и потерь.

3.2.3 Классификация зубов при помощи машины опорных векторов

Для реализации машины опорных векторов, используемый датасет был размечен в мануальном режиме. Разметка производилась путем вычисления размеров зубов снимков и выделения полученных размеров в четыре группы: моляры, премоляры, улыки и резцы. При формировании векторов особенностей учитывались два фактора: ширина и высота зуба. Вычислить порядковый номер зуба в ряду не представляется возможным ввиду представления полученных при сегментации данных.

В листинге 11 (приложение Б) приведена реализация и обучение машины опорных векторов.

3.2.4 Выделение сегментированных участков

Для выделения сегментированных (и классифицированных) участков используется метод ССА (от англ. Connected Components Analysis — анализ связанных компонентов). Данный анализ проводится при помощи выделения на изображении объектов переднего плана (сегментированных участков) и последующего анализа полученного участка. Анализ проводится на основе различных факторов, таких как размер, абсолютное и относительное расположение объекта [30].

В листинге 11 (приложение Б) приведена реализация анализа связанных компонентов с их последующим выделением.

На рисунках 3.6 — 3.9 представлены изображения с выделенными сегментированными участками.

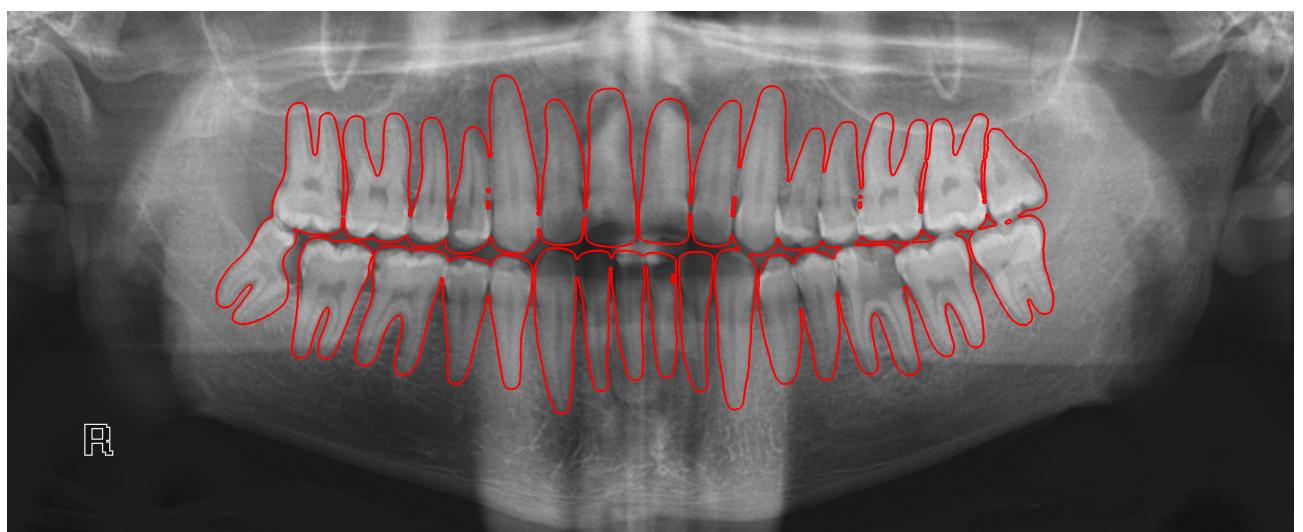


Рис. 3.6: Результат работы семантической сегментации (зубной состав)

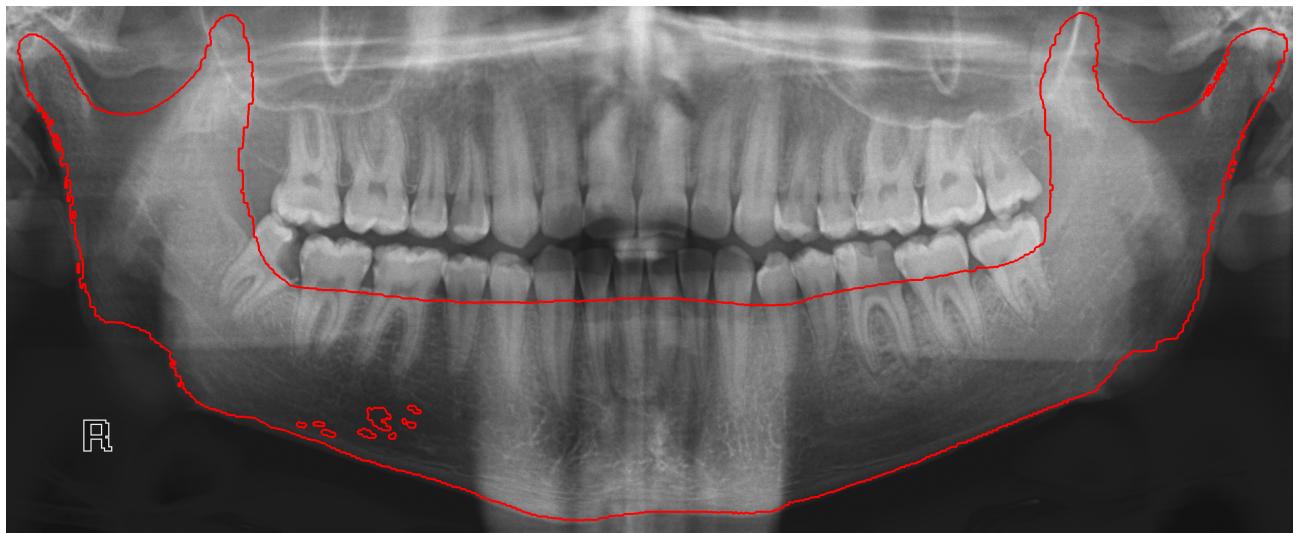


Рис. 3.7: Результат работы семантической сегментации (нижняя челюсть)

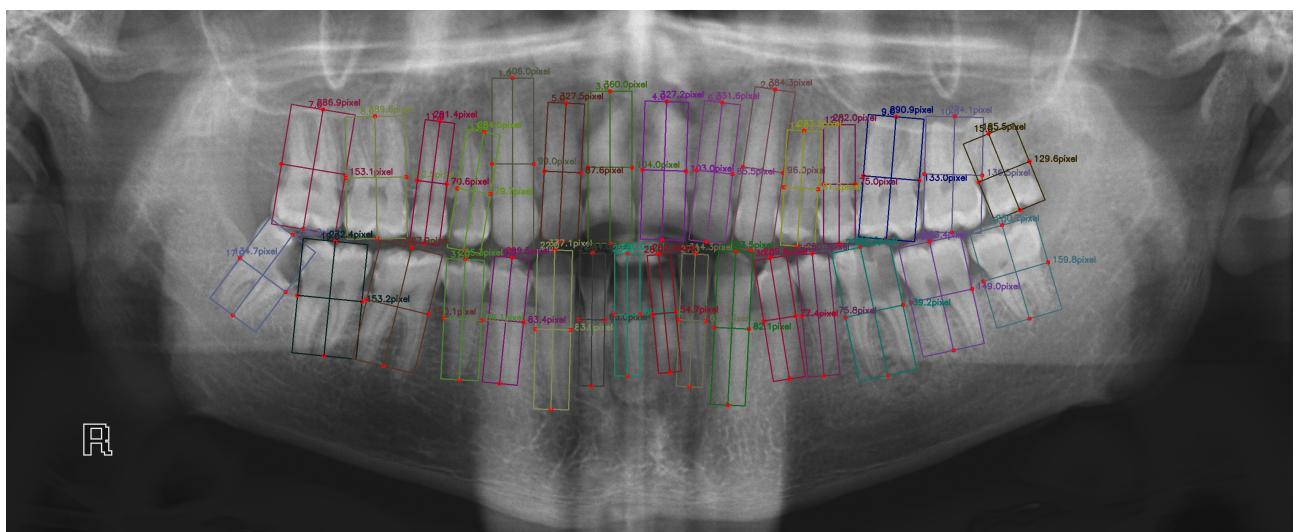


Рис. 3.8: Результат работы сегментации экземпляров

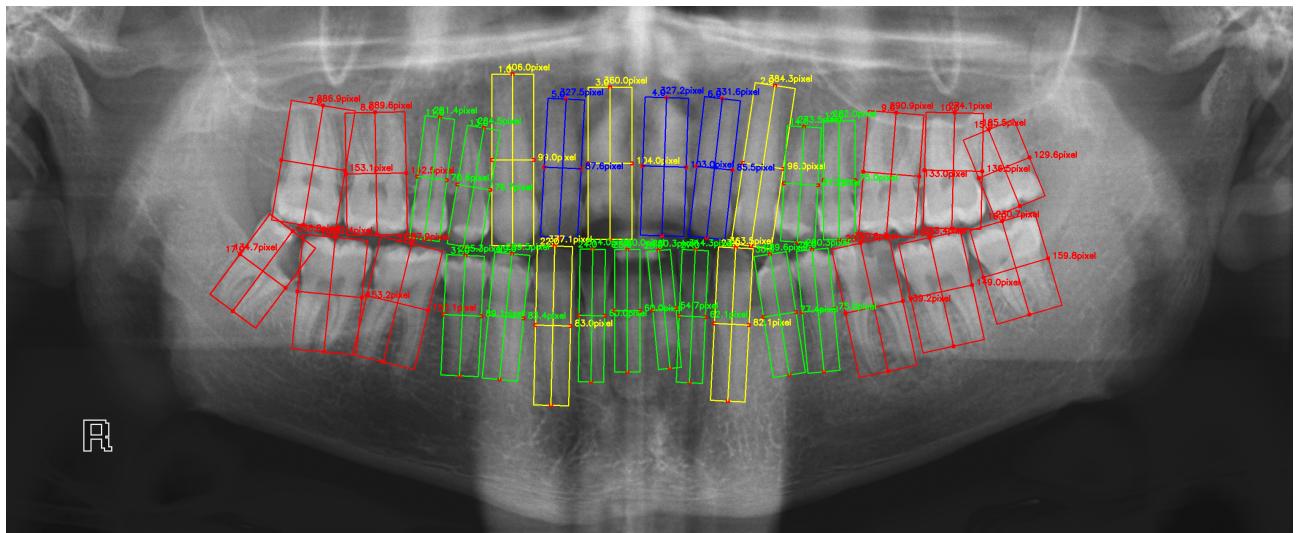


Рис. 3.9: Результат работы сегментации экземпляров (с применением машины опорных векторов)

3.3 Результаты обучения модели

На рисунках 3.10 и 3.11 представлены результаты обучения модели. Точность модели составила 92 процента, а потери составили 4 процента. Стоит заметить, что уже на десятой итерации точность составила не менее 88 процентов.

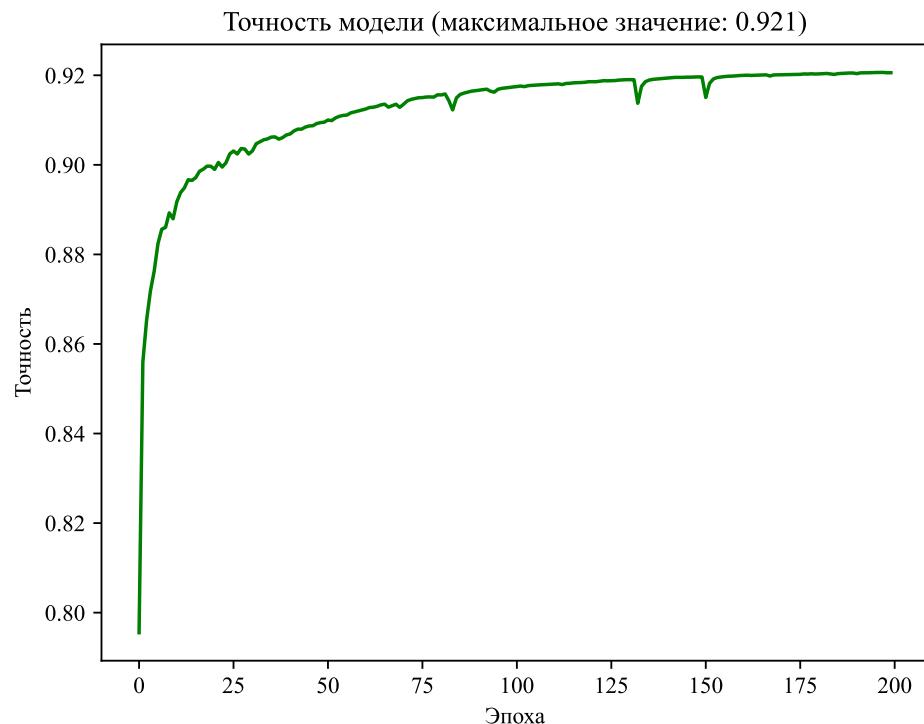


Рис. 3.10: Точность модели

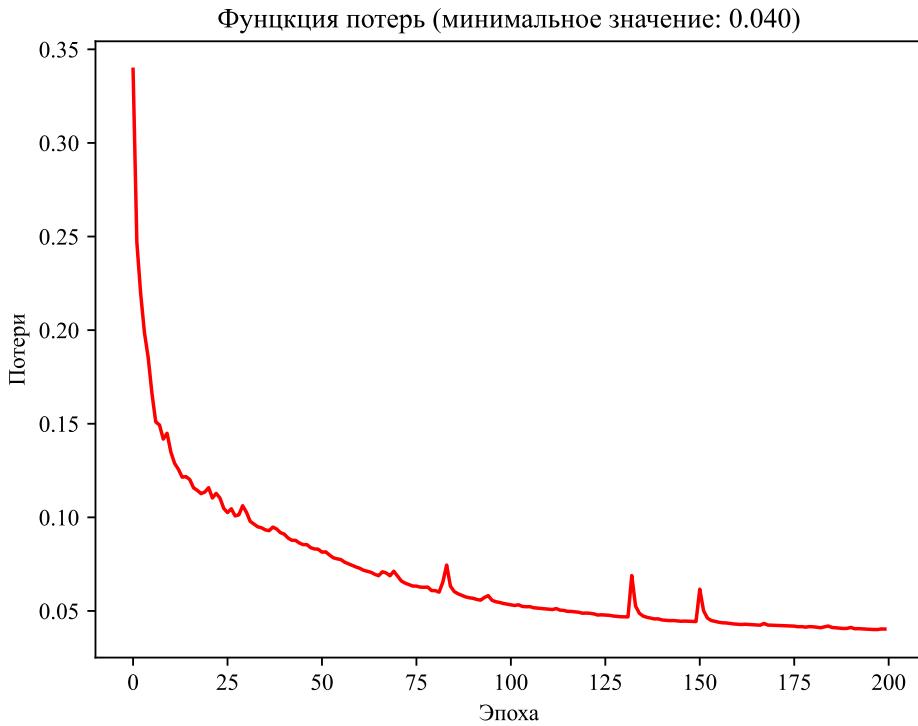


Рис. 3.11: Потери модели

3.4 Примеры использования разработанного программного комплекса

Оба модуля разработанного программного комплекса представляют собой консольные приложения без графического интерфейса, конфигурируемые при помощи аргументов командной строки.

3.4.1 Пример использования модуля модели UNet

В листинге 1 представлены параметры с которыми можно взаимодействовать с модулем обучения модели.

Листинг 1: Взаимодействие с модулем модели

```

1 python main.py -h
2
3 usage: main.py [-h] [-m] [-o OUTPUT] [-d DATA]
4
5 optional arguments:
6 -h, --help            show this help message and exit
7 -m                   Произвести тренировку челюстной сегментации
8 -o OUTPUT, --output OUTPUT Путь сохранения обученной модели
9 -d DATA, --data DATA Путь сохранения истории обучения

```

В листинге 2 представлены пример запуска модуля для обучения зубной сегментации.

Листинг 2: Запуск обучения

```
1 python main.py -o trained/teeth.h5 -d trained/teeth.hist
2
3 Extracting data/dataset.zip...
4 Extracted to data
5 Training teeth segmentation
6 Extracting masks/teeth.zip...
7 Extracted to masks/teeth
8 2022-05-24 01:51:26.391184: I tensorflow/core/platform/cpu_feature_guard.cc
   :151] This TensorFlow binary is optimized with oneAPI Deep Neural Network
   Library (oneDNN) to use the following CPU instructions in performance-
   critical operations: SSE4.2 AVX AVX2 FMA
9 To enable them in other operations, rebuild TensorFlow with the appropriate
   compiler flags.
10 Epoch 1/200
11 1/73 [...........................] - ETA: 33:51 - loss: 0.8040 - accuracy:
   0.2244
12 ...
```

3.4.2 Пример использования модуля пользовательского приложения

В листинге 3 представлены параметры с которыми можно взаимодействовать с модулем обучения модели.

Листинг 3: Взаимодействие с пользовательским приложением

```
1 python main.py -h
2
3 usage: main.py [-h] [-m MODEL] [-i IMAGE] [-o OUTPUT] [-c]
4
5 optional arguments:
6 -h, --help            show this help message and exit
7 -m MODEL, --model MODEL Путь к файлу с обученной моделью
8 -i IMAGE, --image IMAGE Путь к файлу снимка
9 -o OUTPUT, --output OUTPUT Директория для сохранения результатов
10 -c                  Произвести сегментацию по экземплярам
```

В листинге 4 представлены пример запуска приложения с включенной классификацией по типу зубов.

Листинг 4: Запуск приложения

```
1 python main.py -m ../train/trained/teeth.h5 -i ../train/data/Images/19.png -c
2
3 2022-05-24 01:57:50.542260: I tensorflow/core/platform/cpu_feature_guard.cc
   :151] This TensorFlow binary is optimized with oneAPI Deep Neural Network
   Library (oneDNN) to use the following CPU instructions in performance-
   critical operations: SSE4.2 AVX AVX2 FMA
```

- 4 To enable them **in** other operations, rebuild TensorFlow with the appropriate compiler flags.
- 5 Segmented teeth count **is** 31

Выход

Были описаны средства реализации программного комплекса. Приведены листинги реализации каждого компонента комплекса, примеры работы компонентов, их входные и выходные данные. Описаны технологии и методы, использовавшиеся при реализации. Представлены примеры взаимодействия с модулями.

4. Исследовательский раздел

В данном разделе проводится оценка качества разработанного метода и программного комплекса. Описывается его применимость в различных ситуациях. Даётся анализ достоинств и недостатков.

4.1 Сравнение применимости программного комплекса для снимков в разных проекциях

При обучении модели использовались снимки челюстно-лицевого сустава в анфас. Несмотря на это, обученную модель можно применять и на снимках в профиль.

На рисунках 4.1 и 4.2 приведены снимки одного черепа в разных проекциях. На данных снимках будет проведен эксперимент.



Рис. 4.1: Снимок анфас



Рис. 4.2: Снимок в профиль

На рисунках 4.3 и 4.4 соответственно приведены полученная маска и результат сегментации для снимка анфас. Как видно из снимков, никаких артефактов при проведении сегментации замечено не было.

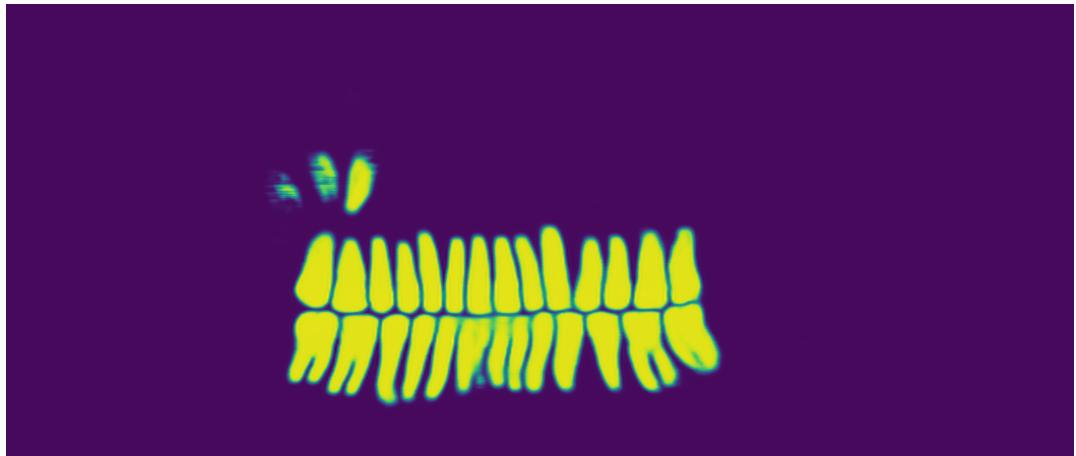


Рис. 4.3: Маска для снимка анфас

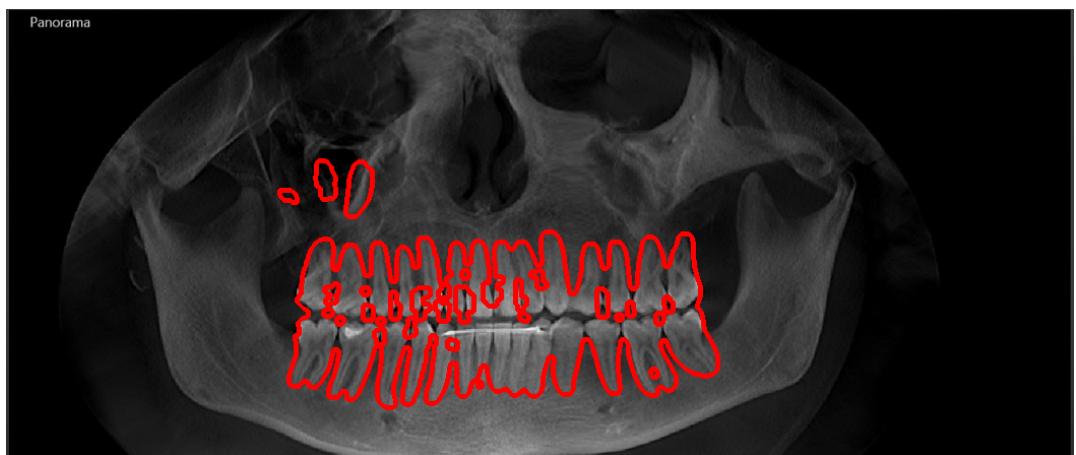


Рис. 4.4: Сегментация для снимка анфас

На рисунках 4.5 и 4.6 соответственно приведены полученная маска и результат сегментации для снимка в профиль. Так же как и со случаем анфас, в данном случае никаких артефактов замечено не было, маска для зубного состава была составлена корректно, за исключением небольшой области в левой верхней части снимка. Эта область является своеобразной водяной знаком снимка, такое отклонение допустимо и не затрагивает зубной состав, то есть не препятствует дальнейшему анализу полученных результатов.

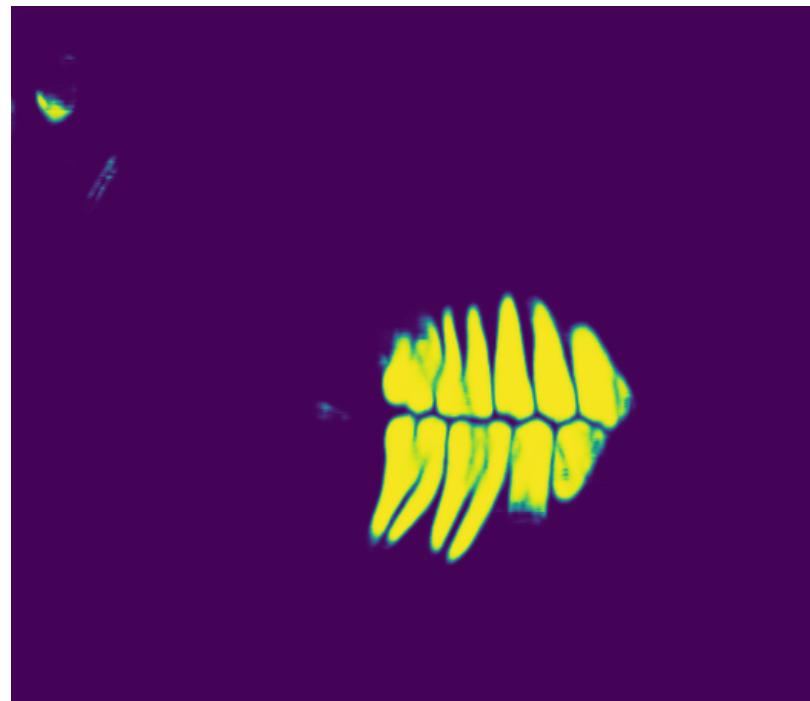


Рис. 4.5: Мaska для снимка в профиль

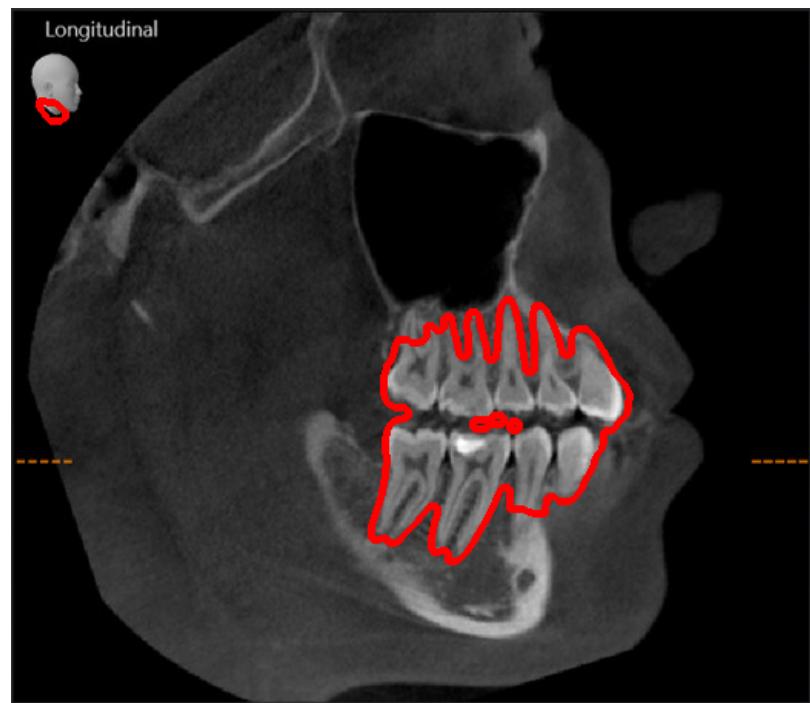


Рис. 4.6: Сегментация для снимка в профиль

4.2 Сравнение применимости программного комплекса для снимков в разных оттенках

В процессе обучения использовались снимки в черно–белой гамме. Помимо таких снимков, существуют аппараты, которые при снятии снимка сохраняют его в других цветовых тонах, например голубой вместо белого.

На рисунке 4.7 представлен пример такого снимка.

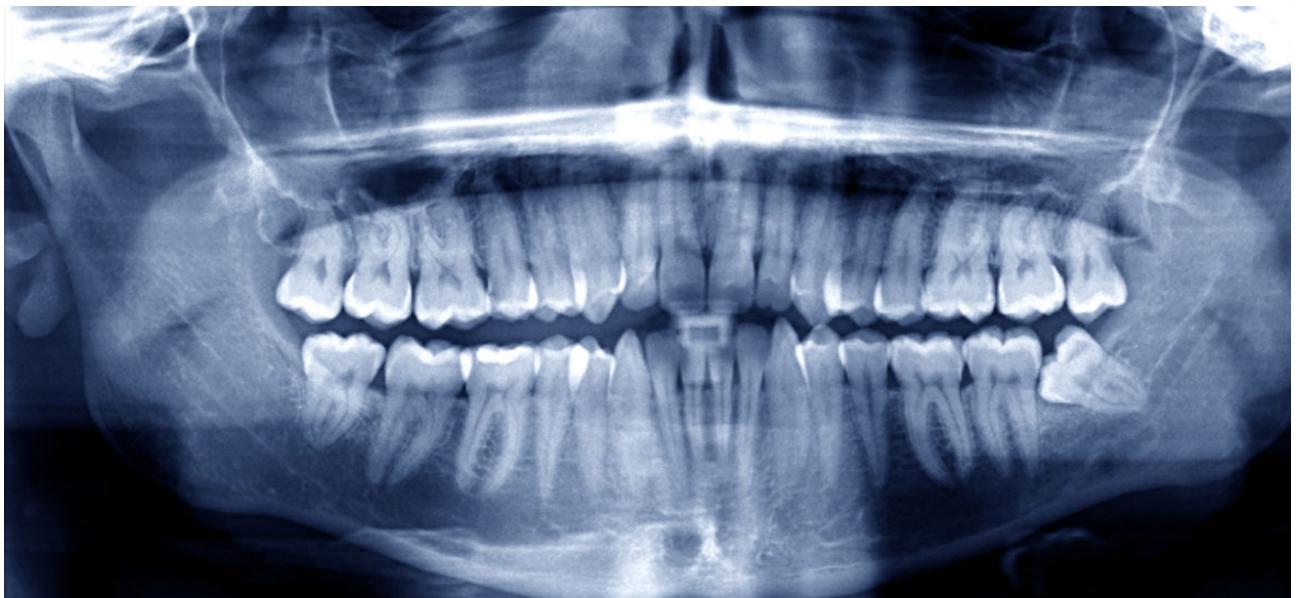


Рис. 4.7: Снимок в другой цветовой гамме

На рисунках 4.8 и 4.9 соответственно приведены полученная маска и результат сегментации для снимка в другой цветовой гамме. В целом сегментация проведена успешно, артефактов в щебном составе не наблюдается.

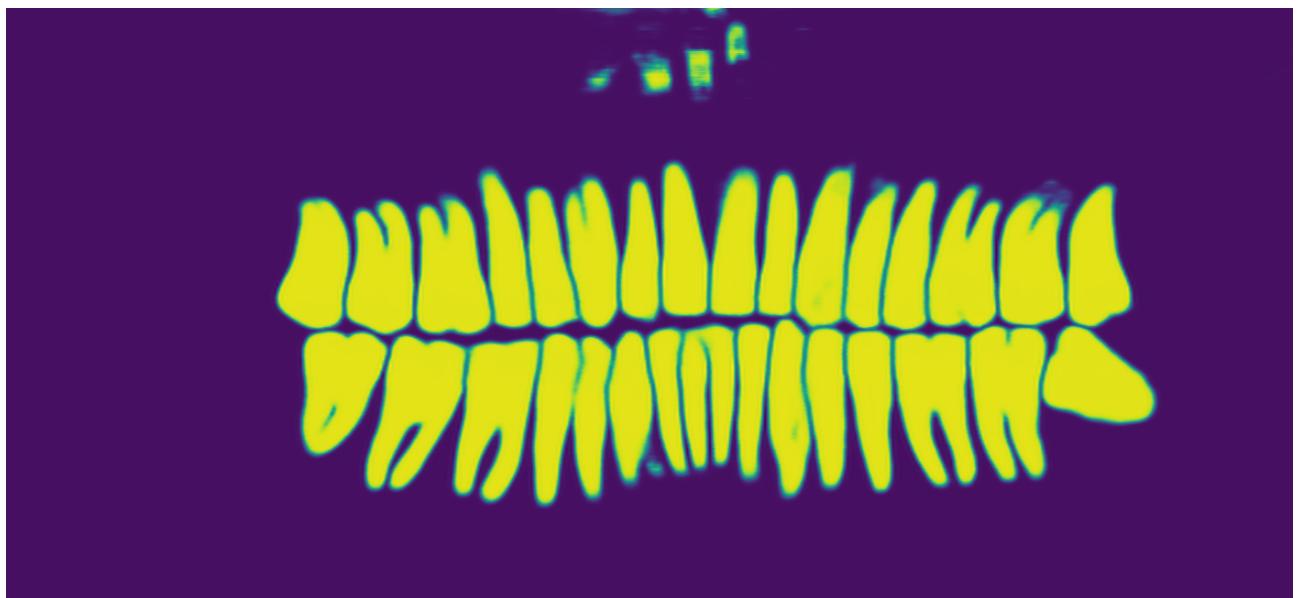


Рис. 4.8: Мaska для снимка в другой цветовой гамме



Рис. 4.9: Сегментация для снимка в другой цветовой гамме

4.3 Сравнение времени работы приложения при различном разрешении исходного снимка

Для проведения эксперимента будем использовать снимок, представленный на рисунке 4.10. Исходный снимок имеет разрешение 3050×1250 пикселей. Эксперимент будет проводиться на изображениях размером 100%, 75%, 50% и 25% от первоначального.

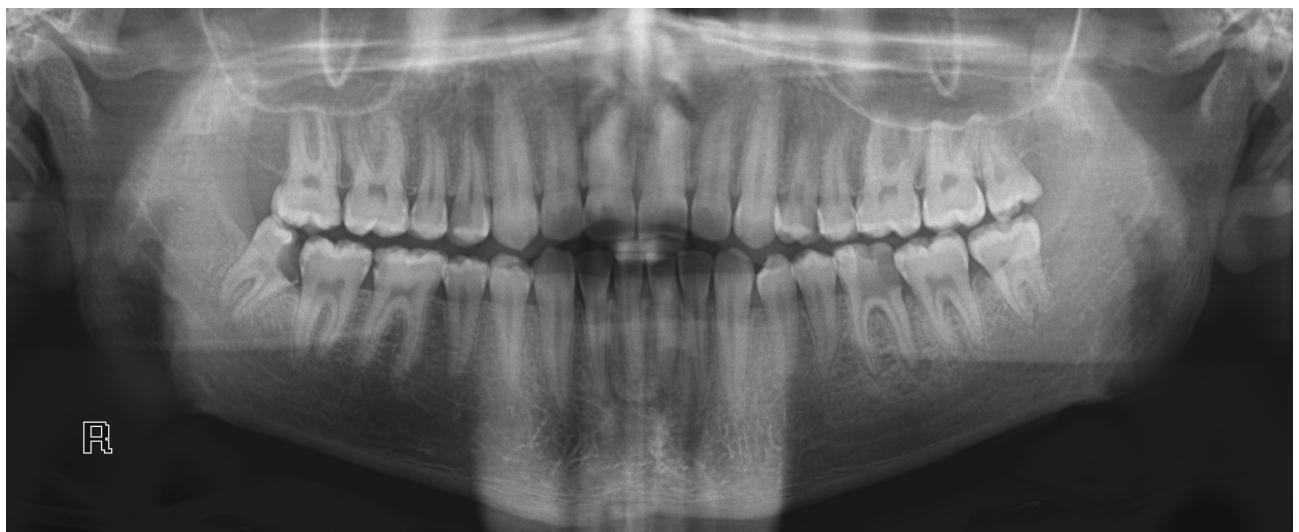


Рис. 4.10: Снимок для проведения эксперимента

В таблице 2 представлена зависимость времени работы приложения от размера исходного изображения. Также эта зависимость представлена на рисунке 4.11

Таблица 2: Время работы приложения в зависимости от размера исходного изображения

Режим работы приложения	100%, сек	75%, сек	50%, сек	25%, сек
Без классификации	14.4	12.68	11.48	10.46
С классификацией	17.5	13.18	11.03	10.37c

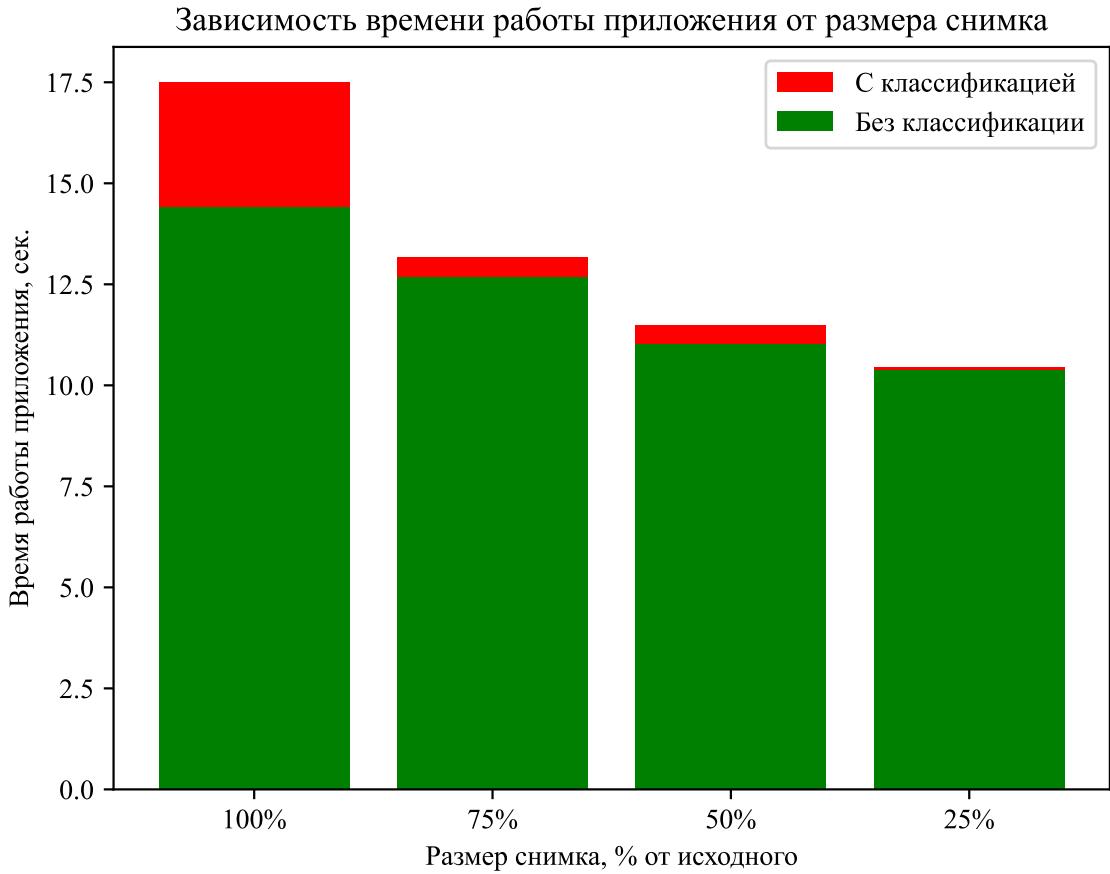


Рис. 4.11: Зависимость времени работы приложения от размера исходного изображения

При уменьшении размера изображения потеря в качестве не наблюдалось, маска для сегментации всегда выделялась одинаково четко, без артефактов.

Уменьшение изображения повлияло на корректность работы классификации при помощи машины опорных векторов. Так как вектор особенностей состоял из параметров ширины и высоты зуба в пикселях, для изображений другого размера эти данные не являются корректными. Это стоит учитывать при работе с приложением.

Кроме того, классификация с уменьшением изображения занимает меньше времени. Это связано с более низкой нагрузкой на машину опорных векторов, так как приходится обрабатывать векторы, содержащие в себе меньшие числовые значения параметров модели.

4.4 Оценка разработанного программного комплекса

У разработанного программного комплекса можно выявить следующие достоинства и недостатки.

Достоинства:

- Универсальность. Возможно использование при анализе снимков в разных проекциях, разных размеров и разных цветовых гаммах.
- Высокая точность. Отсутствие потери точности при сегментации при разных размерах исходного снимка.

Недостатки:

- Классификация. При уменьшении разрешения снимка обнаруживаются ошибки классификации ввиду неполноты данных для обучения.

Вывод

Было проведено сравнение применимости разработанного программного комплекса при различных нетипичных сценариях. Протестировано быстродействие программы при использовании классификации и без ее использования. Представлены достоинства и недостатки программного комплекса.

ЗАКЛЮЧЕНИЕ

Были рассмотрены задачи распознавания образов и нахождения объектов на изображении. Были рассмотрены виды классификаторов и их применимость при распознавании костей томографическим снимкам. Рассмотрены задачи семантической сегментации и сегментации экземпляров, их трактовка в задаче распознавания челюстно–лицевых костей.

Было дано определение понятия нейронной сети, описаны виды нейронных сетей и принцип их работы.

Были рассмотрены особенности применения нейронных сетей в качестве классификаторов для распознавания и классификации костей по томографическим снимкам.

Были рассмотрены и проанализированы технологии (R–CNN, Fast R–CNN и Faster R–CNN) для распознавания образов при помощи нейронных сетей, приведены преимущества и недостатки рассмотренных технологий в задаче распознавания костей по томографическим снимкам.

Были рассмотрены и проанализированы технологии для выделения объектов на изображении в рамках задачи работы, такие как Mask R–CNN и UNet.

Были описаны критерии выбора данных для обучения модели, послужившей основанием для разработанного метода.

Был спроектирован метод распознавания челюстно–лицевых костей а также программный комплекс, реализующий интерфейс для метода. Выбраны данные для обучения модели, обоснован их выбор.

Был разработан программный комплекс, состоящий из двух модулей: модуль модели UNet и модуль пользовательского приложения. Приведены результаты обучения модели и показаны примеры использования разработанного программного комплекса.

Проведено исследование качества созданного программного комплекса. В результате исследования были выделены достоинства, среди которых универсальность и высокая точность, и недостатки, среди которых ошибки классификации при изменении размера изображения.

Таким образом, поставленная цель работы — разработать метод распознавания челюстно–лицевых костей черепа по томографическим снимкам головы человека, была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. РЕНТГЕНОВСКАЯ ТОМОГРАФИЯ • Большая российская энциклопедия - электронная версия [Электронный ресурс]. Режим доступа: <https://bigenc.ru/physics/text/3505948> (дата обращения 10.01.2022).
2. МАГНИТНО-РЕЗОНАНСНАЯ ТОМОГРАФИЯ • Большая российская энциклопедия - электронная версия [Электронный ресурс]. Режим доступа: <https://bigenc.ru/medicine/text/2153066> (дата обращения 10.01.2022).
3. Метод опорных векторов - Университет ИТМО [Электронный ресурс]. Режим доступа: [https://neerc.ifmo.ru/wiki/index.php?title=Метод_опорных_векторов_\(SVM\)](https://neerc.ifmo.ru/wiki/index.php?title=Метод_опорных_векторов_(SVM)) (дата обращения 10.05.2022).
4. Epidemiological analysis of facial fractures / Deepalakshmi Tantry, Aisha Nehla, Mahesh Santhraya [и др.] // International Journal of Otorhinolaryngology and Head and Neck Surgery. 2021. 06. Т. 7. с. 1176.
5. ISAPS. Survey: Aesthetic/Cosmetic procedures performed in 2018. 2018.
6. Facial fractures: classification and highlights for a useful report / Eva Roselló, Ana Granado, Miquel Garcia [и др.] // Insights into Imaging. 2020. 12. Т. 11.
7. Deep neural network improves fracture detection by clinicians / Robert Lindsey, Aaron Daluiski, Sumit Chopra [и др.] // Proceedings of the National Academy of Sciences. 2018. 10. Т. 115. с. 201806905.
8. Computed Tomography and Magnetic Resonance Imaging / Monique Bernsen, Marcel Straten, Gyula Kotek [и др.]. 2020. 06. Т. 216. С. 31–110.
9. Принципы работы компьютерного томографа (КТ) [Электронный ресурс]. Режим доступа: <https://teleradiologia.ru/компьютерная-томография-общие-принц/> (дата обращения 10.05.2022).
10. МРТ аппарат [Электронный ресурс]. Режим доступа: <https://tomograf.spb.ru/mrt-apparat> (дата обращения 10.05.2022).

11. Deep Learning based Staging of Bone Lesions from Computed Tomography Scans (January 2021) / Samira Masoudi, Sherif Mehralivand, Stephanie Harmon [и др.] // IEEE Access. 2021. 04. Т. PP. С. 1–1.
12. Jijo Bahzad, Mohsin Abdulazeez Adnan. Classification Based on Decision Tree Algorithm for Machine Learning // Journal of Applied Science and Technology Trends. 2021. 01. Т. 2. С. 20–28.
13. Automatic Classifier Selection for Non-Experts / Matthias Reif, Faisal Shafait, Markus Goldstein [и др.] // Pattern Analysis and Applications. 2012. 02. Т. 17.
14. Suwanda R, Syahputra Zulfahmi, Zamzami E. Analysis of Euclidean Distance and Manhattan Distance in the K-Means Algorithm for Variations Number of Centroid K // Journal of Physics: Conference Series. 2020. 06. Т. 1566. с. 012058.
15. Что такое нейронные сети? - Российская Федерация | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/ru-ru/cloud/learn/neural-networks> (дата обращения 04.02.2022).
16. О линейной регрессии - Российская Федерация | IBM [Электронный ресурс]. Режим доступа: <https://www.ibm.com/ru-ru/analytics/learn/linear-regression> (дата обращения 04.02.2022).
17. Selective Search for Object Recognition / Jasper Uijlings, K. Sande, T. Gevers [и др.] // International Journal of Computer Vision. 2013. 09. Т. 104. С. 154–171.
18. Krizhevsky Alex, Sutskever I, Hinton G. Imagenet classification with deep convolutional neural networks. 2012. 01. С. 1097–1105.
19. Girshick Ross. Fast r-cnn. 2015. 04.
20. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks / Shaoqing Ren, Kaiming He, Ross Girshick [и др.] // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2015. 06. Т. 39.
21. Chen Yu, Li Ying, Wang Gang. An Enhanced Region Proposal Network for object detection using deep learning method // PLOS ONE. 2018. 09. Т. 13. с. e0203897.

22. U-Net: нейросеть для сегментации изображений [Электронный ресурс]. Режим доступа: <https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/> (дата обращения 13.05.2022).
23. Comparing Mask R-CNN and U-Net architectures for robust automatic segmentation of immune cells in immunofluorescence images of Lupus Nephritis biopsies / Madeleine S. Durkee, Rebecca Abraham, Junting Ai [и др.] // Imaging, Manipulation, and Analysis of Biomolecules, Cells, and Tissues XIX / под ред. Irene Georgakoudi, Attila Tarnok; International Society for Optics and Photonics. Т. 11647. SPIE, 2021. С. 109 – 115. URL: <https://doi.org/10.1117/12.2577785>.
24. Panoramic Dental X-rays with Segmented Mandibles [Электронный ресурс]. Режим доступа: <https://data.mendeley.com/datasets/hxt48yk462/1> (дата обращения 20.05.2022).
25. Welcome to Python.org [Электронный ресурс]. Режим доступа: <https://www.python.org/> (дата обращения 18.05.2022).
26. What is CUDA | NVIDIA official blog [Электронный ресурс]. Режим доступа: <https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/> (дата обращения 18.05.2022).
27. Tensorflow [Электронный ресурс]. Режим доступа: <https://tensorflow.org/> (дата обращения 18.05.2022).
28. Performance Analysis of Deep Learning Libraries: TensorFlow and PyTorch / Felipe Florencio, Thiago Silva, Edward Ordonez [и др.] // Journal of Computer Science. 2019. 05. Т. 15.
29. 1.4. Support Vector Machines | scikit-learn 1.1.0 documentation [Электронный ресурс]. Режим доступа: <https://scikit-learn.org/stable/modules/svm.html/> (дата обращения 18.05.2022).
30. Bailey Donald, Johnston Christopher, Ma Ni. Connected components analysis of streamed images. 2008. 10. С. 679 – 682.

ПРИЛОЖЕНИЕ А

Модуль модели UNet

Листинг 5: Модель UNet

```
1 from tensorflow.keras.layers import (BatchNormalization, Conv2D,
2                                         Conv2DTranspose, Dropout, Input,
3                                         MaxPooling2D, concatenate)
4 from tensorflow.keras.models import Model
5
6
7 def UNet(input_shape=(512, 512, 1), last_activation='sigmoid'):
8     inputs = Input(shape=input_shape)
9
10    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same',
11                  kernel_initializer='he_normal')(inputs)
12    d1 = Dropout(0.1)(conv1)
13    conv2 = Conv2D(32, (3, 3), activation='relu', padding='same',
14                  kernel_initializer='he_normal')(d1)
15    b = BatchNormalization()(conv2)
16
17    pool1 = MaxPooling2D(pool_size=(2, 2))(b)
18    conv3 = Conv2D(64, (3, 3), activation='relu', padding='same',
19                  kernel_initializer='he_normal')(pool1)
20    d2 = Dropout(0.2)(conv3)
21    conv4 = Conv2D(64, (3, 3), activation='relu', padding='same',
22                  kernel_initializer='he_normal')(d2)
23    b1 = BatchNormalization()(conv4)
24
25    pool2 = MaxPooling2D(pool_size=(2, 2))(b1)
26    conv5 = Conv2D(128, (3, 3), activation='relu', padding='same',
27                  kernel_initializer='he_normal')(pool2)
28    d3 = Dropout(0.3)(conv5)
29    conv6 = Conv2D(128, (3, 3), activation='relu', padding='same',
30                  kernel_initializer='he_normal')(d3)
31    b2 = BatchNormalization()(conv6)
32
33    pool3 = MaxPooling2D(pool_size=(2, 2))(b2)
34    conv7 = Conv2D(256, (3, 3), activation='relu', padding='same',
35                  kernel_initializer='he_normal')(pool3)
36    d4 = Dropout(0.4)(conv7)
37    conv8 = Conv2D(256, (3, 3), activation='relu', padding='same',
38                  kernel_initializer='he_normal')(d4)
39    b3 = BatchNormalization()(conv8)
40
```

```

41     pool4 = MaxPooling2D(pool_size=(2, 2))(b3)
42     conv9 = Conv2D(512, (3, 3), activation='relu', padding='same',
43                     kernel_initializer='he_normal')(pool4)
44     d5 = Dropout(0.5)(conv9)
45     conv10 = Conv2D(512, (3, 3), activation='relu', padding='same',
46                      kernel_initializer='he_normal')(d5)
47     b4 = BatchNormalization()(conv10)
48
49     conv11 = Conv2DTranspose(512, (4, 4), activation='relu', padding='same',
50                             strides=(2, 2), kernel_initializer='he_normal')(b4)
51     x = concatenate([conv11, conv8])
52     conv12 = Conv2D(256, (3, 3), activation='relu', padding='same',
53                     kernel_initializer='he_normal')(x)
54     d6 = Dropout(0.4)(conv12)
55     conv13 = Conv2D(256, (3, 3), activation='relu', padding='same',
56                     kernel_initializer='he_normal')(d6)
57     b5 = BatchNormalization()(conv13)
58
59     conv14 = Conv2DTranspose(256, (4, 4), activation='relu', padding='same',
60                             strides=(2, 2), kernel_initializer='he_normal')(b5)
61     x1 = concatenate([conv14, conv6])
62     conv15 = Conv2D(128, 3, activation='relu', padding='same',
63                     kernel_initializer='he_normal')(x1)
64     d7 = Dropout(0.3)(conv15)
65     conv16 = Conv2D(128, 3, activation='relu', padding='same',
66                     kernel_initializer='he_normal')(d7)
66     b6 = BatchNormalization()(conv16)
67
67     conv17 = Conv2DTranspose(128, (4, 4), activation='relu', padding='same',
68                             strides=(2, 2), kernel_initializer='he_normal')(b6)
69     x2 = concatenate([conv17, conv4])
70     conv18 = Conv2D(64, (3, 3), activation='relu', padding='same',
71                     kernel_initializer='he_normal')(x2)
72     d8 = Dropout(0.2)(conv18)
73     conv19 = Conv2D(64, (3, 3), activation='relu', padding='same',
74                     kernel_initializer='he_normal')(d8)
75     b7 = BatchNormalization()(conv19)
76
77     conv20 = Conv2DTranspose(64, (4, 4), activation='relu', padding='same',
78                             strides=(2, 2), kernel_initializer='he_normal')(b7)
79     x3 = concatenate([conv20, conv2])
80     conv21 = Conv2D(32, (3, 3), activation='relu', padding='same',
81                     kernel_initializer='he_normal')(x3)
82     d9 = Dropout(0.1)(conv21)

```

```

83     conv22 = Conv2D(32, (3, 3), activation='relu', padding='same',
84                     kernel_initializer='he_normal')(d9)
85
86     outputs = Conv2D(1, (1, 1), activation=last_activation, padding='same',
87                     kernel_initializer='he_normal')(conv22)
88     model = Model(inputs=inputs, outputs=outputs)
89
90     return model

```

Листинг 6: Обучение модели

```

1 import argparse
2 import os
3 import pickle
4
5 import albumentations as A
6 import cv2
7 import numpy as np
8 from dataset import download_dataset, prepare_dataset, prepare_masks
9 from imageprep import pre_images
10 from maskprep import pre_masks
11 from model import UNet
12 from tensorflow.keras.models import save_model
13
14
15 def main(model_path, history_path, trainTeeth):
16     if not os.path.exists('data/dataset.zip'):
17         download_dataset('data')
18
19     prepare_dataset('data/dataset.zip', 'data')
20
21     if trainTeeth:
22         print('Training teeth segmentation')
23         prepare_masks('masks/teeth.zip', 'masks/teeth')
24         y_train = pre_masks('masks/teeth')
25     else:
26         print('Training mandibles segmentation')
27         prepare_masks('masks/mandibles.zip', 'masks/mandibles')
28         y_train = pre_masks('masks/mandibles')
29
30     x_train = pre_images((512, 512), 'data/Images')
31
32     x_train = np.float32(x_train/255)
33     y_train = np.float32(y_train/255)
34
35     aug = A.Compose([
36         A.OneOf([A.RandomCrop(width=512, height=512),
37                  A.PadIfNeeded(min_height=512, min_width=512, p=0.5)], p=0.4),

```

```

38         A.RandomBrightnessContrast(brightness_limit=0.25, contrast_limit
39             =0.25, p=0.5),
40         A.Compose([A.RandomScale(scale_limit=(-0.15, 0.15), p=1,
41             interpolation=1),
42             A.PadIfNeeded(512, 512, border_mode=cv2.BORDER_CONSTANT),
43             A.Resize(512, 512, cv2.INTER_NEAREST), ], p=0.5),
44         A.ShiftScaleRotate(shift_limit=0.325, scale_limit=0.15, rotate_limit
45             =15,
46             border_mode=cv2.BORDER_CONSTANT, p=1),
47         A.Rotate(15, p=0.5),
48         A.Blur(blur_limit=1, p=0.5),
49         A.Downscale(scale_min=0.15, scale_max=0.25, always_apply=False, p
50             =0.5),
51         A.GaussNoise(var_limit=(0.05, 0.1), mean=0, per_channel=True,
52             always_apply=False, p=0.5),
53         A.HorizontalFlip(p=0.25),
54     ])
55
56     x_train1 = np.copy(x_train)
57     y_train1 = np.copy(y_train)
58
59     count = 0
60
61     while(count < 4):
62         x_aug = np.copy(x_train1)
63         y_aug = np.copy(y_train1)
64
65         for i in range(len(x_train1)):
66             augmented = aug(image=x_train1[i, :, :, :], mask=y_train1[i, :, :
67             :, :])
68             x_aug[i, :, :, :] = augmented['image']
69             y_aug[i, :, :, :] = augmented['mask']
70             x_train = np.concatenate((x_train, x_aug))
71             y_train = np.concatenate((y_train, y_aug))
72
73         if count == 9:
74             break
75
76         count += 1
77
78     mem_to_free = [x_aug, x_train, y_train, y_aug, y_train1, x_train1,
79     augmented]
80
81     for mem in mem_to_free:
82         del mem
83
84
85     model = UNet(input_shape=(512, 512, 1), last_activation='sigmoid')
86     model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
87         accuracy'])
88
89     data = model.fit(x_train, y_train, batch_size=8, epochs=200, verbose=1)

```

```

78
79     data_file = open(history_path, 'wb')
80     pickle.dump(data.history, data_file)
81     data_file.close()
82
83     save_model(model, model_path)
84
85
86 if __name__ == '__main__':
87     parser = argparse.ArgumentParser()
88     teeth = True
89     parser.add_argument('-m', action='store_const', default=teeth, const=False,
90                         teeth,
91                         help='Произвести тренировку челюстной сегментации')
92     parser.add_argument('-o', '--output', default='trained/bonerecon.h5',
93                         help='Путь сохранения обученной модели')
94     parser.add_argument('-d', '--data', default='trained/bonerecon.hist',
95                         help='Путь сохранения истории обучения')
96
97     args = parser.parse_args()
98
99     main(args.output, args.data, args.m)

```

Листинг 7: Загрузка данных для обучения

```

1 import os
2 from io import BytesIO
3 from zipfile import ZipFile
4
5 import requests
6
7 DATASET_LINK = 'https://md-datasets-cache-zipfiles-prod.s3.eu-west-1.
amazonaws.com/hxt48yk462-1.zip'
8
9
10 def download_dataset(path):
11     print(f'Downloading dataset to {path}...')
12     r = requests.get(DATASET_LINK)
13     z = ZipFile(BytesIO(r.content))
14     z.extractall(path)
15     os.rename(f'{path}/{z.namelist()[0]}', f'{path}/dataset.zip')
16
17     print(f'Downloaded to {path}/dataset.zip')
18
19
20 def prepare_dataset(path='data/dataset.zip', extract_to='data'):
21     print(f'Extracting {path}...')
22

```

```

23     if not os.path.exists(extract_to):
24         os.mkdir(extract_to)
25
26     ZipFile(path).extractall(extract_to)
27     print(f'Extracted to {extract_to}')
28
29
30 def prepare_masks(path='masks/masks.zip', extract_to='data'):
31     print(f'Extracting {path}...')
32
33     if not os.path.exists(extract_to):
34         os.mkdir(extract_to)
35
36     ZipFile(path).extractall(extract_to)
37     print(f'Extracted to {extract_to}')

```

Листинг 8: Предобработка снимков

```

1 import os
2
3 import numpy as np
4 from image import convert_one_channel
5 from natsort import natsorted
6 from PIL import Image
7
8
9 def pre_images(resize_shape, path):
10     dirs = natsorted(os.listdir(path))
11     img = Image.open(f'{path}/{dirs[0]}')
12     images = convert_one_channel(np.asarray(img.resize((resize_shape),
13             Image.ANTIALIAS)))
14
15     for i in range(1, len(dirs)):
16         img = Image.open(f'{path}/{dirs[i]}')
17         img = img.resize((resize_shape), Image.ANTIALIAS)
18         img = convert_one_channel(np.asarray(img))
19         images = np.concatenate((images, img))
20
21     images = np.reshape(images, (len(dirs), resize_shape[0], resize_shape[1],
22     1))
23
24     return images

```

Листинг 9: Предобработка масок

```

1 import os
2
3 import numpy as np

```

```
4 from image import convert_one_channel
5 from natsort import natsorted
6 from PIL import Image
7
8
9 def pre_masks(path='masks'):
10     dirs = natsorted(os.listdir(path))
11     masks = img = Image.open(f'{path}/{dirs[0]}')
12     masks = convert_one_channel(np.asarray(masks))
13
14     for i in range(1, len(dirs)):
15         img = Image.open(f'{path}/{dirs[i]}')
16         img = convert_one_channel(np.asarray(img))
17         masks = np.concatenate((masks, img))
18
19     masks = np.reshape(masks, (len(dirs), 512, 512, 1))
20
21     return masks
```

ПРИЛОЖЕНИЕ Б

Модуль пользовательского приложения

Листинг 10: Пользовательское приложение

```
1 import argparse
2 import os
3
4 import cv2
5 import matplotlib.pyplot as plt
6 import numpy as np
7 from cca import analyze
8 from image import convert_one_channel, convert_rgb
9 from PIL import Image
10 from tensorflow.keras.models import load_model
11
12
13 def main(model, image, output_dir, use_svm):
14     model = load_model(model)
15     img = Image.open(image)
16     img = np.asarray(img)
17
18     img_cv = convert_one_channel(img)
19     img_cv = cv2.resize(img_cv, (512, 512), interpolation=cv2.INTER_LANCZOS4)
20     img_cv = np.float32(img_cv/255)
21     img_cv = np.reshape(img_cv, (1, 512, 512, 1))
22
23     predicted = model.predict(img_cv)[0]
24     predicted = cv2.resize(
25         predicted, (img.shape[1], img.shape[0]), interpolation=cv2.
26         INTER_LANCZOS4)
27
28     if not os.path.exists(output_dir):
29         os.mkdir(output_dir)
30
31     plt.imsave(f'{output_dir}/predicted.png', predicted)
32
33     thresh = np.uint8(predicted*255)
34     thresh = cv2.threshold(thresh, thresh=0, maxval=255, type=cv2.
35     THRESH_BINARY+cv2.THRESH_OTSU)[1]
36     kernel = (np.ones((5, 5), dtype=np.float32))
37     thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=1)
38     thresh = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel, iterations=1)
```

```

38     cnts = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
39     [0]
40     output = cv2.drawContours(convert_rgb(img), cnts, -1, (255, 0, 0), 3)
41     plt.imsave(f'{output_dir}/segmented.png', output)
42
43     img = cv2.imread(image)
44     predicted = cv2.imread(f'{output_dir}/predicted.png')
45     predicted = cv2.resize(predicted, (img.shape[1], img.shape[0]),
46                           interpolation=cv2.INTER_LANCZOS4)
47
48     cca, teeth_count = analyze(img, predicted, 3, 2)
49     plt.imsave(f'{output_dir}/segmented_cca.png', cca)
50
51     if use_svm:
52         svm, _ = analyze(img, predicted, 3, 2, use_svm)
53         plt.imsave(f'{output_dir}/segmented_svm.png', svm)
54     print(f'Segmented teeth count is {teeth_count}')
55
56
57 if __name__ == '__main__':
58     parser = argparse.ArgumentParser()
59     parser.add_argument('-m', '--model', default='trained/bonerecon.h5', help=
60                         'Путь к файлу с обученной моделью')
61     parser.add_argument('-i', '--image', default='data/file.jpeg', help='
62                         Путь к файлу снимка')
63     parser.add_argument('-o', '--output', default='predicted', help='
64                         Директория для сохранения результатов')
65     use_svm = False
66     parser.add_argument('-c', action='store_const', default=use_svm, const=
67                         not use_svm,
68                         help='Произвести сегментацию по экземплярам')
69
70     args = parser.parse_args()
71
72     main(args.model, args.image, args.output, args.c)

```

Листинг 11: Анализ связанных компонентов и машина опорных векторов

```

1 import cv2
2 import numpy as np
3 from imutils import perspective
4 from scipy.spatial import distance as dist
5 from sklearn import svm
6
7 MOLARES_COLOR = [255, 0, 0]
8 PREMOLARES_COLOR = [0, 255, 0]
9 CANINOS_COLOR = [255, 255, 0]
10 INCISIVOS_COLOR = [0, 0, 255]
11

```

```

12 COLORS = {
13     0: MOLARES_COLOR,
14     1: PREMOLARES_COLOR,
15     2: CANINOS_COLOR,
16     3: INCISIVOS_COLOR
17 }
18
19
20 def midpoint(ptA, ptB):
21     return ((ptA[0] + ptB[0]) * 0.5, (ptA[1] + ptB[1]) * 0.5)
22
23
24 def classifier():
25     X = [
26         # Molares
27         [286.9, 153.1],
28         [289.6, 142.5],
29         [236.8, 134.7],
30         [272.4, 153.2],
31         [287.9, 150.1],
32         [290.9, 133],
33         [274.1, 136.5],
34         [195.5, 129.6],
35         [310.6, 139.2],
36         [263.4, 149],
37         [230.7, 159.8],
38
39         [338.3, 144.5],
40         [309.5, 137.3],
41         [277.6, 149.5],
42         [365.9, 153.1],
43         [375.7, 147.6],
44         [351.4, 113.6],
45         [256, 133],
46         [283.3, 182.2],
47         [368.4, 153.7],
48         # Premolares
49         [281.4, 70.6],
50         [284.5, 79.7],
51         [285.3, 89.1],
52         [289.5, 83.4],
53         [273.5, 81.2],
54         [289.6, 77.4],
55         [290.3, 75.8],
56         [282, 75],
57         # Caninos
58         [406, 99],
59         [384.3, 96],

```

```

60      [377.1, 83],
61      [363.5, 82.1],
62      [433.6, 111],
63      [350.7, 88.4],
64      [388.8, 101.1],
65      [363.9, 91.2],
66      # Incisivos
67      [327.5, 87.6],
68      [360, 104],
69      [327.2, 103],
70      [331.6, 86.5],
71      [314, 60],
72      [288, 60],
73      [280, 64],
74      [314.3, 62.1],
75  ]
76
77  Y = [
78      # Molares
79      0,
80      0,
81      0,
82      0,
83      0,
84      0,
85      0,
86      0,
87      0,
88      0,
89      0,
90
91      0,
92      0,
93      0,
94      0,
95      0,
96      0,
97      0,
98      0,
99      0,
100     # Premolares
101     1,
102     1,
103     1,
104     1,
105     1,
106     1,
107     1,

```



```

153     count = 0
154
155     unique_labels = np.unique(labels)
156     for label in unique_labels:
157         if label == 0:
158             continue
159
160         mask = np.zeros(thresh.shape, dtype="uint8")
161         mask[labels == label] = 255
162
163         cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.
164 CHAIN_APPROX_SIMPLE)[0][0]
165         c_area = cv2.contourArea(cnts)
166
167         if c_area > 2000:
168             count += 1
169
170         rect = cv2.minAreaRect(cnts)
171         box = cv2.boxPoints(rect)
172         box = np.array(box, dtype="int")
173         box = perspective.order_points(box)
174
175         if use_svm:
176             tooth = rect[1]
177             if tooth[1] > tooth[0]:
178                 tooth = [tooth[1], tooth[0]]
179             color = COLORS[clf.predict([tooth])[0]]
180         else:
181             intcolor = (list(np.random.choice(range(150), size=3)))
182             color = [int(intcolor[0]), int(intcolor[1]), int(intcolor[2])]
183
184             cv2.drawContours(oimage, [box.astype("int")], 0, color, 2)
185             (tl, tr, br, bl) = box
186
187             (tltrX, tltrY) = midpoint(tl, tr)
188             (blbrX, blbrY) = midpoint(bl, br)
189
190             (tblbX, tblbY) = midpoint(tl, bl)
191             (trbrX, trbrY) = midpoint(tr, br)
192
193             cv2.circle(oimage, (int(tltrX), int(tltrY)), 5, (255, 0, 0), -1)
194             cv2.circle(oimage, (int(blbrX), int(blbrY)), 5, (255, 0, 0), -1)
195             cv2.circle(oimage, (int(tblbX), int(tblbY)), 5, (255, 0, 0), -1)
196             cv2.circle(oimage, (int(trbrX), int(trbrY)), 5, (255, 0, 0), -1)
197             cv2.line(oimage, (int(tltrX), int(tltrY)), (int(blbrX), int(blbrY)),
198             color, 2)
199             cv2.line(oimage, (int(tblbX), int(tblbY)), (int(trbrX), int(trbrY)),
200             color, 2)

```

```

198
199     dA = dist.euclidean((tltrX, tltrY), (blbrX, blbrY))
200     dB = dist.euclidean((tblX, tblY), (trbrX, trbrY))
201     pixelsPerMetric = 1
202
203     dimA = dA * pixelsPerMetric
204     dimB = dB * pixelsPerMetric
205
206     cv2.putText(oimage, "{:.1f}pixel".format(dimA), (int(tltrX - 15),
207                                         int(tltrY - 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.65, color,
2)
208     cv2.putText(oimage, "{:.1f}pixel".format(dimB), (int(trbrX + 10),
209                                         int(trbrY)), cv2.FONT_HERSHEY_SIMPLEX, 0.65, color, 2)
210     cv2.putText(oimage, "{:.1f}".format(label), (int(tltrX - 35),
211                                         int(tltrY - 5)), cv2.FONT_HERSHEY_SIMPLEX, 0.65, color,
2)
212
213     return oimage, count

```

ПРИЛОЖЕНИЕ В

Утилитарный модуль

Листинг 12: Конвертирование каналов изображения

```
1 import cv2
2
3
4 def convert_one_channel(img):
5     if len(img.shape) > 2:
6         return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
7
8     return img
9
10
11 def convert_rgb(img):
12     if len(img.shape) == 2:
13         return cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
14
15     return img
```

Листинг 13: Подготовка челюстных масок

```
1 import os
2
3 from natsort import natsorted
4 from PIL import Image
5
6
7 def main():
8     images = natsorted(os.listdir('data/Segmentation1'))
9
10    for img in images:
11        oimg = Image.open(f'data/Segmentation1/{img}')
12        grayed = oimg.convert('L')
13        bw = grayed.point(lambda x: 0 if x < 40 else 255, '1')
14        bw.save(f'masks/mandibles/{img}')
15
16    images = natsorted(os.listdir('masks/mandibles'))
17
18    for img in images:
19        oimg = Image.open(f'masks/mandibles/{img}')
20        oimg = oimg.resize((512, 512))
21        oimg.save(f'masks/mands/{img}')
22
```

```
23
24 if __name__ == '__main__':
25     main()
```

Листинг 14: Визуализация метрик

```
1 import pickle
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5
6
7 def main():
8     with open('../train/trained/teeth.hist', 'rb') as f:
9         data = pickle.load(f)
10
11     plt.rcParams['font.family'] = 'Times New Roman'
12
13     plt.figure(0)
14     plt.plot(data['accuracy'], color='green')
15     plt.title(f"Точность модели максимальное ( значение: {max(data['accuracy'])} )")
16     plt.ylabel('Точность')
17     plt.xlabel('Эпоха')
18     plt.savefig('accuracy.svg', format='svg')
19
20     plt.figure(1)
21     plt.plot(data['loss'], color='red')
22     plt.title(f"Функция потерь минимальное ( значение: {min(data['loss'])} )")
23     plt.ylabel('Потери')
24     plt.xlabel('Эпоха')
25     plt.savefig('loss.svg', format='svg')
26
27     x = ['100%', '75%', '50%', '25%']
28     y1 = [17.5, 13.18, 11.48, 10.46]
29     y2 = [14.4, 12.68, 11.03, 10.37]
30
31     plt.figure(2)
32     plt.bar(x, y1, color='red')
33     plt.bar(x, y2, color='green')
34     plt.title('Зависимость времени работы приложения от размера снимка')
35     plt.xlabel('Размер снимка, % от исходного')
36     plt.ylabel('Время работы приложения, сек.')
37     plt.legend(['С классификацией', 'Без классификации'])
38     plt.savefig('run.svg', format='svg')
39
40
```

```
41 if __name__ == '__main__':
42     main()
```