Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Сыктывкарский государственный университет имени Питирима Сорокина»

(ФГБОУ ВО «СГУ им. Питирима Сорокина»)

Институт точных наук и информационных технологий

Кафедра прикладной информатики

Выпускная квалификационная работа

**Проектирование и разработка приложения для интеллектуальной обработки изображений**

Направление подготовки

09.03.03 Прикладная информатика

Направленность (профиль) программы

Прикладная информатика в экономике

Исполнитель:

Гончаров Игорь Валерьевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Личная подпись

Научный руководитель:

Канд. педагогических наук, доцент

Бабикова Надежда Николаевна \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Личная подпись

Сыктывкар

2023

**Оглавление**

[Аннотация 4](#_Toc136528712)

[Тезаурус 6](#_Toc136528713)

[Введение 7](#_Toc136528714)

[ГЛАВА 1. КОМПЬЮТЕРНОЕ ЗРЕНИЕ И НЕЙРОННЫЕ СЕТИ 8](#_Toc136528715)

[1.1 Понятие компьютерного зрения 8](#_Toc136528716)

[1.2 Понятие нейронной сети 9](#_Toc136528717)

[1.2.1 Искусственные нейронные сети и их составляющие 9](#_Toc136528718)

[1.2.2 Активационная функция нейрона 11](#_Toc136528719)

[1.2.3 Модели нейронов 13](#_Toc136528720)

[1.2.4 Виды архитектур нейронных сетей 16](#_Toc136528721)

[1.3 Обучение нейронных сетей 24](#_Toc136528722)

[1.3.1 Общие понятия в обучении нейронных сетей 24](#_Toc136528723)

[1.3.2 Гиперпараметры нейронных сетей 26](#_Toc136528724)

[1.3.3 Проблемы при обучении нейронных сетей 28](#_Toc136528725)

[1.3.4 Алгоритмы обучения нейронных сетей 32](#_Toc136528726)

[1.3.5 Метрики нейронной сети 33](#_Toc136528727)

[ГЛАВА 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ 36](#_Toc136528728)

[2.1 Постановка задачи 36](#_Toc136528729)

[2.2 Инструментарий: Python, библиотеки TensorFlow, Keras, OpenCV, React, Express 37](#_Toc136528730)

[2.3 Выбор архитектуры нейронной сети 43](#_Toc136528731)

[ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ 47](#_Toc136528732)

[3.1 Подготовка данных 47](#_Toc136528733)

[3.2 Создание и обучение модели 53](#_Toc136528734)

[3.3 Тестирование модели 62](#_Toc136528735)

[3.4 Разработка серверной части 64](#_Toc136528736)

[3.5 Разработка клиентской части 66](#_Toc136528737)

[ГЛАВА 4. МЕНЕДЖМЕНТ И ЭКОНОМИЧЕСКИЙ АНАЛИЗ ПРОЕКТА 69](#_Toc136528738)

[4.1 Проектный менеджмент 69](#_Toc136528739)

[4.2 Оценка затрат проекта 71](#_Toc136528740)

[4.3 Прогноз экономической эффективности проекта 72](#_Toc136528741)

[Заключение 75](#_Toc136528742)

[Список источников 76](#_Toc136528743)

[Приложение 1 78](#_Toc136528744)

**Аннотация**

Целью данной дипломной работы являлось создание программы, обрабатывающей изображения, посредством использования нейронных сетей и библиотек компьютерного зрения.

В работе были изложены основные проблемы, связанные с

данной темой и рассмотрены методы решения этих проблем. Были изучены основные принципы работы нейронных сетей.

Для реализации обработки изображений было решено использовать язык Python, фреймворк TensorFlow, Keras, библиотеку компьютерного зрения OpenCV. В качестве клиентского приложения выступает web-сайт, построенный на библиотеке React. Серверная сторона работает на фреймворке Express.

Работа включает 79 страниц, 17 литературных источников, 6 таблиц, 25 изображений.

Ключевые слова: нейронные сети, обработка изображений, компьютерное зрение, обучение с учителем, сверточная нейронная сеть.

**Abstract**

The purpose of this thesis was to create a program that processes images through the use of neural networks and computer vision libraries.

The paper outlined the main problems related to this topic and considered methods for solving these problems. The basic principles of neural networks were studied.

To implement image processing, it was decided to use Python, TensorFlow framework, Keras, OpenCV computer vision library. The client application is a web site built on the React library. The server side runs on the Express framework.

The work includes 79 pages, contains 17 literary sources, 6 table, 25 images.

Keywords: neural networks, image processing, computer vision, teacher training, convolutional neural network.

**Тезаурус**

**Нейронная сеть** (neural network, искусственная нейронная сеть, нейросеть, ИНС) – математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма [1].

**Нейрон** ­– вычислительная единица нейронной сети, которая получает информацию, производит на ее основе вычисления и передает результат далее.

**Сверточные нейронные сети** (convolutional neural network, CNN, СНС,) – класс нейронных сетей, специализирующихся на обработке данных, представленных в виде матрицы (изображения или видео) [9].

**Введение**

Нейронные сети получили широкое применение в различных областях, в том числе и в обработке изображений. Их использование позволяет решать множество задач, таких как распознавание образов, классификация, сегментация объектов, улучшение качества изображений и другое. Это связано с тем, что нейронные сети способны обрабатывать большие объемы данных и выделять из них закономерности, которые не могут быть выявлены традиционными методами обработки изображений.

В работе рассмотрены основные подходы к обработке изображений с использованием нейронных сетей и библиотек компьютерного зрения, а также методы их обучения и оценки качества. Также рассмотрен процесс создания клиентского приложения для данной системы.

Объектом исследования являются методы улучшения качества изображений на основе нейронных сетей. Предмет исследования – изучение методов создания нейронных сетей.

Целью работы являлось создание нейронной сети (и клиентского приложения для нее) и ее обучение на подготовленном наборе типовых изображений, для дальнейшего ее использования в качестве инструмента улучшения изображений.

**Задачи исследования:**

* анализ предметной области
* проектирование нейронной сети
* реализация нейронной сети на Python при помощи TensorFlow
* обучение сети и оценка результатов
* построение клиентского web-приложения.

# **ГЛАВА 1. КОМПЬЮТЕРНОЕ ЗРЕНИЕ И НЕЙРОННЫЕ СЕТИ**

## **Понятие компьютерного зрения**

**Компьютерное зрение и его составляющие**

Компьютерное зрение (Computer Vision) – это область искусственного интеллекта, которая занимается разработкой методов и алгоритмов для обработки, анализа и интерпретации изображений и видео. Она основывается на использовании математических и статистических методов, а также методов машинного обучения, для создания систем, которые могут распознавать и классифицировать объекты, анализировать сцены и извлекать определенную информацию из изображений [18].

Примерами применения компьютерного зрения могут быть автоматическое распознавание лиц, распознавание номерных знаков автомобилей, анализ медицинских изображений, анализ поведения людей на видео и т.д. Эта область быстро развивается и имеет множество применений в различных отраслях [19].

Наиболее типичные задачи компьютерного зрения включают в себя:

1. Классификация изображений: определение, к какому классу относится изображение.
2. Обнаружение объектов: нахождение и выделение объектов на изображении.
3. Сегментация изображений: разбиение изображения на несколько частей, каждая из которых относится к определенному классу объектов.
4. Оптическое распознавание символов (OCR): распознавание символов на изображении и преобразовании их в текст.
5. Анализ видео: отслеживание движения объектов на видео, распознавание действий на видео и т.д.
6. Распознавание лиц.

**Связанные области**

Компьютерное зрение является общим термином, который охватывает широкий спектр задач. Что-бы конкретизировать решаемые задачи были введены другие термины, обозначающие направления, сфокусированные на определённом предмете изучения [20].

Обработка изображений (Image Processing) – это область, которая занимается обработкой цифровых изображений с целью улучшения их качества, извлечения информации из них, сжатия и т.д.

Машинное зрение (Machine Vision) – это более узкий термин, который обычно относится к применению компьютерного зрения и обработки изображений для автоматического контроля качества, робототехники, автоматизации производства и других промышленных задач.

Визуализация – связана с процессом создания изображений, их обработкой и анализом. Например, рентгенография работает с анализом видеоданных медицинского применения.

Распознавание образов – использует различные методы для получения информации из видеоданных, в основном, основанные на статистическом подходе.

## **1.2 Понятие нейронной сети**

### **1.2.1 Искусственные нейронные сети и их составляющие**

Нейронная сеть представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Нейроны делятся на три основных типа: входной, скрытый и выходной:

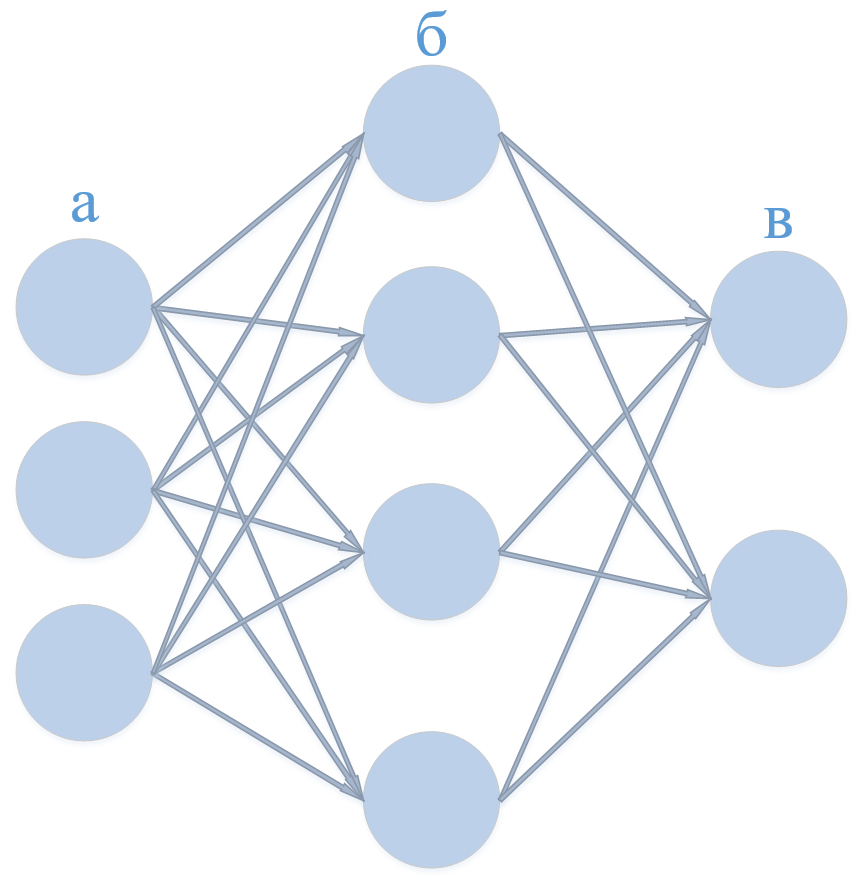


Рисунок 1.1 – Схема простой нейронной сети: а – входные нейроны, б – скрытые нейроны, в – выходные нейроны.

В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Входной слой получает информацию, n скрытых слоев, которые ее обрабатывают и выходной слой, который выводит результат. У каждого из нейронов есть два основных параметра: входные данные и выходные данные.

Возможность обучения – одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей (синапсов) между нейронами.

У синапсов есть только один параметр – вес. Умножаясь на вес, входная информация изменяется, когда передается от одного нейрона к другому. Связи с положительным весом называются возбуждающими, а с отрицательным – тормозящими.

В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение. Это значит, что в случае успешного обучения сеть сможет вернуть верный результат на основании данных, которые отсутствовали в обучающей выборке.

Нейронные сети используются для решения сложных задач, которые требуют аналитических вычислений подобных тем, что делает человеческий мозг. Самыми распространенными применениями нейронных сетей является:

* классификация
* предсказание
* распознавание.

Классификация – распределение данных по параметрам. В случае данной работы – определить по рентгеновскому снимку здоровые легкие у человека или нет.

Предсказание – возможность предсказывать развитие событий. Например, рост или падение акций, основываясь на ситуации на фондовом рынке.

Распознавание – в настоящее время, самое широкое применение нейронных сетей. Используется для определения объектов на изображении, например, выделение людей и животных на фотографии.

### **1.2.2 Активационная функция нейрона**

В теории нейронных сетей активационной называется функция, аргументом которой является взвешенная сумма входов искусственного нейрона, а значением — выход нейрона:

|  |  |  |
| --- | --- | --- |
|  |  | **(1.1)** |

|  |  |  |
| --- | --- | --- |
|  |  | **(1.2)** |

где:

* x — взвешенная сумма входов нейрона;
* N — число входов нейрона;
* Wi — вес i-го входа нейрона;
* Si — значение, поступающее по i-му входу;
* f(x) — активационная функция;
* y — выходное значение нейрона (и, соответственно, активационной функции).

От вида и формы используемой активационный функции зависит выбор алгоритма обучения сети, а также качество ее обучения на конкретном обучающем множестве. Параметры активационной функции подбираются экспериментально в процессе обучения.

Основные активационные функции:

Таблица 1.1 – Основные функции активации

|  |  |  |  |
| --- | --- | --- | --- |
| Название | График | Уравнение | Область значений |
| Единичная ступенька (функция Хевисайда) | C:\Users\Игорь\AppData\Local\Microsoft\Windows\INetCache\Content.Word\heww.png |  | {0, 1} |
| Логистическая (сигмоида или гладкая ступенька) | sig |  | (0, 1) |
| Гиперболический тангенс | C:\Users\Игорь\AppData\Local\Microsoft\Windows\INetCache\Content.Word\th.png |  | (-1, 1) |
| Линейный выпрямитель (ReLU) | C:\Users\Игорь\AppData\Local\Microsoft\Windows\INetCache\Content.Word\relu.png |  | [0, +∞) |

### **1.2.3 Модели нейронов**

#### **Персептрон**

Персептрон — простейший вид нейронных сетей. В основе лежит математическая модель восприятия информации мозгом, состоящая из сенсоров, ассоциативных и реагирующих элементов. В самом общем своем виде он представляет систему из элементов трех разных типов: сенсоров (входных нейронов, S), ассоциативных элементов (скрытых нейронов, A) и реагирующих элементов (выходных нейронов, R).



Рисунок 1.2 – Схема персептрона: а – S-элементы, б – A-элементы, в – R-элементы.

Принцип работы персептрона:

S-элементы находятся либо в состоянии покоя (сигнал равен 0), либо в состоянии возбуждения (сигнал равен 1). Далее сигналы от S-элементов передаются A-элементам по S-A связям. Эти связи могут иметь веса, равные только -1, 0 или 1. Затем сигналы от сенсорных элементов, прошедших по S-A связям попадают в A-элементы, которые еще называют ассоциативными элементами. Стоит заметить, что одному A-элементу может соответствовать несколько S-элементов. Если сигналы, поступившие на A-элемент, в совокупности превышают некоторый его порог, то этот A-элемент возбуждается и выдает сигнал, равный 1. В противном случае (сигнал от S-элементов не превысил порога A-элемента), генерируется нулевой сигнал [3].

Классификация персептронов:

*Персептрон с одним скрытым слоем.* Персептрон, у которого имеется только по одному слою S, A и R элементов.

*Однослойный персептрон.* Каждый S-элемент однозначно соответствует одному A-элементу, все S-A связи имеют вес, равный +1, порог A элементов равен 1.

*Многослойный персептрон.* Под многослойным персептроном понимают два разных вида: многослойный персептрон по Розенблатту и многослойный персептрон по Румельхарту.

* Многослойный персептрон по Розеблатту — персептрон, у которого имеется более 1 слоя А-элементов.
* Многослойный персептрон по Румельхарту — многослойный персептрон по Розенблатту, у которого обучению подлежат еще и S-A связи, а также само обучение производится по методу обратного распространения ошибки.

Даже небольшое изменение весов или смещения одного из персептронов сети может кардинально изменить выходное значение, например, с 0 на 1. Поэтому в современных работах чаще всего используют другую модель искусственного нейрона — сигмоидальный нейрон.

#### **Сигмоидальный нейрон**

Сигмоидальные нейроны похожи на персептроны, однако небольшие

изменения в их весах и смещениях незначительно изменяют выход нейрона. Это достигается путем использования неразрывной сигмоидальной функции активации.

Благодаря этому сеть из сигмоидальных нейронов может обучаться. На вход сигмоидального нейрона подаются любые значения между 0 и 1. На выходе также выдаётся значение между 0 и 1.

Сигмоидальная функция обычно представляется формулой:

|  |  |  |
| --- | --- | --- |
|  |  | **(1.3)** |

Чем больше β (параметр наклона сигмоидальной функции активации), тем сильнее крутизна графика. При β → ∞ сигмоидальная функция превращается в функцию ступенчатого типа, идентичную функции активации персептрона.

Применение непрерывной функции активации позволяет использовать при обучении градиентные методы. Градиентный спуск — метод численной оптимизации, который используется во многих алгоритмах, где требуется найти экстремум функции. Суть градиентного спуска – минимизировать функцию, делая небольшие шаги в сторону наискорейшего убывания функции. Градиентом в нейронных сетях называется вектор частных производных функции потерь по весам нейронной сети. Он указывает на направление наибольшего роста этой функции для всех весов по совокупности. Градиент считается в процессе тренировки нейронной сети и используется в процессе обратного распространения ошибки для корректировки весов [7].

### **1.2.4 Виды архитектур нейронных сетей**

#### **Сеть прямого распространения**

Для решения задачи классификации (обучения с учителем), нейросеть получает на вход множество тренировочных примеров X с метками Y (labels).

Можно представить классическую нейросеть в виде вычислительного графа содержащего:

* входные вершины x
* вершины, являющиеся нейронами со значениями их выхода a
* вершины, отвечающие за bias b
* ребра, умножающие значения выхода предыдущего слоя на соответствующие им коэффициенты матрицы весов w
* гипотезу hw,b(x) — результат выхода последнего слоя.

Нейронная сеть прямого распространения в общем случае строится как соединение множества нейронов, объединенных в слои так, что выходы одного слоя являются входами следующего [5]:

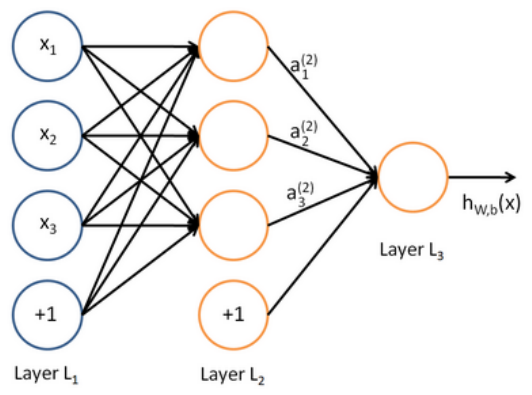


Рисунок 1.3 – Схема сети прямого распространения: Layer L1 – входной слой, Layer L2 – скрытый слой, Layer L3 – выходной слой.

Самый левый слой сети называется входным, самый правый — выходным (на рис. 1.3 он состоит из одного элемента), остальные слои называют скрытыми, потому что их значения отсутствуют в обучающем наборе. Таким образом, данная сеть содержит 3 входных нейрона, 3 скрытых и 1 выходной.

Нейрон смещения (bias нейрон) — его вход и выход всегда равняются 1, и они никогда не имеют входных синапсов. Соединения у нейронов смещения такие же, как у обычных нейронов — со всеми нейронами следующего уровня, за исключением того, что синапсов между двумя bias нейронами быть не может. Следовательно, их можно размещать на входном слое и всех скрытых слоях, но никак не на выходном слое, так как им попросту не с чем будет формировать связь. Нейрон смещения нужен для того, чтобы иметь возможность получать выходной результат, путем сдвига графика функции активации вправо или влево.

Нейросеть параметризируется значениями (*W, b*) = (*W*(1)*,b*(1)*,W*(2)*,b*(2)*,...*), где под понимается параметр, или вес, который отвечает соединению между *j*−м нейроном в слое *l* и *i*−м нейроном в слое *l* + 1. За обозначается смещение или вес, который связан с константными единичными входами на каждом слое сети [4].

Результат применения функции активации обозначается ai для i-ого элемента. Получается следующая система:

|  |  |  |
| --- | --- | --- |
|  |  | **(1.4)** |

Обозначив функцию суммирования через z, получим:

|  |  |  |
| --- | --- | --- |
|  |  | **(1.5)** |

Общая формула будет выглядеть следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | **(1.6)** |

Эти преобразования называются прямым проходом или прямым распространением (forward propagation).Сетью прямого распространения называются нейронные сети, которые используют выход одного слоя в качестве входных данных для следующего слоя.

У сети прямого распространения есть существенный недостаток – слишком много параметров. Например, нейросеть из 3 скрытых слоев, которой нужно обрабатывать картинки 100\*100 пикселей на входе будет иметь 10 000 пикселей, передающихся на 3 слоя. Иными словами, каждый нейрон такой сети получает на вход все пиксели изображения. В конечном итоге такая нейросеть будет иметь порядка миллиона параметров, т.е. классифицировать примеры по миллиону признаков.

Данный недостаток сетей прямого распространения исправлен в сверточных нейросетях.

#### **Сверточная сеть**

Сверточные нейронные сети объединяют три архитектурных идеи, для обеспечения инвариантности к изменению масштаба, повороту сдвигу и пространственным искажениям:

* локальные рецепторные поля (обеспечивают локальную двумерную связность нейронов)
* общие коэффициенты нейронов (обеспечивают детектирование некоторых черт в любом месте изображения и уменьшают общее число весовых коэффициентов)
* иерархическая организация с пространственными подвыборками.

В сети прямого распространения каждый нейрон связан с каждым пикселем изображения, в то время как в сверточной сети происходит разбиение – каждый нейрон связан только с частью изображения. Достигается это посредством свертки изображения:

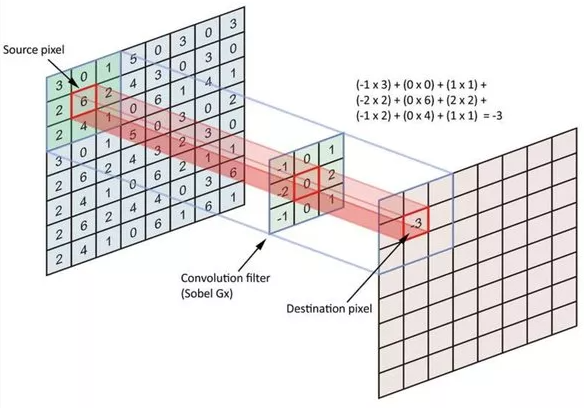


Рисунок 1.4 – Свертка изображения: Source pixel – исходный пиксель, convolution filter – ядро свертки, destination pixel – пиксель свернутого изображения.

Происходит свертка посредством пропуска изображения через ядро свертки. Ядро свертки – это совокупность весов данного нейрона. Применяется ядро свертки на всех пикселях изображения последовательно. Ядро представляет из себя некий фильтр, который проходится по всей области предыдущей карты и находит определенные признаки объектов. Размер ядра обычно берется в пределах от 3х3 до 7х7. Если размер ядра маленький, оно не сможет выделить какие-либо признаки, если слишком большое, увеличится количество связей между нейронами.

В зависимости от метода обработки краев исходной матрицы результат может быть меньше исходного изображения (valid), такого же размера (same) или большего размера (full):

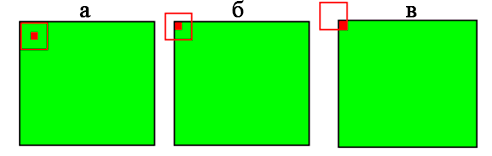


Рисунок 1.5 – Виды свертки: а – valid, б – same, в – full.

СНС состоит из нескольких видов слоев: сверточные (convolutional) слои, подвыборочные (sub-sampling) слои и слои нейронов (например, персептронов):

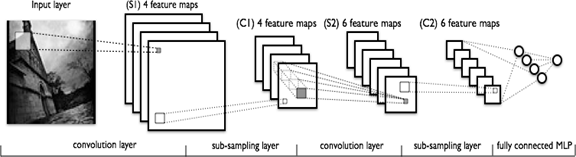


Рисунок 1.6 – Структура СНС.

Как видно из структуры, свертка изображения происходит несколько раз, формируя входной вектор признаков для многослойного персептрона.

Сверточный слой представляет из себя набор карт (матриц), у каждой из которых есть синаптическое ядро. Размер ядра выбирается таким, чтобы размер карт сверточного слоя был четным, это позволяет не терять информацию при уменьшении размерности в подвыборочном слое.

Размер всех карт сверточного слоя одинаков, а их количество карт определяется требованиями к задаче. Большое количество карт повышает качество распознавания, но увеличивает сложность вычислений. Часто на практике используется соотношение один к двум, то есть каждая карта предыдущего слоя связана с двумя картами сверточного слоя.

Разделяемые веса – это одна из главных особенностей сверточной нейросети. В обычной многослойной сети очень много связей между нейронами, что сильно замедляет процесс распознования. В сверточной сети – наоборот, общие веса позволяет сократить число связей и позволить находить один и тот же признак по всей области изображения.

Изначально значения каждой карты сверточного слоя равны 0. Значения весов ядер задаются случайным образом в области от -0.5 до 0.5.

Подвыборочный слой также, как и предшествующий сверточный имеет карты, но их количество совпадает с предыдущим слоем. Цель этого слоя – уменьшение размерности карт предыдущего. Если на предыдущей операции свертки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. Кроме того, фильтрация уже ненужных деталей помогает не переобучаться.

Обычно, каждая карта имеет ядро размером 2x2, что позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Вся карта признаков разделяется на ячейки 2х2 элемента, из которых выбираются максимальные по значению (операция MaxPooling):

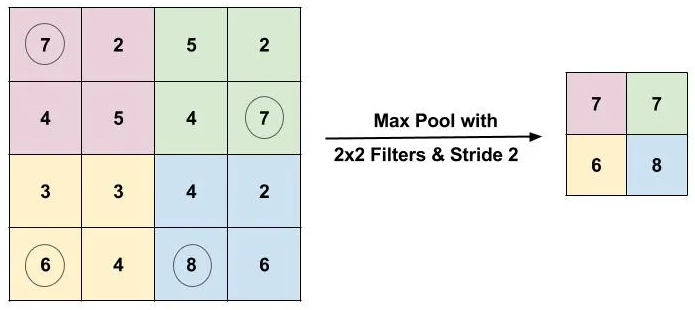


Рисунок 1.7 – Преобразование карты светрочного слоя ядром подвыборочного слоя.

Определение топологии сети ориентируется на решаемую задачу. Можно выделить следующие действия, влияющие на выбор топологии:

* определение решаемой задачи (классификация, прогнозирование, модификация);
* определение ограничений в решаемой задаче (скорость, точность ответа);
* определение входных и выходных данных (входные: изображение, звук, размер: 100x100, 30x30, формат: RGB, в градациях серого, выходные данные – количество классов)

На данный момент сверточная нейронная сеть и ее модификации считаются лучшими по точности и скорости алгоритмами нахождения объектов на изображении.

#### **Глубокие нейронные сети**

Глубокая нейронная сеть (deep neural network) [11] – это искусственная нейронная сеть с несколькими уровнями между входным и выходным уровнями. Каждый нейрон в одном слое соединяется со всеми нейронами следующего слоя. Один или несколько слоев между входным и выходным слоями называются скрытыми слоями.



Рисунок 1.8 – Глубокая нейронная сеть.

Поскольку каждый скрытый слой вычисляет нелинейное преобразование предыдущего слоя, глубокая сеть может представлять значительно более сложные функции, чем малослойная. При обучении глубокой сети важно использовать нелинейную функцию активации в каждом скрытом слое. Это связано с тем, что множество слоев линейных функций сами вычисляли бы только линейную функцию ввода и, следовательно, не были бы более выразительными, чем применение только одного скрытого слоя.

## **1.3 Обучение нейронных сетей**

### **1.3.1 Общие понятия в обучении нейронных сетей**

Обучение нейронной сети – это процесс, при котором параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена.

Общие понятия:

Эпоха (epoch) – прохождение всего набора данных через нейронную сеть в прямом и обратном направлении один раз. Малое число эпох приводит к недостатку обучения, а избыток – к переобучению. Обычно число эпох выбирают в диапазоне от 5 до 30.

Размер серии (batches) – количество тренировочных примеров для одной итерации прямого и обратного проходов. Так как одна целая эпоха слишком велика для памяти компьютера, набор делят на маленькие партии.

Количество итераций (iterations) – число batches, необходимых для завершения одной эпохи. Если набор из 2000 объектов делить на серии по 500 объектов, то для завершения одной эпохи потребуется 4 итерации.

#### **Обучение с учителем и без учителя**

Обучение с учителем предполагает наличие полного набора размеченных данных для тренировки модели на всех этапах ее построения.

Наличие полностью размеченного набора означает, что каждому примеру в обучающем наборе соответствует ответ, который нейросеть должна получить. Каждый образец из набора подается на вход сети, проходит обработку внутри структуры нейросети, вычисляется выходной сигнал сети, который сравнивается с соответствующим значением целевого вектора, представляющего собой требуемый выход сети [12]. Затем вычисляется ошибка, и происходит изменение весовых коэффициентов связей внутри сети до тех пор, пока ошибка по всему обучающему массиву не достигнет приемлемо низкого уровня. Этот процесс называется “метод обратного распространения ошибок”. Есть, конечно, и другие методы, но этот используют чаще остальных. Изначально веса задаются в случайном порядке.

Схематично процесс обучения можно представить так:

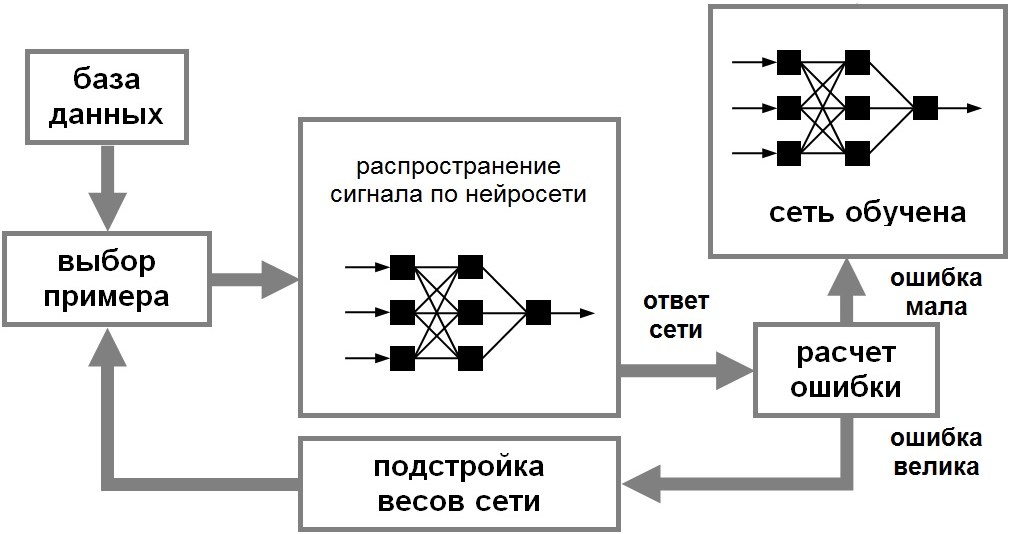


Рисунок 1.9 – Обучение нейронной сети с учителем.

В основном обучение с учителем применяется для решения задач классификации и регрессии.

В задачах классификации результатом работы нейронной сети являются дискретные значения, соответствующие номерам классов, к которым принадлежат объекты. Задачи регрессии связаны с непрерывными данными. Например, линейная регрессия, вычисляет ожидаемое значение переменной y, учитывая конкретные значения x.

Обучение с учителем больше всего подходит для задач, когда имеется большой набор маркированных данных для обучения сети. На практике такой набор есть далеко не всегда. В таком случае и применяется обучение без учителя.

При обучении без учителя набор данных состоит лишь из входных векторов, без подписей. Нейронная сеть самостоятельно находит корреляции в данных, извлекая полезные признаки и анализируя их. Далее сеть подстраивает веса сети так, чтобы получались согласованные выходные векторы, то есть чтобы предъявление достаточно близких входных векторов давало одинаковые выходы. Процесс обучения без учителя выделяет статистические свойства обучающего множества и группирует сходные векторы в классы самостоятельно.

Кластеризация – наиболее распространенная задача для обучения без учителя. Алгоритм подбирает похожие данные, находя общие признаки, и группируют их вместе.

Обучение с частичным привлечением учителя – обучающий набор содержит как размеченные, так и неразмеченные данные. Этот метод особенно полезен, когда трудно извлечь из данных важные признаки или разметить все объекты – трудоемкая задача.

### **1.3.2 Гиперпараметры нейронных сетей**

Помимо весов, определенных в нейросети, обучающим алгоритмам нужен ряд дополнительных параметров.

Гиперпараметры – это настраиваемые параметры, позволяющие управлять процессом обучения модели. Например, определение количество скрытых слоев и количество узлов в каждом слое. Производительность модели в значительной степени зависит от гиперпараметров.

Настройка гиперпараметров, также называемая оптимизациейгиперпараметров. Это процесс поиска конфигурации гиперпараметров, приводящей к лучшей производительности. Этот процесс обычно требует значительных вычислительных ресурсов и выполняется вручную.

#### **Общие гиперпараметры**

Для большинства нейронных сетей характерны следующие параметры [13], [14]:

*Темп обучения*. Модели обычно обучаются оптимизатором стохастического градиентного спуска. Существует несколько вариаций: AdaGrad, RMSProp, Adam и другие. Все они позволяют установить скорость обучения. Этот параметр сообщает оптимизатору, как далеко нужно переместить веса в направлении, противоположном градиенту для мини-партии. Если скорость обучения низкая, то обучение будет более качественным, но оптимизация займет много времени, потому что шаги к минимуму функции потерь маленькие. Если скорость обучения высока, то обучение может не сходиться или даже расходиться. Изменения веса могут быть настолько значительными, что оптимизатор пересекает минимум и усугубляет потерю. Обучение должно начинаться с относительно большой скорости обучения, потому что вначале случайные веса далеки от оптимальных, и затем скорость обучения может уменьшаться во время тренировки, чтобы обеспечить более детальные обновления веса.

*Ранняя остановка*. Это техника оптимизации, которая реализована путем вычисления потери (ошибки) проверки. Если отклонение проверки не падает ниже определенного количества итераций, то модель останавливает свое обучение. Проверочное множество формируется независимо от обучающего и тестового множеств, и используется для проверки предсказательной способности модели после ее обучения и тестирования, а также для оптимизации ее сложности.

*Регуляризация*. Методы регуляризации используются для уменьшения погрешности путем соответствующей подгонки функции к данному тренировочному набору, чтобы избежать переобучения, о котором будет написано далее.

Кроме перечисленного, к гиперпараметрам также относят эпохи, размеры серий и количество итераций.

### **1.3.3 Проблемы при обучении нейронных сетей**

#### **Взрывающийся и затухающий градиент**

В процессе обратного распространения ошибки, при прохождении через слои нейронной сети, в элементах градиента могут накапливаться большие значения, что приводит к сильным изменениям весов. В таком случае элементы градиента могут переполнить тип данных, в котором они хранятся. Такое явление называется взрывающимся градиентом (exploding gradient) [8].

Существует аналогичная обратная проблема, когда в процессе обучения, при обратном распространении ошибки, градиент становится все меньше. Это приводит к тому, что веса при обновлении изменяются на слишком малые значения, и обучение проходит неэффективно или останавливается, то есть алгоритм обучения не сходится. Это явление называется затухающим градиентом (vanishing gradient).

Такая проблема может возникнуть при использовании нейронных сетях классической функцией активации сигмоиды.

Возникновение проблемы взрывающегося градиента можно определить по следующим признакам:

* высоком значение функции потерь
* сильные скачки значения функции потерь
* значение функции потерь принимает значение NaN
* веса модели растут экспоненциально
* веса модели принимают значение NaN.

Признаки затухающего градиента:

* точность модели растет медленно, при этом возможно срабатывание механизма ранней остановки, так как алгоритм может решить, что дальнейшее обучение не будет оказывать существенного влияния
* градиент ближе к концу показывает более сильные изменения, в то время как градиент ближе к началу почти не показывает никакие изменения
* веса модели уменьшаются экспоненциально во время обучения
* веса модели стремятся к 0 во время обучения.

Способы устранения проблемы:

1. Использование другой функции активации.
2. Изменение модели. Для решения проблемы может оказаться достаточным сокращение числа слоев. Это связано с тем, что частные производные по весам растут экспоненциально в зависимости от глубины слоя.
3. Использование Residual blocks. В данной конструкции вывод нейрона подается как следующему нейрону, так и нейрону на расстоянии 2-3 слоев впереди, который суммирует его с выходом предшествующего нейрона, а функция активации в нем – ReLU. Такая связка называется shortcut. Это позволяет при обратном распространении ошибки значениям градиента в слоях быть более чувствительным к градиенту в слоях, с которыми связаны с помощью shortcut, то есть расположенными несколько дальше следующего слоя.
4. Регуляризация весов. Регуляризация заключается в том, что слишком большие значения весов будут увеличивать функцию потерь. Таким образом, в процессе обучения нейронная сеть помимо оптимизации ответа будет также минимизировать веса, не позволяя им становиться слишком большими.
5. Обрезание градиента. Обрезание заключается в ограничении нормы градиента. То есть если норма градиента превышает заранее выбранную величину *T*, то следует масштабировать его так, чтобы его норма равнялась этой величине.

#### **Переобучение**

Переобучение (overfitting) нейронной сети состоит в следующем: модель хорошо классифицирует только примеры из обучающей выборки, адаптируясь к обучающим примерам, вместо того чтобы учиться классифицировать примеры, не участвовавшие в обучении (теряет способность к обобщению).

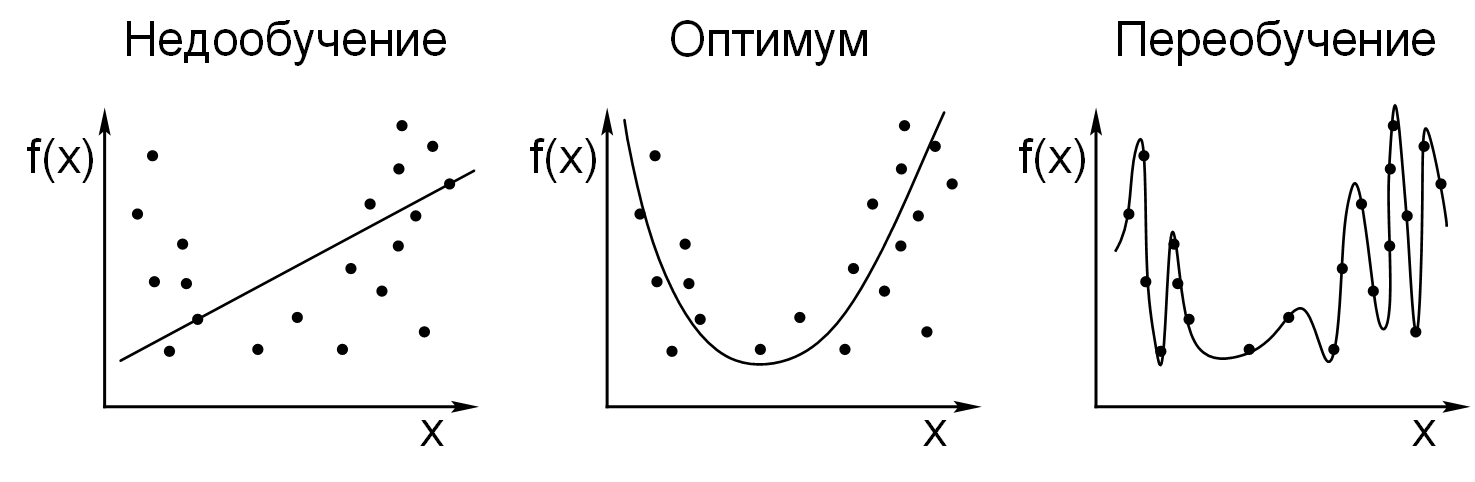


Рисунок 1.10 – Качество обучения.

Это связано с тем, что в процессе обучения модели в обучающей выборке обнаруживаются некоторые случайные закономерности, которые отсутствуют в генеральной совокупности. Появляется слишком много признаков. Например, на изображениях, помимо рассматриваемого объекта, часто присутствуют другие случайные объекты, не относящиеся к исследованию. Нейросеть может начать рассматривать такие объекты как новые параметры изображения.

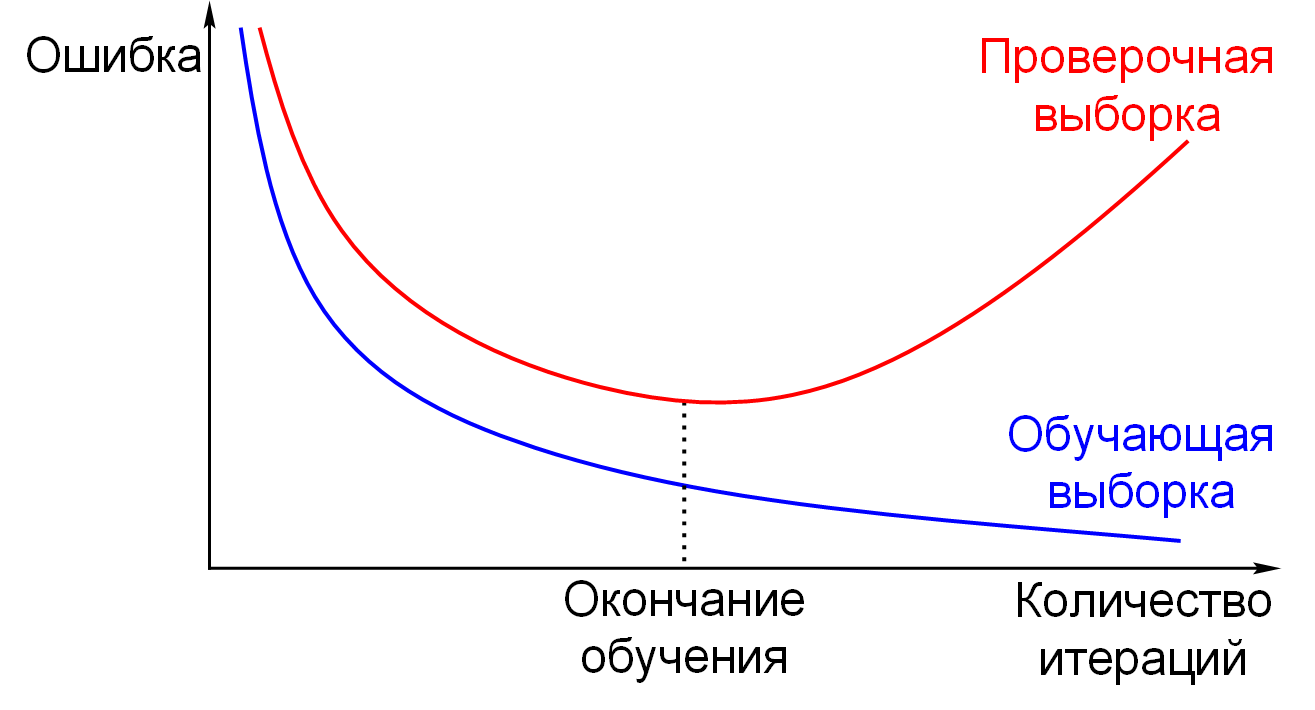


Рисунок 1.11 – График ошибки переобученной сети.

Причины переобучения:

* слишком мало данных для обучения, либо слишком много слоев
* слишком долгое обучение – модель находит закономерности в шуме
* плохо подготовленные данные.

Методы предотвращения переобучения:

* перекрёстная проверка
* регуляризация
* ранняя остановка
* априорная вероятность
* байесовское сравнение моделей
* метод исключения (dropout).

Стоит отметить, что даже тогда, когда обученная модель не имеет чрезмерного количества параметров, можно ожидать, что эффективность её на новых данных будет ниже, чем на данных, использовавшихся для обучения.

Из всех решений данной проблемы лучшие результаты показывает метод dropout [10]. Главная идея Dropout – вместо обучения одной сети обучить сразу несколько, а затем усреднить полученные результаты.

Сети для обучения получаются с помощью исключения из сети (dropping out) некоторых нейронов. Исключение нейрона означает, что при любых входных данных или параметрах он возвращает 0. Исключенные нейроны не вносят свой вклад в процесс обучения ни на одном из этапов алгоритма обратного распространения ошибки, поэтому исключение хотя бы одного из нейронов равносильно обучению новой нейронной сети.

Dropout хорошо работает на практике, потому что предотвращает взаимоадаптацию нейронов на этапе обучения.

### **1.3.4 Алгоритмы обучения нейронных сетей**

Метод обратного распространения является одним из основных способов обучения, но есть и другие алгоритмы [3], [12]:

*Метод упругого распространения (Resilient propagation).* Этот метод был предложен как альтернатива методу обратного распространения, который требует слишком много времени и становится неудобным, если результаты нужно получить в короткие сроки. Для подгонки весовых коэффициентов он использует лишь знаки производных частного случая функции потери. Если производная меняет свой знак на противоположный, то это говорит о слишком большом изменении и о упущении локального минимума. Следовательно, нужно возвратить весу предыдущее значение и уменьшить величину изменения. Если же знак остался прежним, то следует поднять величину изменения веса для максимальной сходимости.

*Генетический алгоритм обучения (Genetic Algorithm).* Это эвристический алгоритм, заключающийся в случайном подборе, комбинировании и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Идея генетических алгоритмов основана на эволюционной теории.

Процесс начинается с набора особей, который называется популяцией. Каждая особь – это решение проблемы, которая была поставлена. Особь характеризуется набором параметров, которые называют генами. Гены объединены в одну строку и формируют хромосому – решение задачи.

В глубоком обучении с помощью генетических алгоритмов оптимизируются параметры нейросети. В качестве особей популяции выступают сами параметры. Этот алгоритм был разработан для решения задач, в которых пространство решений крайне велико и использование других алгоритмов занимает слишком много времени.

### **1.3.5 Метрики нейронной сети**

В бинарной классификации каждая выборка относится к одному из двух классов. Обычно им присваиваются такие метки, как 1 и 0, или положительный и отрицательный (Positive и Negative).

Для получения дополнительной информации о характеристиках модели используется матрица ошибок (confusion matrix). Матрица ошибок помогает нам визуализировать, ошибки модели при различении двух классов. Названия строк представляют собой эталонные метки, а названия столбцов — предсказанные:

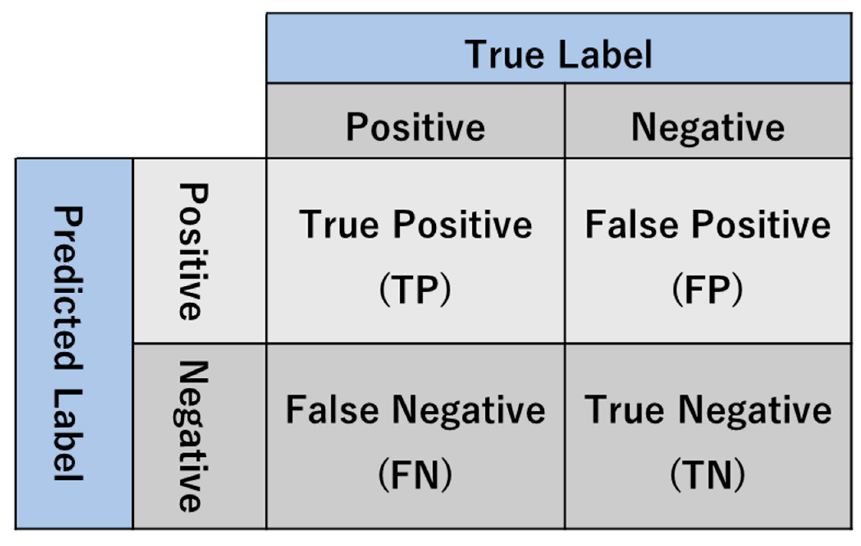


Рисунок 1.12 – Матрица ошибок

Четыре метрики в матрице ошибок представляют собой следующее:

1. Верхний левый элемент (True Positive): сколько раз модель правильно классифицировала Positive как Positive?
2. Верхний правый (False Negative): сколько раз модель неправильно классифицировала Positive как Negative?
3. Нижний левый (False Positive): сколько раз модель неправильно классифицировала Negative как Positive?
4. Нижний правый (True Negative): сколько раз модель правильно классифицировала Negative как Negative?

Матрица ошибок предлагает четыре индивидуальных показателя. На их основе можно рассчитать другие метрики, которые предоставляют дополнительную информацию о поведении модели:

* Accuracy
* Precision
* Recall

Accuracy — это показатель, который описывает общую точность предсказания модели по всем классам. Он рассчитывается как отношение количества правильных прогнозов к их общему количеству. Метрика accuracy может быть не объективной, если данные несбалансированны. Например, есть всего 600 единиц данных, из которых 550 относятся к классу Positive и только 50 — к Negative. Поскольку большинство семплов принадлежит к одному классу, accuracy для этого класса будет выше, чем для другого.

|  |  |  |
| --- | --- | --- |
|  |  | **(2.1)** |

Precision представляет собой отношение числа экземпляров, верно классифицированных как Positive, к общему числу выборок с меткой Positive (распознанных правильно и неправильно).

Precision измеряет точность модели при определении класса Positive. Когда модель делает много неверных Positive классификаций, это увеличивает знаменатель и снижает precision.

Precision отражает, насколько надежна модель при классификации Positive-меток. Цель precision – классифицировать все Positive экземпляры как Positive, не допуская ложных определений Negative как Positive.

|  |  |  |
| --- | --- | --- |
|  |  | **(2.2)** |

Recall рассчитывается как отношение числа Positive выборок, корректно классифицированных как Positive, к общему количеству Positive экземпляров. Recall измеряет способность модели обнаруживать выборки, относящиеся к классу Positive. Чем выше recall, тем больше Positive экземпляров было найдено.

|  |  |  |
| --- | --- | --- |
|  |  | **(2.3)** |

# **ГЛАВА 2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ**

## **2.1 Постановка задачи**

Обработка фотографий, как правило, занимает много времени и требует от человека знаний соответствующих прикладных программ. Сейчас есть возможность ускорить и автоматизировать этот процесс посредством применения нейронных сетей.

Целью данной работы является построение модели сверточной нейронной сети и веб-приложения для нее, которая будет обучаться на парах (исходное изображение и его ухудшенная версия) и в дальнейшем улучшать изображения.

**Функциональные требования к системе (рис. 2.1):**

1. Предоставление пользователю возможности загрузить изображение.
2. Провести колоризацию, сегментацию изображения, увеличить разрешение и сделать его более четким.
3. Сохранить полученное изображение.

**Не функциональные требования:**

1. Наличие персонального компьютера, имеющего доступ к интернету.
2. В случае невозможности загрузить изображение, выдавать сообщение об ошибке.
3. Работа на любом устройстве при наличии браузера.
4. Простота в эксплуатации.

**Программные требования:**

Сервер с поддержкой python 3 и платформы Node.js.



Рисунок 2.1 –Диаграмма вариантов использования системы обработки изображений.

## **2.2 Инструментарий: Python, библиотеки TensorFlow, Keras, OpenCV, React, Express**

#### **Язык Python**

Для реализации системы был выбран язык программирования Python версии 3.10. Этот язык чаще других используется для создания нейронных сетей. Данный факт обуславливается простым синтаксисом, гибкостью и хорошим инструментарием в виде большого числа библиотек. Использование Python позволяет полностью сосредоточиться на решаемых задачах, не отвлекаясь на технические аспекты языка. Выбор данной версии связан с поддержкой всех необходимых зависимостей.

Множество библиотек Python помогают существенно уменьшить количество времени, необходимого для разработки нейронной сети. В данной работе основными библиотеками являются TensorFlow, Keras, OpenCV.

#### **TensorFlow и Keras**

TensorFlow – это библиотека с открытым исходным кодом, объединяющая в себе множество различных алгоритмов и моделей, позволяющая реализовать глубокие нейронные сети для использования в таких задачах, как распознавание и классификация изображений, обработка естественного языка. Она позволяет создавать и обучать нейронные сети различной архитектуры для обнаружения и распознавания образцов и поиска взаимосвязей. TensorFlow также включает в себя TensorBoard, который представляет собой средство визуализации в браузере для оценки эффективности обучения и сетевых параметров модели.

TensorFlow достигает своей производительности благодаря распараллеливанию задач между центральным и графическими процессорами. Ядро каждой операции реализовано на C++ с применением библиотек Eigen и cuDNN для лучшей производительности.

Каждое вычисление в TensorFlow представляется как граф потока данных, он же граф вычислений. Граф вычислений является моделью, описывающей как, будут выполняться вычисления. Граф состоит из плэйсхолдеров (tf.Placeholder), переменных (tf.Variable) и операций. В нём производится вычисление тензоров — многомерных массивов, которые, впрочем, могут быть числом или вектором.

Графы выполняются в сессиях (tf.Session). Существуют два типа сессий — обычные и интерактивные (tf.InteractiveSession); интерактивная сессия подходит для выполнения в консоли. Сессия хранит состояние переменных (Variables) и очередей (queues). Явное создание сессий и графов гарантирует надлежащее освобождение ресурсов памяти.

В графе каждая вершина имеет 0 или больше входов и 0 или больше выходов, и представляет собой реализацию операции. Тензоры представляют собой рёбра графа, а именно массивы произвольного размера (тип массива указывается во время построения графа). Особые вершины, управляющие зависимости (control dependencies), также могут быть в графе: они указывают, что исходный узел для контрольной зависимости должен закончить выполнение до того, как узел получателя контрольной зависимости начнет выполняться.

Каждая операция имеет название и представляет собой абстрактное вычисление (например, суммирование). У операции могут быть аттрибуты: например, возможность сделать операцию полиморфной для разных типов 45 тензоров. Ядро — специфическая реализация операции, которая может выполнена на определенном типе устройства (центральный или графический процессор).

Переменная — особый вид операции, возвращающий указатель на постоянно меняющийся тензор: такая переменная не исчезает после единичного использования графа. Указатели на подобные тензоры передаются многочисленным операциям, которые затем изменяют указанный тензор.

В задачах машинного обучения, параметры модели обычно хранят тензоры в переменных, которые обновляются на каждом шаге обучения. Данная работа выполнена на единственном устройстве с применением CPU.

Особенности В Tensorflow существуют несколько форм параллелизма:

* Параллелизм в отдельных операциях (например, tf.nn.conv2d () и tf.matmul ()). Эти операции имеют эффективные параллельные реализации для многоядерных процессоров и графических процессоров, и TensorFlow использует эти реализации во всех возможных случаях;
* Параллелизм между операциями. TensorFlow использует представление графа вычислений и там, где есть два узла, которые не связаны прямым путем, они могут выполняться параллельно;
* Параллелизм между копиями моделей. Стандартная схема для параллельного обучения — разделить данные между workers, провести одинаковые вычисления для разных данных и обменяться параметрами модели между workers.

Keras — это API (программный интерфейс приложения) глубокого обучения на Python, которое облегчает использование TensorFlow. Keras позволяет реализовать множество мощных, но зачастую сложных функций TensorFlow максимально просто, к тому же он настроен для работы с Python без каких-либо серьезных изменений или настроек.

Концепции глубокого обучения в Keras представлены следующим образом:

* слои, комбинируемые в модель
* функция потерь, которая определяет обратную связь обучения
* оптимизатор, определяющий как будет продвигаться обучение
* метрики для вычисления производительности модели
* цикл обучения, производящий градиентный спуск для подвыборки.

**OpenCV**

OpenCV (Open Source Computer Vision) — это библиотека компьютерного зрения с открытым исходным кодом, которая позволяет быстро и эффективно обрабатывать изображения и видео.

Основные возможности OpenCV:

1. Обработка изображений и видео – увеличение резкости, размытие, выделение граней, фильтрацию, морфологические операции, преобразования и многое другое.
2. Обнаружение и распознавание объектов.
3. Машинное обучение – включает в себя множество алгоритмов машинного обучения, которые можно использовать для классификации, сегментации и других задач.
4. Калибровка камеры – позволяет улучшить точность распознавания объектов и сегментации изображений.

#### **Jupyter Notebook**

Для визуализации данных в работе применяется Jupyter Notebook. Это веб-приложение с открытым исходным кодом позволяющее создавать “блокноты” – текстовые файлы, который можно редактировать в браузере. Можно совмещать код на Python и формат Markdown, чтобы комментировать свои действия. Блокнот так же позволяет разбить длинный код на меньшие части, которые могут выполняться независимо. Это делает разработку более интерактивной, не нужно перезапускать весь предыдущий код, если необходимо изменить что-то в одном из фрагментов.

Jupyter Notebook позволяет хорошо визуализировать данные с помощью Python библиотеки Matplotlib. Это кроссплатформенная библиотека для создания 2D графиков из данных в массивах. Кроме того, Jupyter Notebook сохраняет все выводы, в том числе и изображения, до следующего запуска кода, что очень удобно.

#### **React**

React – это JavaScript-библиотека для разработки пользовательских интерфейсов. Она была создана компанией Facebook и предназначена для создания масштабируемых и эффективных веб-приложений.

Основной принцип React состоит в построении интерфейса из отдельных компонентов. Каждый компонент отвечает за отображение определенной части пользовательского интерфейса и имеет свое состояние. Компоненты могут быть вложенными, что позволяет создавать сложные иерархии интерфейса.

Для работы с React используется JSX — расширение синтаксиса JavaScript, которое позволяет описывать структуру интерфейса в виде компонентов прямо в JavaScript-коде. JSX сочетает в себе возможности HTML и JavaScript, делая разработку интерфейса более декларативной и удобной.

React является мощным инструментом для создания современных веб-приложений, обладающим высокой производительностью, модульностью и переиспользуемостью компонентов. Благодаря широкому сообществу разработчиков и активной поддержке со стороны Facebook, React остается одним из наиболее популярных выборов для разработки фронтенда.

#### **Express**

Express – это минималистичный и гибкий веб-фреймворк для Node.js, который позволяет разрабатывать веб-приложения и API. Он предоставляет простой и интуитивно понятный способ создания серверных приложений, обрабатывая маршрутизацию запросов, управление сессиями, шаблонизацию и другие задачи, связанные с веб-разработкой.

В данной работе он необходим для загрузки пользовательских изображений на сервер и отправки их обратно, после обработки.

#### **Прочие библиотеки**

NumPy (Numerical Python) - это библиотека для языка программирования Python, которая добавляет поддержку многомерных массивов и матриц, а также большой набор функций для работы с ними. Она является одной из основных библиотек для научных вычислений в Python. С помощью NumPy можно быстро и эффективно выполнять математические операции над массивами и матрицами, такие как умножение, сложение, вычитание, деление и другие. В данной работе это поможет удобно взаимодействовать с изображениями как с матрицами и векторами.

Pandas и seaborn вместе помогают строить более красивые и информативные графики, позволяя не ограничиваться возможностями Matplotlib.

## **2.3 Выбор архитектуры нейронной сети**

Существует множество архитектур нейронных сетей, которые используются для обработки изображений. Самые используемые:

1. Сверточные нейронные сети (Convolutional Neural Networks, CNN): это наиболее распространенный тип нейронных сетей для обработки изображений. О них было сказано ранее.
2. Рекуррентные нейронные сети (Recurrent Neural Networks, RNN): это тип нейронных сетей, который обрабатывает последовательности данных, включая последовательности изображений. Они используются, например, для анализа видео и обнаружения движущихся объектов.
3. Автокодировщики (Autoencoders): это тип нейронных сетей, который может использоваться для сжатия изображений или для удаления шума из изображений. Они могут также использоваться для генерации новых изображений.
4. Глубокие вероятностные модели (Deep Probabilistic Models): могут использоваться, например, для генерации новых изображений на основе обучающего набора.
5. Сети преобразований (Transformers): это тип нейронных сетей, который позволяет моделировать зависимости между различными частями изображения. Они обычно используются для задач обработки естественного языка, но также могут быть применены к изображениям.
6. Генеративно состязательные сети (Generative Adversarial Networks, GAN): это тип нейронных сетей, который используется для генерации новых изображений. Они работают путем создания двух нейронных сетей – генератора и дискриминатора, которые влияют на весовые коэффициенты друг друга.

Стоит отметить, что большинство современных архитектур сетей для работы с изображениями являются сверточными, а также могут включать в себя и некоторые другие элементы разных архитектур.

Для решения задач, поставленных в данной работе, была выбрана архитектура U-Net.

U-Net – это сверточная нейронная сеть, которая была разработана для решения в 2015 году для сегментации биомедицинских изображений в отделении Computer Science Фрайбургского университета. Архитектура сети представляет собой полносвязную свёрточную сеть, модифицированную так, чтобы она могла работать с меньшим количеством примеров и делала более точную сегментацию.

В ходе использования выяснилось, что данная архитектура подходит для решения и множества других задач. Она получила свое название из-за своей особенной формы: сеть имеет форму буквы "U":

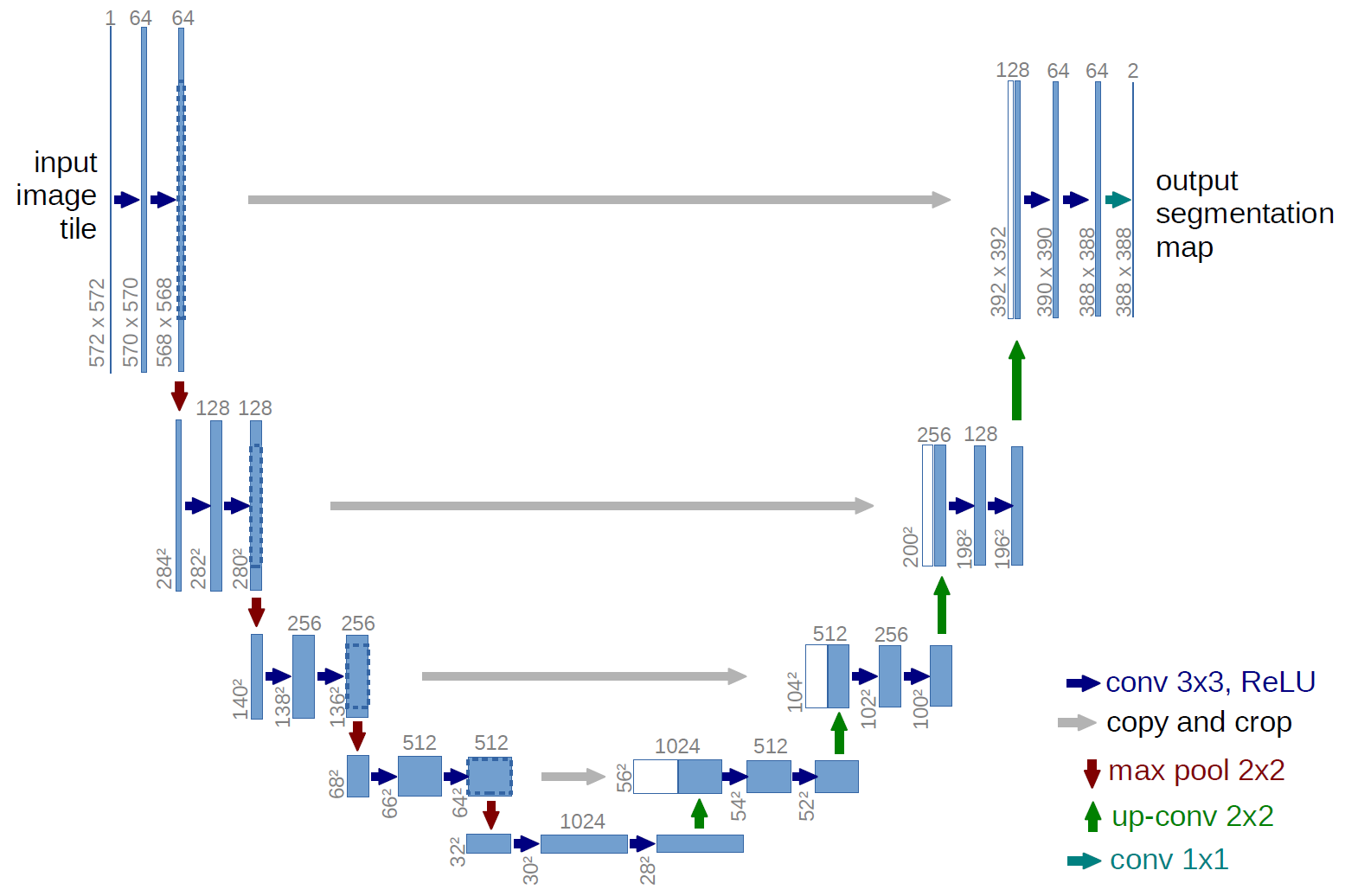


Рисунок 2.2 – Схема архитектуры U-Net.

Как видно из рис. 2.2, архитектура U-Net состоит из двух основных частей: сжимающего пути (encoder) и расширяющего пути (decoder).

Сжимающий путь состоит из сверточных слоев, а также слоев объединения (pooling layers), которые позволяют уменьшить размерность входных изображений и выделить важные признаки. Затем эти признаки передаются в расширяющий путь.

Расширяющий путь состоит из слоев деконволюции (deconvolution layers) и слоев объединения (upsampling layers), которые позволяют увеличить размерность входных данных и восстановить карту сегментации. В конце расширяющего пути добавляется последний слой, который используется для получения итоговой карты сегментации.

В отличие от простого увеличения размера изображения, U-Net использует информацию о признаках из энкодера, чтобы создать более точное и реалистичное изображение. Это достигается благодаря соединительным мостам между энкодером и декодером, которые позволяют использовать информацию о признаках, полученных на более ранних этапах, для создания более точных результатов.

# **ГЛАВА 3. РЕАЛИЗАЦИЯ СИСТЕМЫ**

## **3.1 Подготовка данных**

Создание обучающей выборки – самый трудный этап в машинном обучении. Для сбора качественных данных обычно требуется много времени, поэтому для обучения сетей были выбраны готовые наборы изображений с сайта Kaggle. Kaggle – это сайт для специалистов по обработке данных и машинному обучению, куда каждый желающий может загрузить свой набор данных.

**Набор для увеличения разрешения – Image Super Resolution**

Набор содержит по 855 примеров различных изображений хорошего и плохого качества: high\_res и low\_res.

Данные в наборах обычно требуют небольшой подготовки, особенно в случае изображений. Эта подготовка направлена на облегчение процесса оптимизации сети и максимизацию вероятности получения хороших результатов.

Для унификации изображений, их необходимо перевести в RGB и масштабировать к одинаковому размеру (обычно это квадрат со сторонами в диапазоне 140-255 пикселей). Это можно сделать при помощи OpenСV. Функция resize() изменяет высоту и ширину у изображения, а для конвертации в RGB функция cvtColor():

Листинг 1 – Типовой вариант обработки изображений.

|  |
| --- |
| *# общий размер для всех изображений*  SIZE = 256  high\_img = []  *# путь к изображениям с высоким разрешением*  path = '../../data/image-super-resolution/dataset/Raw Data/high\_res'  files = os.listdir(path)  files = sorted\_alphanumeric(files)  for i in tqdm(files):  if i == '855.jpg':  break  else:  img = cv2.imread(path + '/'+i,1)  *# opencv по умолчанию считывает в BGR формате, сразу конвертируем в RGB*  img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  *# делаем одинаковый размер*  img = cv2.resize(img, (SIZE, SIZE))  *# нормализуем значения пикселей*  img = img.astype('float32') / 255.0  high\_img.append(img\_to\_array(img))  low\_img = []  *# путь к изображениям с низким разрешением и далее все аналогично*  path = '../../data/image-super-resolution/dataset/Raw Data/low\_res'  files = os.listdir(path)  files = sorted\_alphanumeric(files)  for i in tqdm(files):  if i == '855.jpg':  break  else:  img = cv2.imread(path + '/'+i,1)  img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  img = cv2.resize(img, (SIZE, SIZE))  img = img.astype('float32') / 255.0  low\_img.append(img\_to\_array(img)) |

Цвета пикселей необходимо нормализовать, то есть привести значение каждого пикселя к диапазону [0, 1], так как Keras работает только с таким диапазоном. Для этого значение цвета каждого пикселя делится на 255 (в случае 8-битного цвета).

Теперь имеются два массива (high\_img и low\_img), с которыми уже можно работать.

Для того, чтобы посмотреть на примеры изображений, обычно применяется Matplotlib (рис. 3.1):



Рисунок 3.1 – Примеры изображений из набора Image Super Resolution.

Как видно из примеров, соответствующие изображения разных классов отличаются своей резкостью.

**Набор для колоризации – Landscape color and grayscale images**

Набор содержит 7129 снимков (в формате jpg) городских улиц и природных объектов (горы, леса, ледники) в двух категориях – color и gray (рис. 3.2):

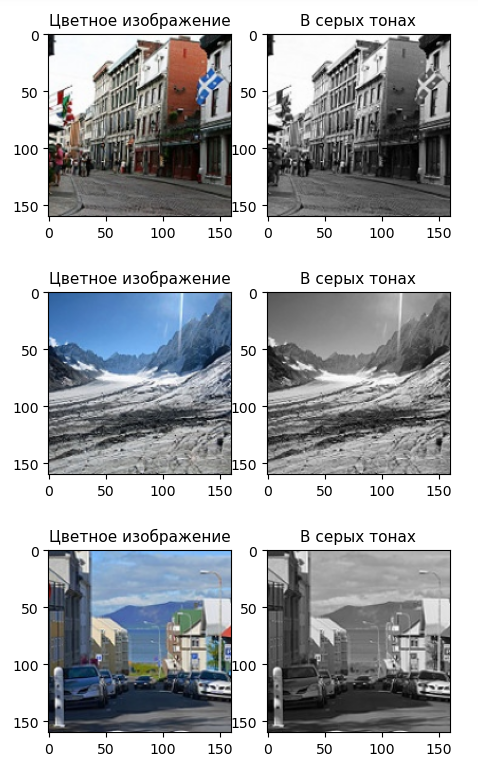


Рисунок 3.2 – Примеры изображений из набора Landscape color and grayscale images.

Поскольку данные в наборах не разделены на тестовые, валидационные (проверочные) и данные для обучения, необходимо выполнить это разделение самостоятельно (далее примеры для набора Image Super Resolution):

Листинг 2 – Разделение изображений.

|  |
| --- |
| train\_high\_image = high\_img[:700]  train\_low\_image = low\_img[:700]  train\_high\_image = np.reshape(train\_high\_image,(len(train\_high\_image),SIZE,SIZE,3))  train\_low\_image = np.reshape(train\_low\_image,(len(train\_low\_image),SIZE,SIZE,3))  validation\_high\_image = high\_img[700:830]  validation\_low\_image = low\_img[700:830]  validation\_high\_image= np.reshape(validation\_high\_image,(len(validation\_high\_image),SIZE,  SIZE,3))  validation\_low\_image = np.reshape(validation\_low\_image,(len(validation\_low\_image),SIZE,SIZE,3))  test\_high\_image = high\_img[830:]  test\_low\_image = low\_img[830:]  test\_high\_image= np.reshape(test\_high\_image,(len(test\_high\_image),SIZE,SIZE,3))  test\_low\_image = np.reshape(test\_low\_image,(len(test\_low\_image),SIZE,SIZE,3)) |

С помощью срезов в python можно разделить часть массива, а функция numpy.reshape используется для изменения формы любого существующего массива (в данном случаем многомерного) без изменения данных, которые в нем находятся.

В итоге shape (форма) всех трех массивов будет типа (*кол-во экземпляров, число пикселей по вертикали, число пикселей по горизонтали, число цветовых каналов*), (рис. 3.3):



Рисунок 3.3 – Форма подготовленных массивов.

Основной массив (train\_image) всегда самый большой, и используется для подбора параметров внутри сети (веса нейронов). Проверочный массив (validation\_image) обеспечивает беспристрастную оценку соответствия модели набору обучающих данных при настройке гиперпараметров (например, количество скрытых слоев). Наборы данных для проверки могут использоваться для регуляризации путем ранней остановки (остановка обучения при увеличении ошибки в наборе данных для проверки, поскольку это признак чрезмерной подгонки к набору данных для обучения). С помощью набора для тестирования (test\_image) после обучения можно будет протестировать модель.

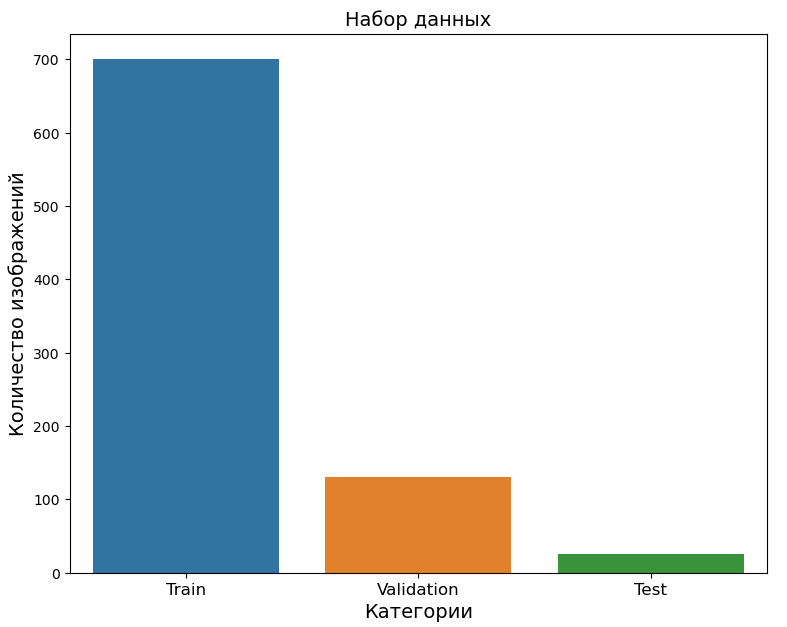


Рисунок 3.4 – Распределение изображений по категориям для обучения.

## **3.2 Создание и обучение модели**

#### **Создание модели**

Для повышения читаемости кода и избегания его избыточного дублирования лучше создать функции для добавления слоев:

Листинг 3 – Функции для создания энкодера и декодера.

|  |
| --- |
| def down(filters, kernel\_size, apply\_batch\_normalization = True):  downsample = tf.keras.models.Sequential()  downsample.add(layers.Conv2D(filters,kernel\_size,padding =  'same', strides = 2))  if apply\_batch\_normalization:  downsample.add(layers.BatchNormalization())  downsample.add(keras.layers.LeakyReLU())  return downsample  def up(filters, kernel\_size, dropout = False):  upsample = tf.keras.models.Sequential()  upsample.add(layers.Conv2DTranspose(filters, kernel\_size,padding  = 'same', strides = 2))  if dropout:  upsample.dropout(0.2)  upsample.add(keras.layers.LeakyReLU())  return upsample |

Данные функции создают блоки свертки и расширения.

Функция down создает блок с двумя слоями: свертка (Conv2D) и, если аргумент apply\_batch\_normalization=True, слой нормализации пакета, а затем выполняет нелинейную функцию активации ReLU. Этот блок уменьшает пространственное разрешение входного изображения, используя свертку с ядром размера kernel\_size (будет передан при вызове) и шагом 2, что уменьшает размерность тензора в два раза.

Функция up создает блок с двумя слоями: обратную свертку (Conv2DTranspose) и слой активации ReLU. Если аргумент dropout=True, то также применяется слой Dropout со значением 0.2. Этот блок увеличивает пространственное разрешение входного тензора путем обратной свертки с ядром размера kernel\_size и шагом 2, что увеличивает размерность тензора в два раза.

Пакетная нормализация (BatchNormalization) — метод, который позволяет повысить производительность и стабилизировать работу искусственных нейронных сетей. Суть метода заключается в том, что некоторым слоям нейронной сети на вход подаются данные, предварительно обработанные и имеющие нулевое математическое ожидание и единичную дисперсию.

Слои Dropout применяются для решения проблемы переобучения. Для этого слой Dropout случайным образом выбирает некоторые входные значения равные единице и при каждом обновлении устанавливает им значение 0.

В качестве функции активации выбрана LeakyReLU. Вычисление сигмоиды и гиперболического тангенса требует выполнения ресурсоемких операций, таких как возведение в степень, в то время как вычисление LeakyReLU проще. Кроме того, LeakyReLU не подвержена насыщению (затуханию градиента). Применение LeakyReLU существенно повышает скорость сходимости стохастического градиентного спуска по сравнению с сигмоидой и гиперболическим тангенсом. Это обусловлено линейным характером и отсутствием насыщения данной функции.

Исходный код модели принимает следующий вид:

Листинг 4 – Определение модели.

|  |
| --- |
| def model():  inputs = layers.Input(shape=[SIZE,SIZE,3])  *# энкодер*  d1 = down(128,(3,3),False)(inputs)  d2 = down(128,(3,3),False)(d1)  d3 = down(256,(3,3),True)(d2)  d4 = down(512,(3,3),True)(d3)  d5 = down(512,(3,3),True)(d4)  *# декодер*  u1 = up(512,(3,3),False)(d5)  u1 = layers.concatenate([u1,d4])  u2 = up(256,(3,3),False)(u1)  u2 = layers.concatenate([u2,d3])  u3 = up(128,(3,3),False)(u2)  u3 = layers.concatenate([u3,d2])  u4 = up(128,(3,3),False)(u3)  u4 = layers.concatenate([u4,d1])  u5 = up(3,(3,3),False)(u4)  u5 = layers.concatenate([u5,inputs])    output = layers.Conv2D(3,(2,2),strides = 1, padding = 'same')(u5)    return tf.keras.Model(inputs=inputs, outputs=output) |

Модель получает на вход изображение (матрицу) размером 255x255 пикселей в трехканальном цветовом режиме (слой inputs).

Переменная "d1" и аналогичные имеют тип "KerasTensor" и последовательно приравнивается новому слою и получает предыдущий слой:

*x = (класс слоя)(предыдущий слой)*

Слои Conv2D осуществляют свертку изображения, при этом растет число распознаваемых параметров (в случае Conv2DTranspose идет обратный процесс). Всего используется 5 таких слоев. Эти слои параметризируются следующим образом – (число карт свертки, размер ядра свертки, функция активации, padding, имя слоя). Атрибут "padding" отвечает за размер новой карты. При выборе "same" размер новой карты будет как у предыдущей.

BatchNormalization используются для оптимизации работы сети. В качестве входного параметра принимают только имя.

Слои Dropout должны снизить эффект переобучения. Они параметризируются долей отклонения.

Последний слой в этой модели – это Conv2D с 3 выходными фильтрами и ядром размера (2,2), который используется для генерации выходного изображения. Этот слой выполняет свертку с фильтрами размера (2,2) с обработанным представлением входных данных из предыдущего слоя.

По умолчанию, этот слой выполняет действия, которые приводят к созданию нового изображения размера 255x255x 3 канала (RGB).

Вызвав метод summary() к созданной модели можно увидеть информацию о ней (рис. 3.5):

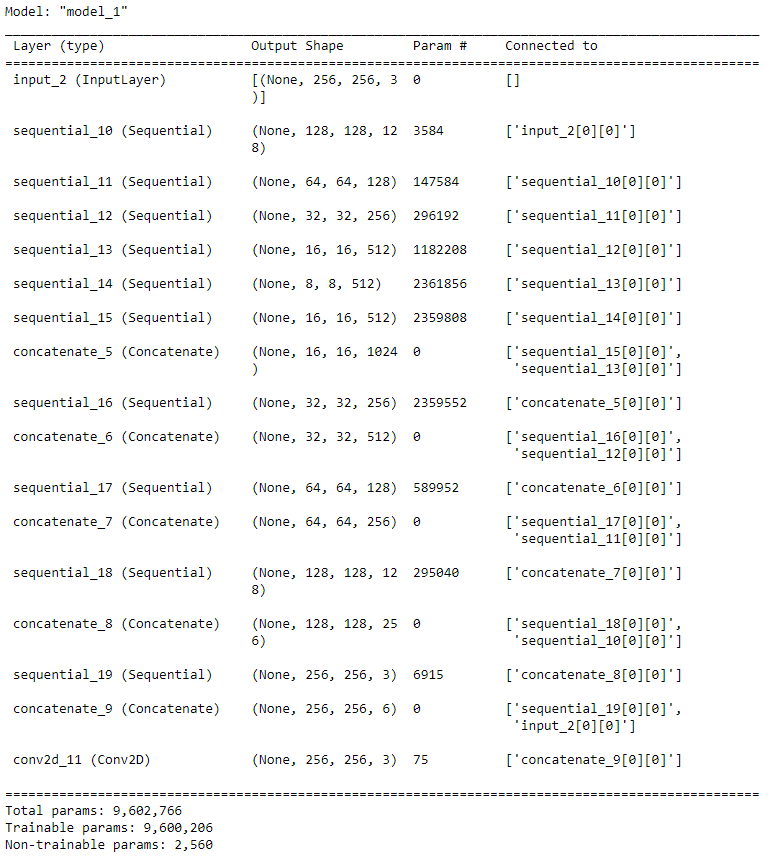


Рисунок 3.5 – Сводка по модели

Суммарно модель имеет 9 600 206 обучаемых параметров. Наличие необучаемых параметров означает, что 2560 весов не будут обновляться во время обучения с обратным распространением. Эти параметры появляются при использовании слоев BatchNormalization.

#### **Обучение модели**

Для успешного обучения необходимо правильно настроить гиперпараметры.

В качестве алгоритма оптимизации стохастического градиентного спуска был выбран Adam – он объединяет лучшие свойства алгоритмов AdaGrad и RMSProp, может обрабатывать редкие градиенты при обработке шумных изображений, таких как рентгеновские снимки.

Ранняя остановка (EarlyStopping) установлена на срабатывание в случае, если параметр monitor (в данном случае он установлен на метрику val\_loss) перестает улучшаться в течении трех эпох обучения. Это поможет снизить эффект переобучения и не подбирать количество эпох вручную.

Функция ModelCheckpoint будет сохранять модель после каждой эпохи обучения. Если после очередной эпохи точность снизится, можно будет вернуться к предыдущей модели, чтобы не запускать обучение с самого начала.

Перед обучением модель необходимо скомпилировать – model.compile(), внутрь функции передается название выбранной функции ошибки (была выбрана mean\_absolute\_error), оптимизатор (Adam) и список метрик (accuracy)

Размер партии (batch\_size) был установлен на 5. Выбор размера партии обуславливается объемом выделяемой памяти: чем меньше памяти, тем меньше размер партии. Можно увеличивать значение до тех пор, пока программа не перестанет работать по причине нехватки памяти. Batch\_size не влияет на качество обучения.

Количество эпох установлено на 50. Это значение взято с запасом, так как меньшее число эпох может оказаться недостаточно для полного обучения. В любом случае, как только точность перестанет увеличиваться, сработает ранняя остановка и обучение прекратится.

После установки batch\_size и количества эпох дополнительно можно определить количество шагов за эпоху (steps\_per\_epoch) – это количество изображений, обрабатываемых за одну эпоху. Обычно вычисляется путем деления общего числа образцов обучающей выборки на batch\_size.

Обучение модели в Keras запускается вызовом метода model.fit(), в который передается генератор и параметры, определенные ранее:

Листинг 5 – Компиляция модели и запуск обучения.

|  |
| --- |
| *# компиляция модели*  model.compile(optimizer = tf.keras.optimizers.Adam(learning\_rate = 0.001), loss = 'mean\_absolute\_error', metrics = ['acc'])  *# запуск обучения*  history = model.fit(train\_low\_image, train\_high\_image, epochs = 30, batch\_size = 5,  validation\_data = (validation\_low\_image,validation\_high\_image), callbacks=[early\_stopping]) |

После обучения модель лучше сразу сохранить методом save(), чтобы использовать ее в дельнейшем (метод load\_model()).

Использование GPU для обучения существенно ускоряет этот процесс, относительно использования CPU. Этот факт обусловлен более удачной для таких вычислений архитектурой ядра, позволяющей GPU эффективно справляются с большим количеством несложных однотипных задач. Для обучения на GPU необходимо иметь одну из последних версий TensorFlow и установленный CUDA Toolkit от NVIDIA, или воспользоваться облачными сервисами на основе Jupyter Notebook – google collab или kaggle, они предоставляют ресурсы своих GPU бесплатно, но с ограничениями по времени.

Для классификации новых изображений вызывается метод predict(). Функция, обрабатывающая одно новое изображение из указанной директории, выглядит следующим образом:

Листинг 6 – Классификация загруженного изображения.

|  |
| --- |
| def single\_predict(path, save=False):  *# Функция для печати изображения*  def plot\_for\_single(low,predicted):  plt.figure(figsize=(10,10))  plt.subplot(1,2,1)  plt.title('Low Image ', color = 'black', fontsize = 20)  plt.imshow(low)  plt.subplot(1,2,2)  plt.title('Predicted', color = 'green', fontsize = 20)  plt.imshow(predicted)  plt.show()  *# Считываем и подготавливаем изображение*  SIZE = 256  img = cv2.imread(path,1)  img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)  img = cv2.resize(img, (SIZE, SIZE))  img = img.astype('float32') / 255.0  *# Нормализуем цвета и используем predict*  predicted = np.clip(model.predict(img.reshape(1,SIZE,  SIZE,3)),0.0,1.0).reshape(SIZE, SIZE,3)    *# используем OpenCv для дополнительной обработки*  kernel = np.array([[0, -1, 0],  [-1, 5, -1],  [0, -1, 0]])  predicted = cv2.filter2D(predicted, -1, kernel)  predicted = cv2.resize(predicted, (2560, 2560))  *# Печатаем изображение*  plot\_for\_single(img, predicted)    *# Сохраняем, если нужно*  if save:  file\_name = 'predicted.jpg'  cv2.imwrite(file\_name, cv2.cvtColor(predicted\*255,  cv2.COLOR\_BGR2RGB)) |

Model.predict(img.reshape(1,SIZE, SIZE,3)) выполняет предсказание модели на входном изображении img и возвращает массив значений выходных данных (рис. 3.13).

Np.clip(arr, a\_min, a\_max) обрезает массив arr так, чтобы его значения находились в диапазоне от a\_min до a\_max. В данном случае a\_min равно 0.0 и a\_max равно 1.0, что означает, что значения выходных данных модели будут ограничены от 0 до 1.

Reshape(SIZE, SIZE, 3) используется для приведения выходных данных модели к форме, соответствующей размеру изображения.

В результате выполнения этого кода переменная predicted будет содержать массив значений выходных данных, ограниченный диапазоном от 0 до 1, и имеющий форму, соответствующую размеру входного изображения.



Рисунок 3.6 – Результат вызова функции single\_predict.

Как видно из рис. 3.6, полученное изображение гораздо более четкое, чем изначальное. Полученный результат можно дополнительно улучшить с помощью OpenCV.

В OpenCV функция resize используется для изменения размера изображения. Она позволяет уменьшать или увеличивать размер изображения в соответствии с заданными параметрами.

В функции resize доступны различные методы интерполяции, такие как:

* cv2.INTER\_NEAREST - ближайшее соседнее значение
* cv2.INTER\_LINEAR - линейная интерполяция
* cv2.INTER\_CUBIC - кубическая интерполяция (лучше всего подходит для уменьшения размера изображения)
* cv2.INTER\_LANCZOS4 - интерполяция Ланцоша (лучше всего подходит для увеличения размера изображения)

При уменьшении размера изображения OpenCV использует сглаживание, чтобы сгладить пиксели и избежать появления шумов. При увеличении размера изображения OpenCV использует интерполяцию, чтобы вычислить новые значения пикселей на основе существующих значений.

Еще одна подходящая в данном случае функция – filter2D. Она позволяет применять пользовательские фильтры к изображению, позволяет выполнять свертку изображения с ядром (kernel), которое можно настроить для различных задач обработки изображений.

В качестве ядра для функции filter2D можно использовать любую матрицу, например, можно применять ядра для фильтрации изображений, такие как:

* ядро Гаусса (cv2.getGaussianKernel())
* ядро Собеля для нахождения градиента изображения (cv2.Sobel())
* ядро Лапласа для обнаружения границ изображения (cv2.Laplacian()),

После этих действий, результирующее изображение станет более резким. Кроме того, было увеличено его разрешение в 10 раз – интерполяция, предоставляемая функцией resize, особенно хорошо работает на не очень четких изображениях:



Рисунок 3.7 – Сравнение итоговых изображений

## **3.3 Тестирование модели**

Для построения графиков по модели Keras можно использовать библиотеку Matplotlib:

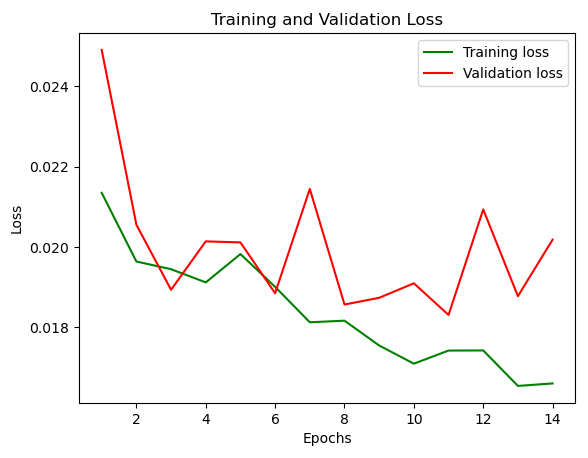


Рисунок 3.8 – График обучения: Training loss – потери на наборе для обучения, Validation loss – потери на наборе для проверки.

Потери (loss) представляют собой функцию, которая измеряет расхождение между предсказанными значениями модели и истинными значениями целевой переменной. Целью оптимизации нейронной сети является минимизация этих потерь. Расхождение графиков на рис. 3.17 говорит о переобучении – модель начинает лучше работать на данных для обучения и хуже на данных, не используемых при обучении.

После нескольких запусков обучения модели выяснилось, что наибольшая точность (Accuracy) и наименьшие потери на обучающей выборке находятся на пятой-шестой эпохе (рис. 3.8), после чего эти показатели только падают, и в конечном счете срабатывает ранняя остановка, так как точность дальше либо не увеличивается, либо увеличивается крайне медленно. Из этого следует, что для данной модели оптимальное число эпох обучения 5 или 6 – оно обеспечит наилучшие числовые показатели метрик и как следствие, качество работы модели на реальных данных.

Изображения, генерируемые моделью после 14 эпох обучения, получаются явно лучше, чем после 5 эпох:

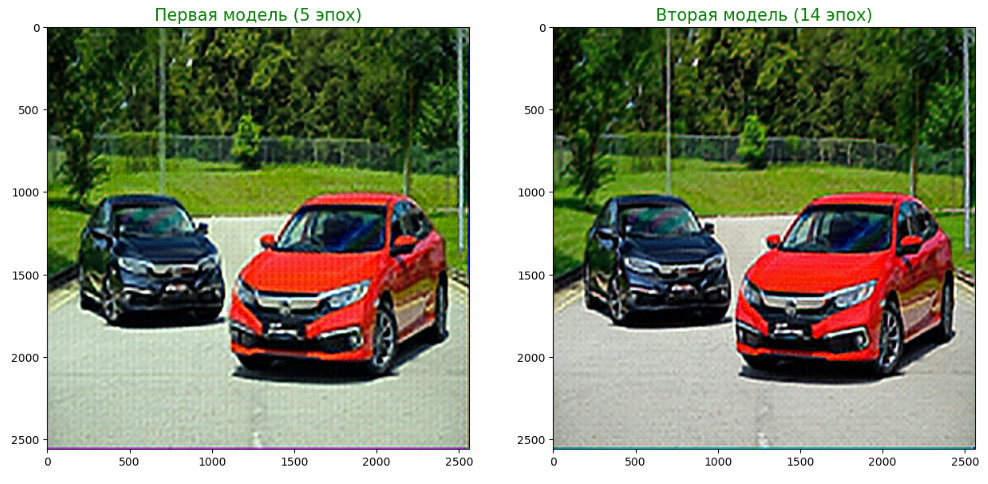


Рисунок 3.9 – Сравнение изображений, полученных после разного количества эпох: слева – 5, справа – 14.

Как видно из рис. 3.9 цвета становятся более похожими на изначальное изображение, исчезает шум, появляющийся у модели, обученной на пяти эпохах, некоторые элементы картинки становятся более плавными.

Это видно и по графику на рис. 3.8 (Training loss), но не стоит забывать, что это происходит по причине излишней подгонки весовых коэффициентов под набор для обучения, на реальных данных такая модель будет работать хуже (Validation loss на рис. 3.8).

## **3.4 Разработка серверной части**

Чтобы обработать изображение пользователя полученной моделью, необходимо передать его в подготовленный скрипт на python, где к нему будет применена функция predict. Но до этого момента изображение нужно еще отправить на сервер, где данный скрипт и будет работать, а после этого, отправить пользователю готовый результат. Реализовать такой функционал можно с помощью популярного на данный момент фреймворка Express для Node.js и библиотеки Multer для работы с файлами.

Для этого нужно создать соответствующие конечные точки (далее – эндпоинты), отвечающие за загрузку и отправку файлов. Конечная реализация эндпоинта для получения и обработки изображения выглядит следующим образом:

Листинг 7 – Эндпоинт для получения и обработки изображения.

|  |
| --- |
| *// Эндпоинт для загрузки изображения*  app.post('/upload', upload.single('image'), (req, res) => {  *// Определяем желаемый тип обработки изображения*  const type = req.query.type;  *// Если параметр с типом не определен выдаем сообщение об ошибке*  if (!type) {  res.json({  message: "Add query params: type='colorization' or  'resolution'",  });  }  *// Получаем оригинальное название изображения*  const { originalname, path } = req.file;  *// Подготавливаем новые уникальные имена*  const newName = `${Date.now() + '\_' + originalname}`;  const newPath = `uploads/${newName}`;  const predictedName = `predicted\_${newName}`;  const predictedPath = `uploads/${predictedName}`;  const predictedProcessedName = `predicted\_processed\_${newName}`;  const predictedProcessedPath =  `uploads/${predictedProcessedName}`;  *// Переименовываем полученный файл*  fs.renameSync(path, newPath);  *// Выбераем скрипт, в зависимости от параметров запроса*  const python = type === 'resolution' ? 'srgan.py' :  'colorization.py'  *// Команда для запуска скрипта*  const command = `conda run -n tensorflow python ${python}  ${newPath} ${predictedName} ${predictedProcessedName}`  *// Запускаем скрипт и по завершении его работы отправляем пользователю ссылки на изображения*  const pythonScript = spawn(command, { shell: true });  pythonScript.on('close', (code) => {  res.json({  message: 'File uploaded successfully',  link: newPath,  predictedLink: predictedPath,  predictedProcessedLink: predictedProcessedPath  });  });  }); |

Теперь можно отправить изображение на сервер, используя url и созданный эндпоинт “/upload”, указав в качестве query-параметра тип обработки (например, *http://localhost:3000/upload?type=resolution*). После этого будет выполнен соответствующий скрипт, и в качестве ответа пользователь получит json с ссылками на результирующие изображения.

После получения ссылок на изображения, клиент должен получить и сами файлы:

Листинг 8 – Эндпоинт для получения изображения.

|  |
| --- |
| *// Эндпоинт для отправки изображения*  app.get('/uploads/:imageName', (req, res) => {  *// Получаем название изображения*  const imageName = req.params.imageName;  *// Получаем путь до него*  const imagePath = `uploads/${imageName}`;  const options = {  root: path.resolve()  };  *// Отправляем изображение клиенту*  res.sendFile(imagePath, options);  }); |

Теперь можно получить изображения выполнив подобный запрос – *http://localhost:3000/uploads/predicted\_1683727000210\_0.png.*

Такого функционала на сервере хватит для реализации требований к системе на клиентской стороне.

## **3.5 Разработка клиентской части**

Клиентская часть системы реализована на React. Для развертывания проекта используется утилита Vite – она инициализирует начальную структуру приложения с минимальным количеством необходимых для старта зависимостей.

Для отправки запросов на сервер используется библиотека axios. Запрос выполняется после выбора пользователем фотографии из системы. Реализация отправки изображений и получения ссылки на сгенерированный результат выглядит следующим образом:

Листинг 9 – Отправка и получение изображений на клиенте.

|  |
| --- |
| *// Локалиные состояния (стейты) для хранения изображений*  const [image, setImage] = React.useState(null);  const [predictedImage, setPredictedImage] = React.useState(null);  const [predictedProcessedImage, setPredictedProcessedImage] =  React.useState(null);  const [activeOption, setActiveOption] =  React.useState(options.res);  *// Выполняется при выборе изображения*  const handleFileInputChange = (event: any) => {  *// Рендерим значек загрузки на время выполнения запроса*  setIsLoading(true);  *// Получаем изображение*  const file = event.target.files[0];  *// Создаем объект для отправки изображения*  const formData = new FormData();  formData.append('image', file);  *// Выполняем post запрос на определенный на сервере эндпоинт*  *axios*  .post(`/upload?type=${activeOption}`, formData)  .then(response => {  *// Заполняем локальные стейты ссылками на сгенерированные изображения*  const data = response.data;  setImage(data.link);  setPredictedImage(data.predictedLink);  setPredictedProcessedImage(data.predictedProcessedLink);  setIsLoading(false);  })  .catch(error => {  setIsLoading(false);  });  }; |

Полученное из тега input изображение помещается в объект FormData отправляется на сервер. На время выполнения запроса пользователь видит индикатор загрузки. После выполнения запроса выполняется рендер тегов img, содержащих в себе ссылки на изображения, полученные в качестве ответа от сервера. Главная страница приложения имеет следующий вид (рис. 3.10):

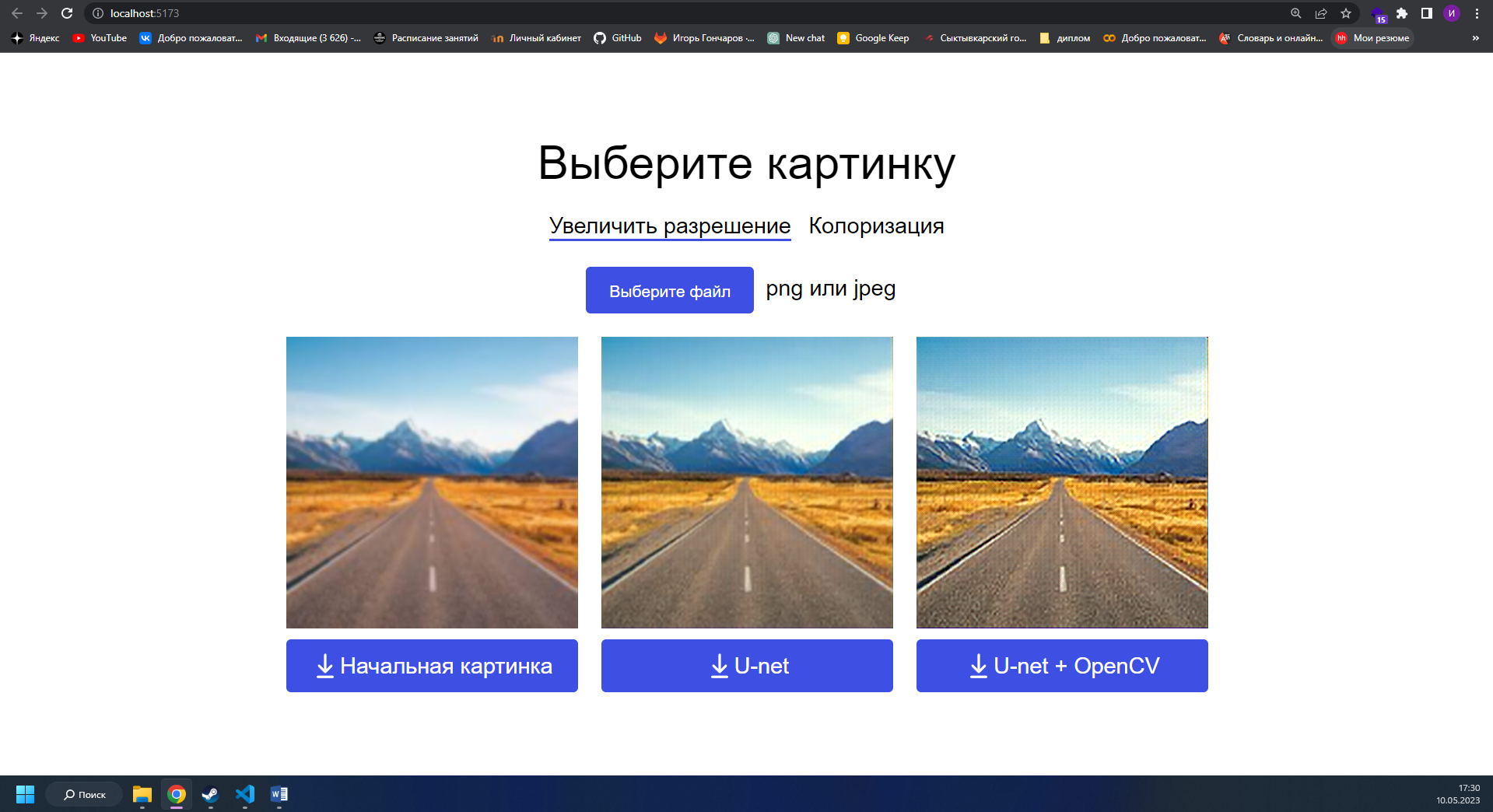


Рисунок 3.10 – Вид главной страницы приложения.

Пользователь выбирает нужный вид обработки, далее выбирает изображение и получает результат от сервера. Все полученные изображения можно сразу сохранить.

# **ГЛАВА 4. МЕНЕДЖМЕНТ И ЭКОНОМИЧЕСКИЙ АНАЛИЗ ПРОЕКТА**

## **4.1 Проектный менеджмент**

**Структура работ в рамках научного исследования**

Ниже представлена таблица выполненных работ с указанием исполнителей.

Таблица 4.1 – Перечень этапов, работ и распределение исполнителей.

|  |  |  |  |
| --- | --- | --- | --- |
| **Основные этапы** | **№ работы** | **Содержание работ** | **Исполнители** |
| Выбор направления исследования | 1 | Выбор темы исследования | Исполнитель, руководитель |
| 2 | Подбор материалов по теме | Исполнитель |
| 3 | Изучение материалов по теме | Исполнитель |
| 4 | Выбор направления | Исполнитель, руководитель |
| 5 | Календарное планирование работ по теме | Исполнитель |
| Проектирование структуры и разработка ИС | 6 | Проектирование структуры ИС | Исполнитель |
| 7 | Разработка ИС | Исполнитель |
| 8 | Тестирование ИС | Исполнитель |
| Обобщение и оценка результатов | 9 | Оценка эффективности полученных результатов | Исполнитель, руководитель |
| 10 | Прогноз экономических показателей | Исполнитель |
| Оформление отчета по работе | 11 | Написание текста работы | Исполнитель |
| 12 | Составление диаграм, схем, таблиц | Исполнитель |

**SWOT-анализ**

Для исследования внешней и внутренней среды проекта был проведен SWOT-анализ, который отражает сильные и слабые стороны разрабатываемого проекта.

Таблица 4.2 – SWOT-анализ проекта.

|  |  |
| --- | --- |
| **Сильные стороны** | **Слабые стороны** |
| 1. Удобство в эксплуатации (соответствует требованиям потребителей). 2. Функциональная мощность (предоставляемые возможности). 3. Конкурентоспособность продукта (бесплатное распространение). 4. Доступные технические средства разработки (Node.js, React, Express) | 1. Срок выхода на рынок. 2. Значительные временные и интеллектуальные затраты на реализацию. |
| **Возможности** | **Угрозы** |
| 1. Привлечение разработчиков для продолжения проекта. 2. Повышение стоимости конкурентных разработок. 3. Появление дополнительного спроса на новый продукт (популярное направление). 4. Публикации о проекте в тематических источниках. | 1. Отсутствие спроса на расширение разработки. 2. Отказ от технической поддержки проекта после внедрения. 3. Нехватка финансирования. 4. Развитая в данной отрасли конкуренция. |

Основным преимуществом проекта является простота в использовании и мощный функционал, а основные недостатки – сложность разработки и развитая в отрасли работы с изображениями конкуренция.

**Разработка графика проведения научного исследования**

Для наглядного отображения распределения работ по реализации системы во времени была построена диаграмма Ганта (рис. 4.1).

Диаграмма Ганта – горизонтальный ленточный график, на котором работы по теме представляются протяженными во времени отрезками, характеризующимися датами начала и окончания выполнения данных работ. Диаграмма Ганта для данной работы представлена в Приложении 1.

## **4.2 Оценка затрат проекта**

Цены на решения для разработки приведены в Таблице 4.3:

Таблица 4.3 – Цены на решения для разработки.

|  |  |  |
| --- | --- | --- |
|  | **Название** | **Цена** |
| **Среда разработки** | Visual Studio Code | Бесплатно |
| PyCharm | 249,00 $ |
| WebStorm | 159,00 $ |
| **Среда для обучения нейронной сети** | Личный ПК (GPU) | - |
| Google Collaboratory | 9,99 $ |
| Amazon Machine Learning | 0,10 $ за 1000 прогнозов |

Visual Studio Code является бесплатным и по функционалу почти не уступает платным аналогам, его вполне достаточно для быстрой и комфортной разработки.

Для обучения нейронных сетей на больших наборах данных нужно иметь большие вычислительные мощности, но в данном случае используются наборы из нескольких тысяч картинок, это крайне мало по современным меркам, и для быстрого обучения хватит мощности одной средней видеокарты или даже процессора.

Плановая стоимость проекта на первый год:

Таблица 4.4 – Стоимость проекта.

|  |  |
| --- | --- |
| **Затраты** | **Цена** |
| Среда разработки (Visual Studio Code) | 0,00 ₽ |
| Среда для обучения сети | 0,00 ₽ |
| Аренда сервера (1 год) | 12 000,00 ₽ |
| Аренда домена (1 год) | 150,00 ₽ |
| Итого: | 12 150,00 ₽ |

Таким образом, фактическая стоимость проекта за первый год составила 12 150 рублей.

## **4.3 Прогноз экономической эффективности проекта**

Потенциальные потребители результатов исследования – пользователи, работающие с изображениями: направления связанных с рекламой, маркетингом или для личного пользования с целью обработки изображений.

Оценка потенциальной экономической эффективности проекта проводилась с помощью технологии BSC-KPI (метод сбалансированных показателей по ключевым критериям эффективности[17]). Полученная карта BSC стратегий и результирующая BSC-карта показана в Приложении 2 и в таблице 4.5 соответственно.

Таблица 4.5 – BSC-карта оценки экономической эффективности веб-

сервиса для обработки изображений.

|  |  |  |
| --- | --- | --- |
| **Уровень** | **Улучшения в процессе** | **Отражение качественных изменений в количественных показателях (BSC-KPI)** |
| Финансовые цели | Доход от рекламы внутри приложения | Сумма от доходов на рекламе в год - 100 тысяч рублей |
| Сбор пожертвований | Сумма пожертвований в год - 50 тысяч рублей |
| Рыночные цели | Формирование рынка интеллектуальной обработки изображений | Количество запросов на обработку в месяц - 200000 |
| Внутренние цели | Увеличение скорости обработки | Время на обработку одного изображения - 3 секунды |
| Повышение качества обработки | Потери (loss) на проверочном наборе < 0.018 |

Анализ позволяет предположить, что разработанный веб-сервис может приносить доход 150 тысяч рублей в год от пожертвований пользователей и размещения рекламы.

# **Заключение**

Интеллектуальная обработка изображений посредством использования библиотек компьютерного зрения и нейронных сетей является перспективным направлением деятельности и активно применяется уже сейчас.

В ходе выполнения данной работы были осуществлены исследование, проектирование и разработка приложения обработки изображений. Основная цель заключалась в создании инструмента, способного автоматизировать процесс обработки и анализа изображений, обеспечивая точность, эффективность и удобство использования. Были рассмотрены основные актуальные методы обработки изображений с использованием нейронных сетей и библиотек компьютерного зрения, а также методы их обучения и оценки качества, рассмотрен процесс создания клиентского приложения для данной системы.

Использование программ, таких как представленная в данной работе, не требует от пользователя никаких прикладных знаний в области обработки изображений и существенно экономит время при использовании, относительно работы с графическими редакторами.

Результат данной дипломной работы – готовая сверточная нейронная сеть архитектуры U-net для работы с изображениями и web-приложение для нее.

Направлением для дальнейшей работы может быть увеличение качества обработки путем изменения модели, гиперпараметров или увеличения набора данных, расширение функционала приложения.

# **Список источников**

1. Ю. С. Осипов – Большая российская энциклопедия, 2017
2. Anil K. Jain, Jianchang Mao, K.M. Mohiuddin – Artificial Neural Networks: A Tutorial
3. Розенблатт, Ф. – Принципы нейродинамики: Перцептроны и теория механизмов мозга, 1965
4. Фомин, С. В., Беркинблит, М. Б. – Математические проблемы в биологии
5. Bishop, C. M. – Pattern recognition. Machine Learning
6. Nair, V., & Hinton, G. E. – Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th international conference on machine learning, 2010
7. Yu, D., & Deng, L. – Automatic speech recognition: A deep learning approach. Springer, 2014
8. Pascanu, R., Mikolov, T., & Bengio, Y. – On the difficulty of training recurrent neural networks, 2013
9. LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. – Backpropagation applied to handwritten zip code recognition. Neural computation, 1989
10. Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. – Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 2014
11. Ciresan, Dan; Meier, U.; Schmidhuber, J. – Multi-column deep neural networks for image classification, 2012
12. Уоссермен, Ф. Нейрокомпьютерная техника: Теория и практика, 1992
13. Claesen, Marc & Bart De Moor – Hyperparameter Search in Machine Learning, 2015
14. James Bergstra, Yoshua Bengio. Random Search for Hyper-Parameter, 2012
15. Орельен Жерон – Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем, 2018
16. Джулли А., Пал С – Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow, 2017
17. Нортон Д., Каплан Р. Сбалансированная система показателей. От стратегии к действию. – М.: Олимп-Бизнес, 2010. – 320 с.
18. Л. Шапиро, Дж. Стокман. Компьютерное зрение = Computer Vision. — М.: Бином. Лаборатория знаний, 2006. — 752 с. — ISBN 5-94774-384-1.
19. Дэвид Форсайт, Жан Понс. Компьютерное зрение. Современный подход = Computer Vision: A Modern Approach. — М.: «Вильямс», 2004. — 928 с. — ISBN 5-8459-0542-7.
20. Желтов С.Ю. и др. Обработка и анализ изображений в задачах машинного зрения. — М.: Физматкнига, 2010. — 672 с. — ISBN 978-5-89155-201-2

# **Приложение 1**

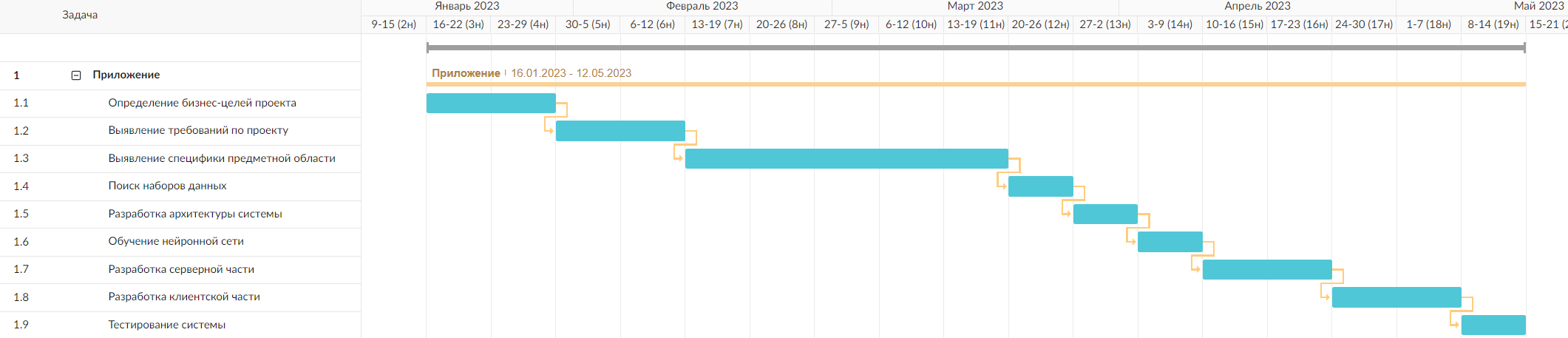


Рисунок 5.1 – Диаграмма Ганта для проекта.

# **Приложение 2**

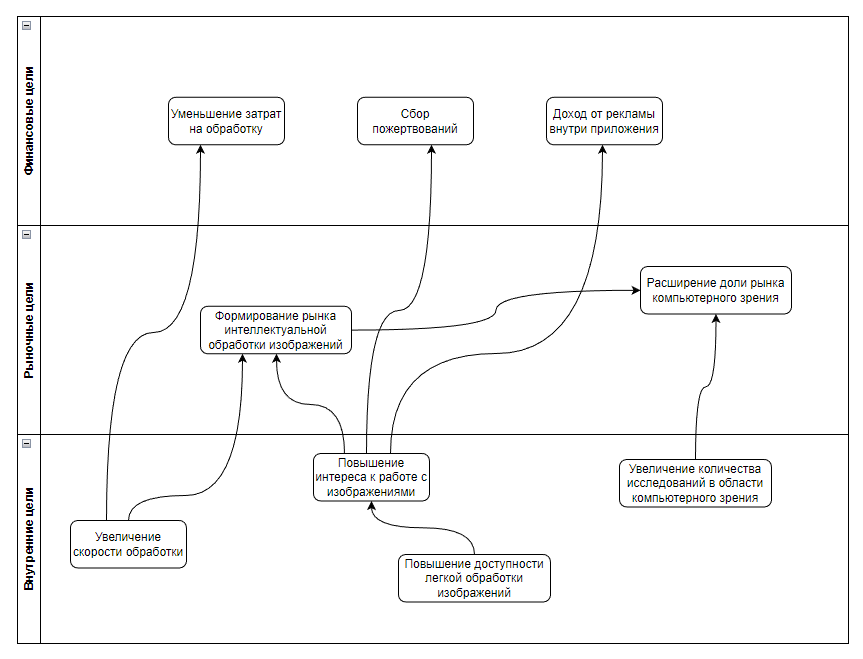


Рисунок 5.2 – Карта BSC-стратегий, связанных с использованием веб-сервиса для обработки изображений.