

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

Курсовая работа по дисциплине «Информационные системы»  
**Использование нейронных сетей для классификации 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  
2022

## Оглавление

Аннотация .....	3
Введение .....	4
ГЛАВА 1. НЕЙРОННЫЕ СЕТИ .....	5
1.1 Понятие нейронной сети .....	5
1.1.1 Искусственные нейронные сети и их составляющие .....	5
1.1.2 Активационная функция нейрона .....	7
1.1.3 Модели нейронов .....	8
1.1.4 Архитектура нейронных сетей .....	12
1.2 Обучение нейронных сетей .....	20
1.2.1 Общие понятия в обучении нейронных сетей .....	20
1.2.2 Гиперпараметры нейронных сетей .....	22
1.2.3 Проблемы при обучении нейронных сетей .....	24
1.2.4 Алгоритмы обучения нейронных сетей .....	28
1.2.5 Метрики нейронной сети .....	29
ГЛАВА 2. СОЗДАНИЕ НЕЙРОННОЙ СЕТИ НА PYTHON .....	33
2.1 Постановка задачи: классификация изображений .....	33
2.2 Инструментарий: Python, библиотеки TensorFlow, Keras .....	33
2.3 Подготовка данных .....	36
2.4 Обучение и тестирование модели .....	41
Заключение .....	53
Список источников .....	54

## **Аннотация**

Целью данной курсовой работы является создание нейронной сети, решающей задачу классификации изображений определенного типа.

В курсовой работе были изложены основные проблемы, связанные с данной темой и рассмотрены методы решения этих проблем. Были изучены основные принципы работы нейронных сетей.

Для реализации системы было решено использовать язык Python, фреймворк TensorFlow, Keras.

Работа включает 53 страницы, содержит 16 литературных источников, 1 таблицу, 26 изображений.

Ключевые слова: искусственные нейронные сети, глубокие нейронные сети, классификация изображений, обучение с учителем, глубокое обучение, сверточная нейронная сеть.

## **Введение**

Объем информации в интернете с каждым годом увеличивается уже почти экспоненциально. Возникает необходимость обработки большого объема данных. На текущее время самый оптимальный способ – создание нейронной сети. Нейросетями обрабатывается любая информация, от графической до огромных массивов данных. В представленной работе рассматривается применение нейронной сети для выявления пневмонии по рентгенографии грудной клетки.

Объектом исследования являются методы на основе нейронных сетей для классификации изображений. Предмет исследования – изучение методов создания нейронных сетей.

Целью работы является создание нейронной сети и ее обучение на подготовленном наборе типовых изображений, для дальнейшего ее использования в качестве инструмента классификации изображений.

### **Задачи исследования:**

- анализ предметной области
- проектирование нейронной сети
- реализация нейронной сети на Python при помощи TensorFlow
- обучение сети и оценка результатов

# ГЛАВА 1. НЕЙРОННЫЕ СЕТИ

## 1.1 Понятие нейронной сети

### 1.1.1 Искусственные нейронные сети и их составляющие

Нейронная сеть – математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма [1].

Нейронная сеть (также искусственная нейронная сеть, ИНС) представляет собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Нейрон – это вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше. Они делятся на три основных типа: входной, скрытый и выходной:

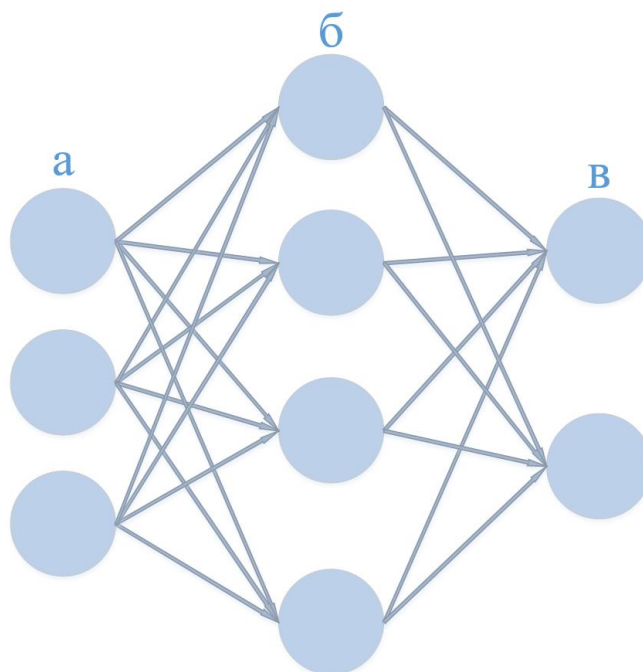


Рисунок 1.1 – Схема простой нейронной сети: а – входные нейроны, б – скрытые нейроны, в – выходные нейроны

В том случае, когда нейросеть состоит из большого количества нейронов, вводят термин слоя. Входной слой получает информацию, и 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.

### 1.1.2 Активационная функция нейрона

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

$$y = f(x) \quad (1.1)$$

$$x = \sum_{i=1}^N w_i s_i \quad (1.2)$$

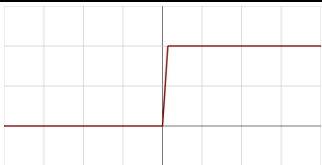



где:

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

От вида и формы используемой активационной 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)		$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$	$\{0, 1\}$
Логистическая (сигмоида или гладкая ступенька)		$\sigma(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Гиперболический тангенс		$\text{th}(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$(-1, 1)$
Линейный выпрямитель (ReLU)		$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$	$[0, +\infty)$

### 1.1.3 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).

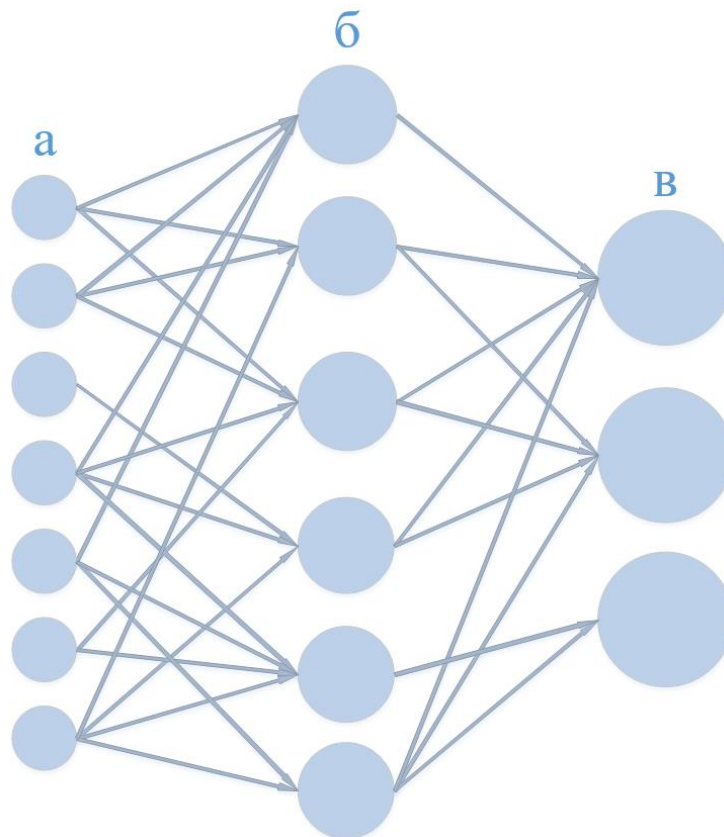


Рисунок 1.2 – Схема персептрона: а – S-элементы, б – А-элементы, в – R-элементы

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 А-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 А-элементы, которые еще называют ассоциативными элементами. Стоит заметить, что одному А-элементу может соответствовать несколько S-элементов. Если сигналы, поступившие на А-элемент, в совокупности превышают некоторый его порог, то этот А-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 Розенблатту — персептрон, у которого имеется более 1 слоя A-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:

$$f(x) = \frac{1}{1 + e^{-\beta x}} \quad (1.3)$$

Чем больше  $\beta$  (параметр наклона сигмоидальной функции активации), тем сильнее крутизна графика. При  $\beta \rightarrow \infty$  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].

### 1.1.4 Архитектура 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  $h_{w,b}(x)$  — 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]:

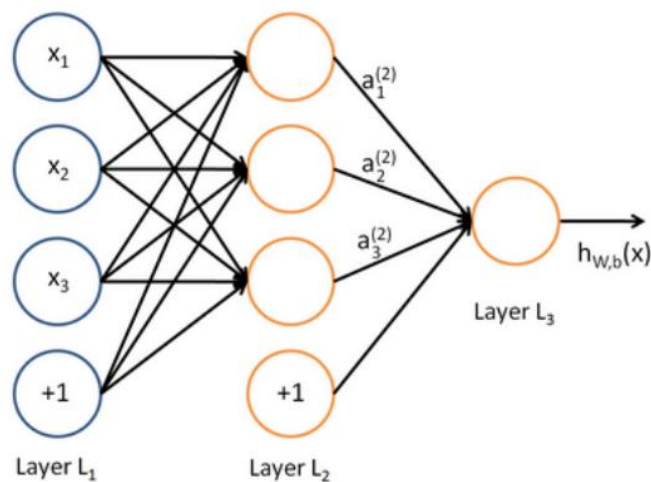


Рисунок 1.3 – Схема сети прямого распространения: Layer  $L_1$  – входной слой, Layer  $L_2$  – скрытый слой, Layer  $L_3$  – выходной слой

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 параметризуется значениями  $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \dots)$ , где под  $W_{ij}^{(l)}$  понимается параметр, или вес, который отвечает соединению между  $j$ -м нейроном в слое  $l$  и  $i$ -м нейроном в слое  $l + 1$ . За  $b_i^{(l)}$  обозначается смещение или вес, который связан с константными единичными входами на каждом слое сети [4].

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

$$\begin{cases} a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_{(1)}^1) \\ a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_{(1)}^2) \end{cases} \quad (1.4)$$

Обозначив функцию суммирования через  $z$ , получим:

$$\begin{cases} z^{(2)} = W^{(1)}x + b^{(1)} \\ a^{(2)} = f(z^{(2)}) \\ z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \\ h = f(z^{(3)}) \end{cases} \quad (1.5)$$

Общая формула будет выглядеть следующим образом:

$$\begin{cases} z^{(l)} = W^{(l)}a^{(l)} + b^{(l)} \\ a^{(l)} = f(z^{(l+1)}) \end{cases} \quad (1.6)$$

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 [9] (СНС, convolutional neural network) – класс нейронных сетей, специализирующихся на обработке данных, представленных в виде матрицы (например, изображения).

СНС 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:

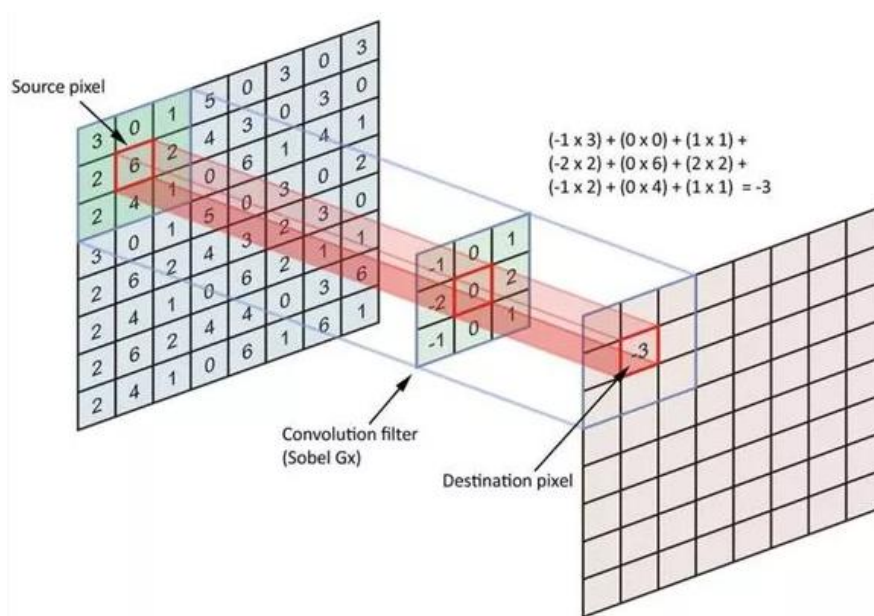


Рисунок 1.4 – Свертка изображения: Source pixel – исходный пиксель, convolution filter – ядро свертки, destination pixel – пиксель свернутого изображения

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 3x3 до 7x7. 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):

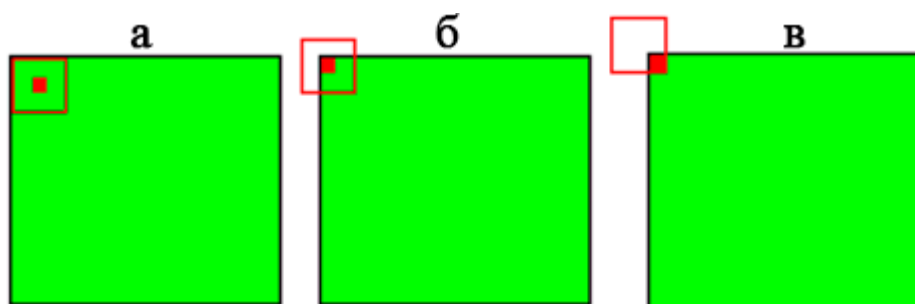


Рисунок 1.5 – Виды свертки: а – valid, б – same, в – full

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

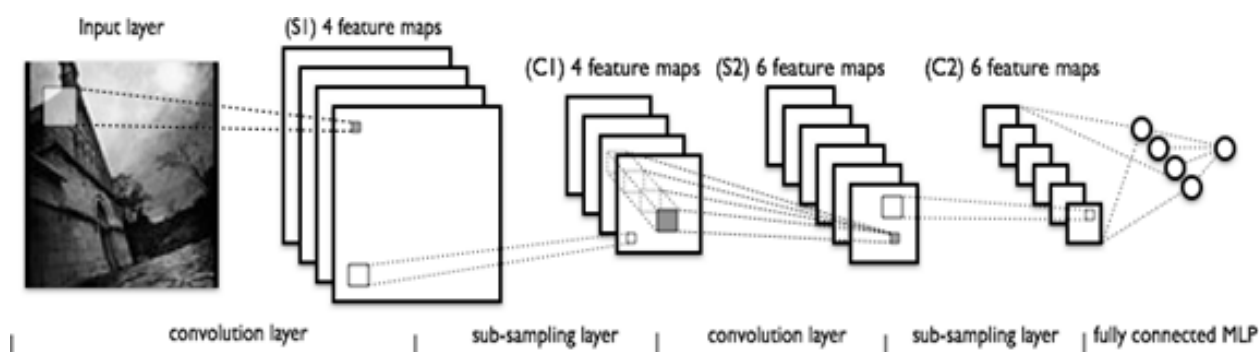


Рисунок 1.6 – Структура СНС



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 распознавания. В сверточной сети – наоборот, общие веса позволяют 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  $2 \times 2$ , что позволяет уменьшить предыдущие карты сверточного слоя в 2 раза. Вся карта признаков разделяется на ячейки  $2 \times 2$  элемента, из которых выбираются максимальные по значению (операция MaxPooling):

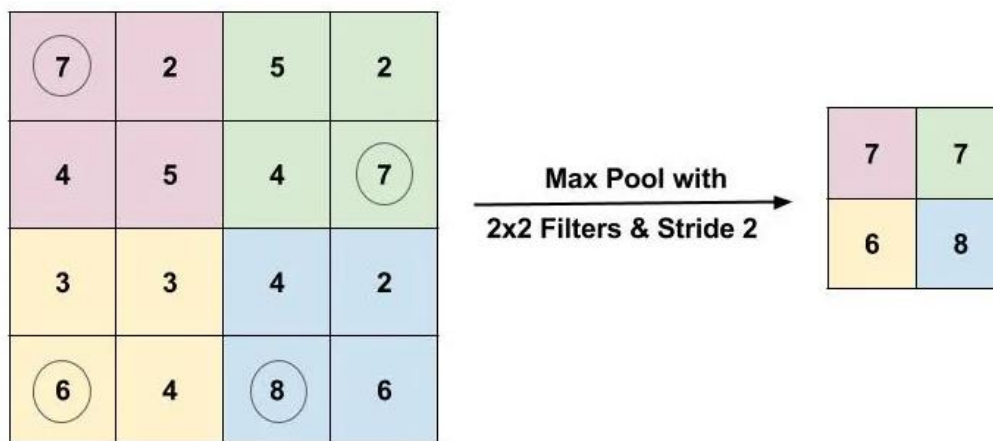


Рисунок 1.7 – Преобразование карты светочного слоя ядром подвыборочного слоя

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.

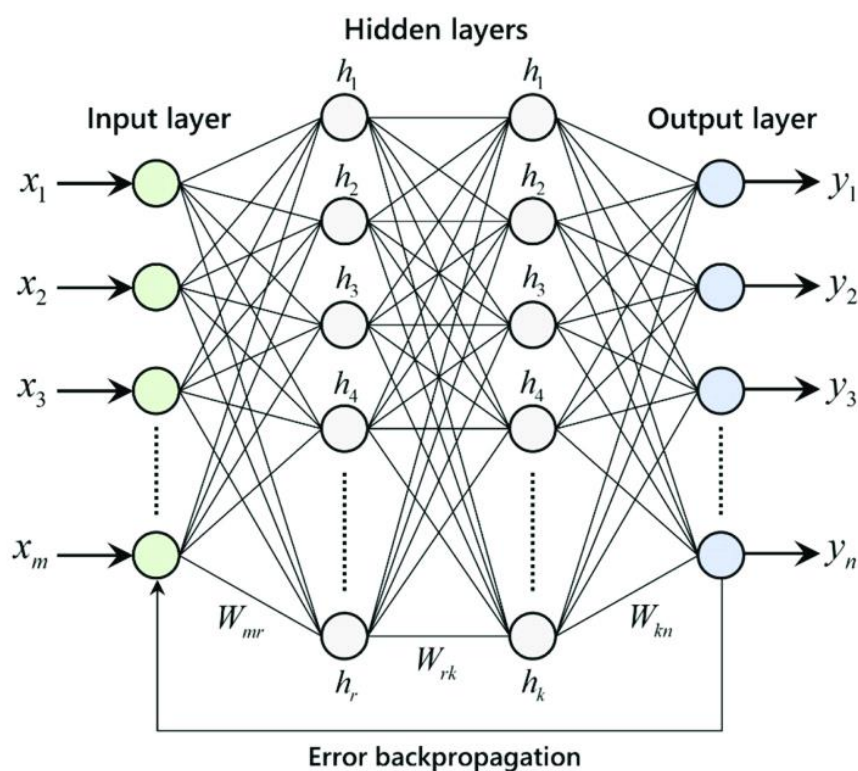


Рисунок 1.8 – Глубокая нейронная сеть

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.

## **1.2 Обучение нейронных сетей**

### **1.2.1 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) – весь набор данных прошел через нейронную сеть в прямом и обратном направлении один раз. Малое число эпох приводит к недостатку обучения, а избыток – к переобучению. Обычно число эпох от 5 до 10.

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

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.

Различают алгоритмы обучения с учителем и без учителя.

#### **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:

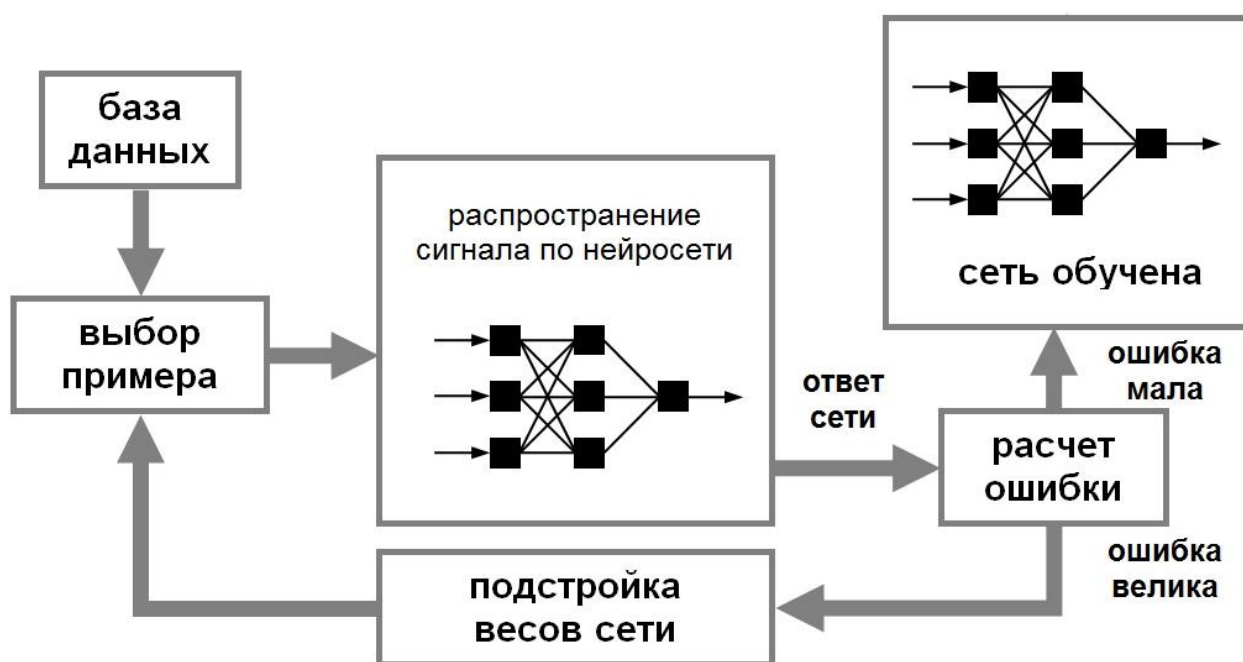


Рисунок 1.9 – Обучение нейронной сети с учителем

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.

### **1.2.2 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 оптимизацией гиперпараметров. 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.

### **1.2.3 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 сетей классической функцией активации сигмоиды.



Возникновение проблемы взрывающегося градиента можно определить по следующим признакам:

- высоком значении 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).

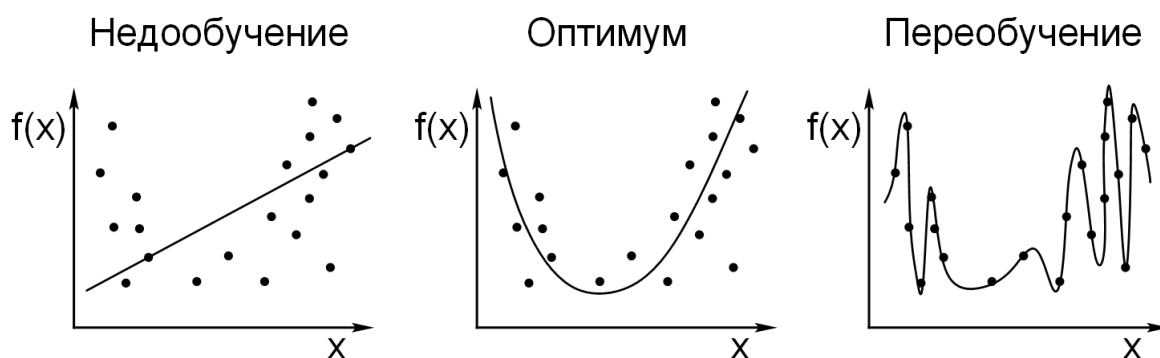


Рисунок 1.10 – Качество обучения

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.

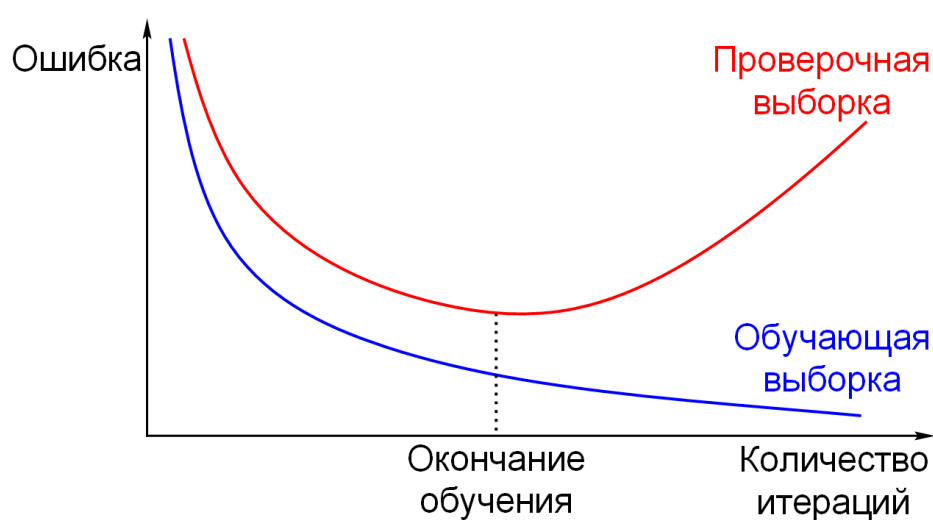


Рисунок 1.11 – График ошибки переобученной сети

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 взаимoadaptацию нейронов на этапе обучения.

#### **1.2.4 Алгоритмы обучения нейронных сетей**

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

*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 вернуть 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 искоемых 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.

### **1.2.5 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:

		True Label	
		Positive	Negative
Predicted Label	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

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.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (2.1)$$

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. С другой стороны, precision высока, когда: модель делает много корректных предсказаний класса Positive (максимизирует True Positive метрику) и делает меньше неверных Positive классификаций (минимизирует False Positive).

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

$$Precision = \frac{TP}{TP + FP} \quad (2.2)$$

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.

$$Recall = \frac{TP}{TP + FN} \quad (2.3)$$



## **ГЛАВА 2. СОЗДАНИЕ НЕЙРОННОЙ СЕТИ НА PYTHON**

### **2.1 Постановка задачи: классификация изображений**

Каждый день врачи получают множество рентгеновских снимков, результатов электрокардиограмм, эндоскопий и много других изображений и материалов, которые надо проанализировать, чтобы поставить диагноз. Это невероятно масштабная работа. Из-за количества материала, который нужно обработать, многие пациенты узнают свой диагноз с задержкой. Уже сейчас есть возможность ускорить и автоматизировать этот процесс посредством применения нейронных сетей.

Целью данной работы является построение модели сверточной нейронной сети, которая будет обучаться на парах (изображение, класс) и в дальнейшем анализировать рентгеновские снимки, относя их к одному из двух классов (здоровый/пневмония).

### **2.2 Инструментарий: Python, библиотеки TensorFlow, Keras**

#### **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.

Множество библиотек Python помогают существенно уменьшить количество времени, необходимого для разработки нейронной сети. В данной работе используются TensorFlow и Keras.

## **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.

Каждое вычисление в TensorFlow представляется как граф потока данных или граф вычислений. Граф вычислений является моделью, описывающей как, будут выполняться вычисления. У графа три составляющие:

- тензоры, хранящие состояние сети (переменные)
- тензоры-операции (арифметические, функции активации и т.д.)
- метод обратного распространения ошибки

Тензор, в данном случае, может быть матрицей, вектором или числом.

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

## **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 Rich Text, 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.

## **Прочие библиотеки**

Библиотека алгоритмов компьютерного зрения OpenCV предназначена для анализа, классификации и обработки изображений, например, для изменения размера или цветового режима.

Библиотека Imgaug позволяет увеличить число изображения в наборе. Она преобразует набор входных изображений в новый, гораздо больший набор

слегка измененных изображений. Это метод увеличения недостающих данных (data augmentation). Увеличение данных – это генерация новых образцов на основе существующих. Для этого имеющиеся образцы переворачиваются на несколько градусов, немного изменяется их яркость, добавляется шум и так далее.

## **2.3 Подготовка данных**

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

Выбранный набор данных называется "Chest X-Ray Images (Pneumonia)". Рентгеновские снимки были отобраны из историй болезни пациентов в возрасте от одного до пяти лет из женского и детского медицинского центра Гуанчжоу (Китай). Все снимки в наборе были первоначально подготовлены путем удаления всех некачественных или нечитаемых снимков. Диагнозы для изображений были затем оценены двумя опытными врачами, прежде чем их разрешили применять для обучения системы искусственного интеллекта.

Набор данных организован в три папки (train, test, val) и содержит вложенные папки для каждой категории изображений (Pneumonia/Normal). Имеется 5863 рентгеновских снимка (типа JPEG) и две категории (пневмония/нормальный):

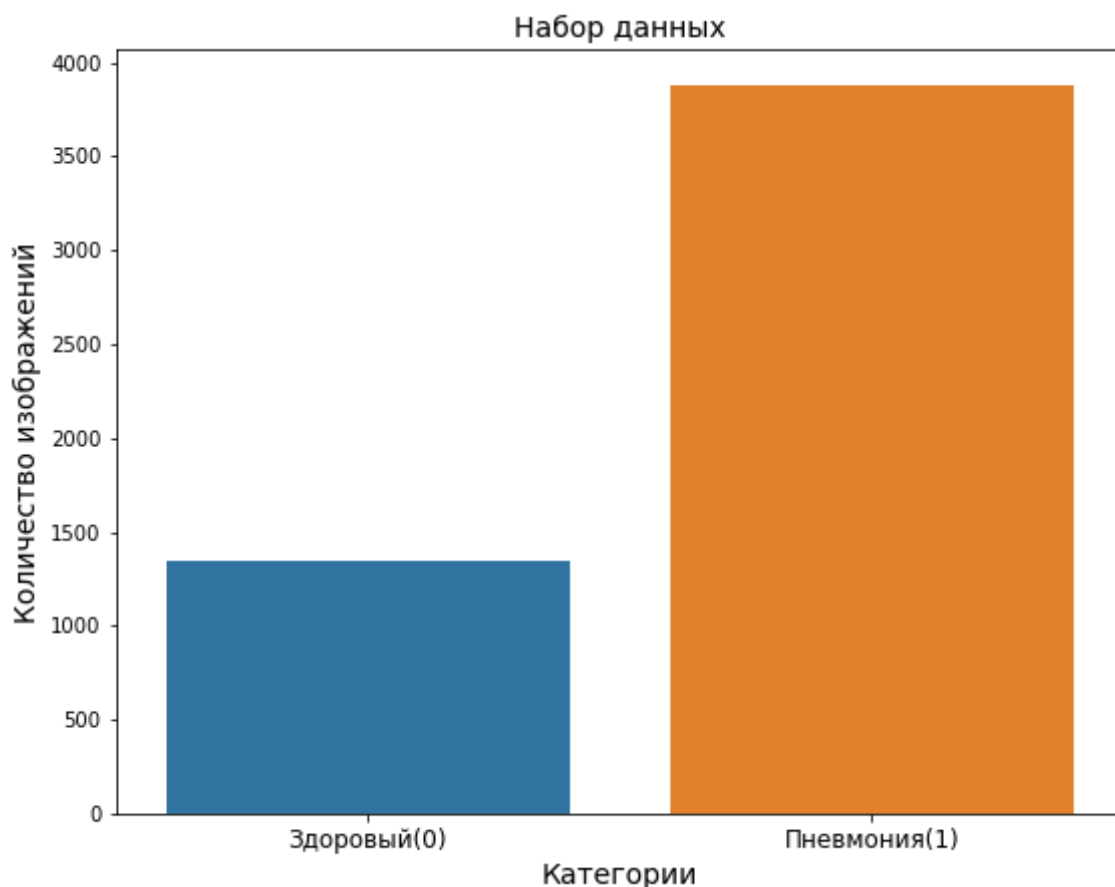


Рисунок 2.1 – Количество образцов из набора данных

Как можно заметить, данные несбалансированны. Случаев пневмонии почти в три раза больше по сравнению со здоровыми легкими. Такая ситуация встречается часто, особенно в медицине. Данные всегда будут несбалансированными: либо будет слишком много нормальных случаев, либо будет слишком много случаев с болезнью.

Взглянув на несколько примеров из выборки можно увидеть, что в некоторых случаях не сложно отличить здоровые легкие от больных. Достоверным признаком пневмонии является затемнение в каком-либо участке легочного поля с нечеткими размытыми контурами:

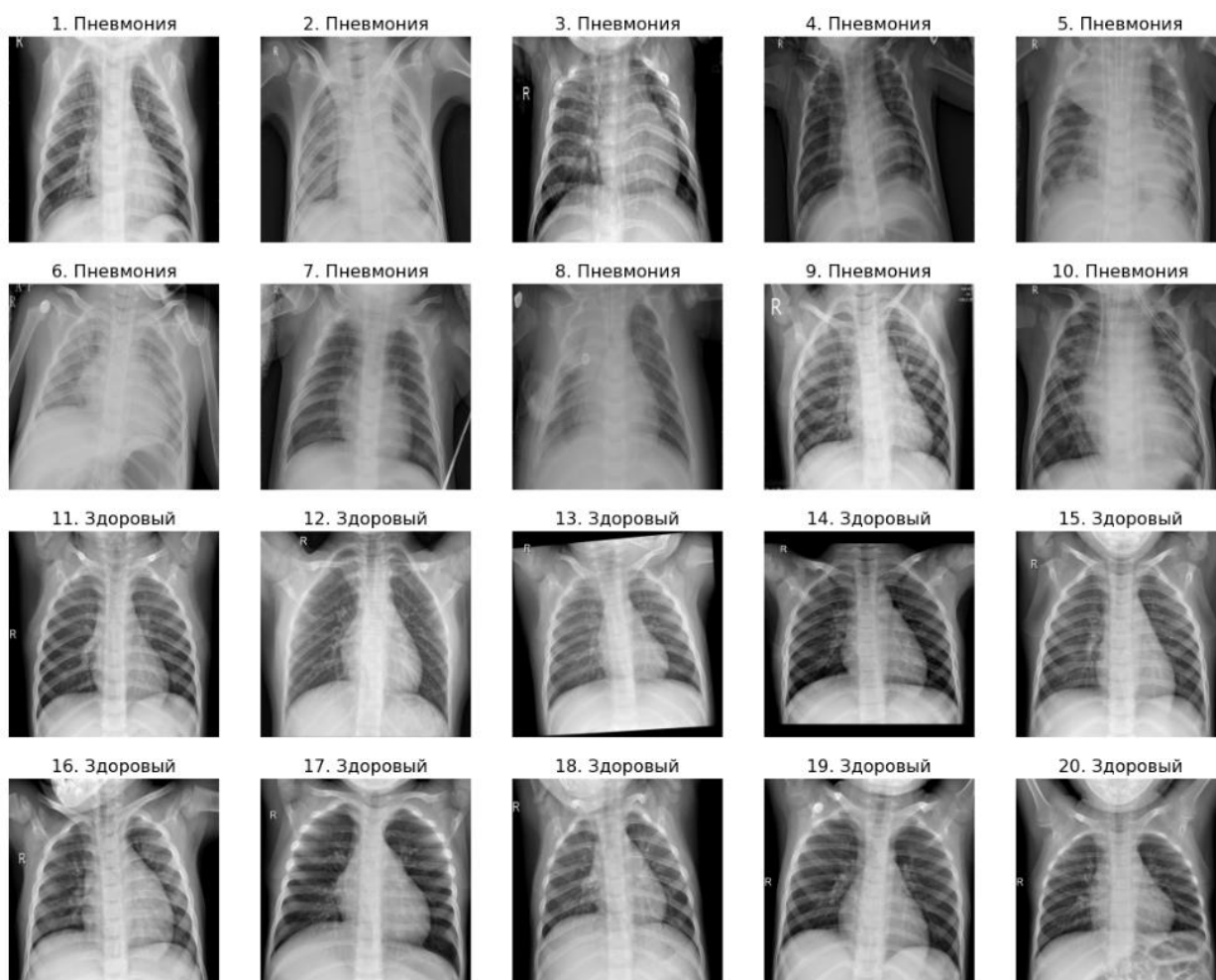


Рисунок 2.2 – Примеры изображений

Лучшими примерами снимков с пневмонией из этой выборки можно назвать изображения с номерами 2, 5, 6, 8:



Рисунок 2.3 – Примеры изображений с меткой "Пневмония"

Хорошие примеры здоровых легких на изображениях с номерами 11, 14, 17, 19:



Рисунок 2.4 – Примеры изображений с меткой "Здоровый"

Отличие в том, что грудная клетка на изображениях с меткой "Здоровый" почти полностью прозрачная, четко видны ребра и сердце, в то время как на изображениях с меткой "Пневмония" этот участок почти полностью белый. Стоит отметить, что так происходит не всегда, и некоторые снимки крайне тяжело однозначно классифицировать, но нейронная сеть с большим числом слоев и нейронов при наличии нескольких эпох обучения и правильной настройке гиперпараметров, скорее всего, выделит определенные закономерности.

Кроме тренировочного набора данные есть также набор для тестирования (test set) – 624 примера, и набор для проверки (validation set) – 16 примеров.

Редко данные в выбранном наборе сразу пригодны для обучения нейронной сети, особенно в случае изображений. Обычно осуществляется подготовка, которая направлена на облегчение процесса оптимизации сети и максимизацию вероятности получения хороших результатов.

Изображения из представленного набора имеют разное разрешение и цветовую гамму. Некоторые изображения представлены в оттенках серого, в то время как большинство из них в RGB модели.

Для унификации изображений, их необходимо перевести в RGB и масштабировать к размеру 224x224 пикселей. Это можно сделать при помощи библиотеки OpenCV. Функция `resize()` изменяет высоту и ширину у изображения, а для конвертации в RGB функция `cvtColor()`:

```
# путь к проверочным изображениям
normal_cases_dir = val_dir / 'NORMAL'
pneumonia_cases_dir = val_dir / 'PNEUMONIA'

# получаем изображения
normal_cases = normal_cases_dir.glob('*.jpeg')
pneumonia_cases = pneumonia_cases_dir.glob('*.jpeg')

valid_data = []
valid_labels = []

# приведем все проверочные данные к одному виду
for img in normal_cases:
    img = cv2.imread(str(img))
    img = cv2.resize(img, (224,224))
    if img.shape[2] == 1:
        img = np.dstack([img, img, img])
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = img.astype(np.float32)/255.
    label = to_categorical(0, num_classes=2)
    valid_data.append(img)
    valid_labels.append(label)
```

Рисунок 2.5 – Листинг обработки изображений

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).

Поскольку данные в наборе несбалансированны, нужно использовать метод увеличения недостающих данных. Для этого была использована библиотека `Imgaug`. Функция `OneOf()` будет использовать один из методов увеличения и применять его к некоторым образцам с меткой `Normal`. В качестве методов увеличения были выбраны горизонтальный переворот



(Fliplr()), поворот на 20 градусов (Affine(rotate=20)) и случайное изменение яркости (Multiply((1.2, 1.5))).

```
seq = iaa.OneOf([
    iaa.Fliplr(), # переворачиваем по горизонтали
    iaa.Affine(rotate=20), # меняем угол наклона
    iaa.Multiply((1.2, 1.5))] # выставляем случайную яркость
```

Рисунок 2.6 – Листинг увеличения данных

Объект "seq" позже передается в генератор для получения данных, который передает изображения партиями (batch), и, если в такой партии изображений с меткой 0 (здоровый) окажется меньше, чем с меткой 1, новые изображения будут добавлены. В итоге число изображений с разными метками будет одинаковое.

После проделанных преобразований изображения можно использовать для качественного обучения.

## 2.4 Обучение и тестирование модели

### Создание модели

Для свертки изображения используем слои SeparableConv и Conv. SeparableConv вводит меньшее количество параметров по сравнению с обычной сверткой (Conv), и, поскольку к каждому каналу применяются разные фильтры, он захватывает больше информации.

Необходимо использовать пакетную нормализацию (BatchNormalization) со слоями свертки. Пакетная нормализация — 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.

После нескольких сверточных слоев необходимо применять операцию подвыборки (MaxPooling). Операция подвыборки – это процесс сжатия (уменьшения размеров) изображения путём сложения значений блоков пикселей.

Далее добавляются плотные полносвязные (dense) слои. Слой Dense состоит из нейронов, соединённых синапсами с элементами входного тензора по его последнему индексу. Количество нейронов в этих слоях подбирается, как правило, экспериментально.

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.

Единственный Flatten слой используется для конвертации входящих данных в меньшую размерность. Он не имеет параметров для обучения, задача этого слоя состоит в преобразовании многомерного входного тензора в одномерный.

В качестве функции активации выбрана ReLU. Вычисление сигмоиды и гиперболического тангенса требует выполнения ресурсоемких операций, таких как возведение в степень, в то время как вычисление ReLU проще. Кроме того, ReLU не подвержена насыщению (затуханию градиента). Применение ReLU 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:

```

# создаем модель
def build_model():
    input_img = Input(shape=(224,224,3), name='ImageInput')
    x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1_1')(input_img)
    x = Conv2D(64, (3,3), activation='relu', padding='same', name='Conv1_2')(x)
    x = MaxPooling2D((2,2), name='pool1')(x)

    x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2_1')(x)
    x = SeparableConv2D(128, (3,3), activation='relu', padding='same', name='Conv2_2')(x)
    x = MaxPooling2D((2,2), name='pool2')(x)

    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_1')(x)
    x = BatchNormalization(name='bn1')(x)
    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_2')(x)
    x = BatchNormalization(name='bn2')(x)
    x = SeparableConv2D(256, (3,3), activation='relu', padding='same', name='Conv3_3')(x)
    x = MaxPooling2D((2,2), name='pool3')(x)

    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_1')(x)
    x = BatchNormalization(name='bn3')(x)
    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_2')(x)
    x = BatchNormalization(name='bn4')(x)
    x = SeparableConv2D(512, (3,3), activation='relu', padding='same', name='Conv4_3')(x)
    x = MaxPooling2D((2,2), name='pool4')(x)

    x = Flatten(name='flatten')(x)
    x = Dense(1024, activation='relu', name='fc1')(x)
    x = Dropout(0.7, name='dropout1')(x)
    x = Dense(512, activation='relu', name='fc2')(x)
    x = Dropout(0.5, name='dropout2')(x)
    x = Dense(2, activation='softmax', name='fc3')(x)

    model = Model(inputs=input_img, outputs=x)
    return model

```

Рисунок 2.7 – Листинг создания модели

Для создания модели в Keras необходимо создать экземпляр класса Model, в который передать входные значения (InputLayer) и выходные (это все остальные слои).

Для удобства дальнейшего использования лучше оформить создание модели в виде функции. Модель получает на вход изображение (матрицу) размером 224x224 пикселя в трехканальном цветовом режиме (слой InputLayer).

Переменная "x" типа "KerasTensor" последовательно приравнивается новому слою и получает предыдущий слой:

$$x = (\text{класс слоя})(\text{предыдущий слой})$$

Слои Conv2D и SeparableConv2D осуществляют свертку изображения, при этом растет число распознаваемых параметров. Всего используется 10 таких слоев. Эти слои параметризуются 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.

Слои MaxPooling2D получают на вход размер ядра 2x2 и название слоя. Они уменьшают размер изображения в два раза.

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

Вся свертка происходит в несколько этапов, пока исходная матрица не уменьшается до размера 14x14. После этого слой Flatten преобразует матрицу в вектор, который далее и подается на вход Dense слоев модели. Полносвязные Dense слои получают на вход количество нейронов, функцию активации и имя.

Два слоя Dropout должны снизить эффект переобучения. Они параметризуются долей отклонения.

Последний слой в сети типа Dense имеет два выходных нейрона, соответствующих количеству распознаваемых классов и использует функцию активации softmax, в отличие от других слоев. Softmax эффективно выбирает самое большое из получаемых значений.

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

Model: "model\_1"

Layer (type)	Output Shape	Param #
ImageInput (InputLayer)	[(None, 224, 224, 3)]	0
Conv1_1 (Conv2D)	(None, 224, 224, 64)	1792
Conv1_2 (Conv2D)	(None, 224, 224, 64)	36928
pool1 (MaxPooling2D)	(None, 112, 112, 64)	0
Conv2_1 (SeparableConv2D)	(None, 112, 112, 128)	8896
Conv2_2 (SeparableConv2D)	(None, 112, 112, 128)	17664
pool2 (MaxPooling2D)	(None, 56, 56, 128)	0
Conv3_1 (SeparableConv2D)	(None, 56, 56, 256)	34176
bn1 (BatchNormalization)	(None, 56, 56, 256)	1024
Conv3_2 (SeparableConv2D)	(None, 56, 56, 256)	68096
bn2 (BatchNormalization)	(None, 56, 56, 256)	1024
Conv3_3 (SeparableConv2D)	(None, 56, 56, 256)	68096
pool3 (MaxPooling2D)	(None, 28, 28, 256)	0
Conv4_1 (SeparableConv2D)	(None, 28, 28, 512)	133888
bn3 (BatchNormalization)	(None, 28, 28, 512)	2048
Conv4_2 (SeparableConv2D)	(None, 28, 28, 512)	267264
bn4 (BatchNormalization)	(None, 28, 28, 512)	2048
Conv4_3 (SeparableConv2D)	(None, 28, 28, 512)	267264
pool4 (MaxPooling2D)	(None, 14, 14, 512)	0
flatten (Flatten)	(None, 100352)	0
fc1 (Dense)	(None, 1024)	102761472
dropout1 (Dropout)	(None, 1024)	0
fc2 (Dense)	(None, 512)	524800
dropout2 (Dropout)	(None, 512)	0
fc3 (Dense)	(None, 2)	1026
Total params: 104,197,506		
Trainable params: 104,194,434		
Non-trainable params: 3,072		

Рисунок 2.8 – Сводка по модели

Суммарно модель имеет 104194434 обучаемых параметров. Наличие необучаемых параметров означает, что 3072 веса не будут обновляться во время обучения с обратным распространением. Эти параметры появляются при использовании слоев BatchNormalization.

## ImageNet

ImageNet — это крупная база данных с аннотированными (обобщенными) изображениями, включает около 1000 категорий изображений с примечаниями.

Для увеличения точности предсказаний будет лучше, если первые несколько слоев сети будут предварительно обучены в imagenet. Это связано с тем, что первые несколько слоев фиксируют общие детали, такие как цветные пятна, края объекта и т.д. Вместо случайно инициализированных весов для этих слоев лучше сразу их настроить точно, это позволит сделать обучение гораздо эффективнее. Для этого необходимо загрузить файл с весами и установить их в выбранные слои используя метод `model.layers.set_weights`:

```
# путь к файлам с весами VGG16
f = h5py.File('../models/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5', 'r')

# выбираем слои, в которые загрузим веса (это пара первых слоев свертки)

w,b = f['block1_conv1']['block1_conv1_W_1:0'], f['block1_conv1']['block1_conv1_b_1:0']
model.layers[1].set_weights = [w,b]

w,b = f['block1_conv2']['block1_conv2_W_1:0'], f['block1_conv2']['block1_conv2_b_1:0']
model.layers[2].set_weights = [w,b]

w,b = f['block2_conv1']['block2_conv1_W_1:0'], f['block2_conv1']['block2_conv1_b_1:0']
model.layers[4].set_weights = [w,b]

w,b = f['block2_conv2']['block2_conv2_W_1:0'], f['block2_conv2']['block2_conv2_b_1:0']
model.layers[5].set_weights = [w,b]

f.close()
```

Рисунок 2.9 – Загрузка готовых весов на первые слои

Теперь слои с индексами 1, 2, 4, 5 имеют определенные веса, это ускорит обучение модели и сделает ее в разы точнее.

В работе использовались веса модели VGG16 – это модель сверточной нейронной сети, достигающая точности 92.7% при тестировании на ImageNet в задаче распознавания объектов на изображении.

## **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 accuracy) перестает улучшаться в течении пяти эпох обучения. Это поможет во многих степени снизить эффект переобучения.

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 `binary_crossentropy`), оптимизатор (Adam) и список метрик (accuracy):

```
# устанавливаем гиперпараметры
opt = Adam(learning_rate=0.0001, decay=1e-5)
es = EarlyStopping(patience=5)
chkpt = ModelCheckpoint(filepath='best_model_todate', save_best_only=True, save_weights_only=True)
model.compile(loss='binary_crossentropy', metrics=['accuracy'], optimizer=opt)
```

Рисунок 2.10 – Настройка гиперпараметров

Для того, чтобы обработать сразу несколько тысяч фотографий необходимо написать функцию-генератор, которая будет передавать изображения небольшими партиями. Размер партии (`batch_size`) был установлен на 16. 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. Также в генератор передается объект "seq", определенный ранее, отвечающий за увеличение данных.

Количество эпох установлено на 20. Это значение взято с запасом, так как 10 эпох (часто выбирается именно такое число) 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 необходимо 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:

```
# обучение модели
history = model.fit(train_data_gen, epochs=nb_epochs, steps_per_epoch=nb_train_steps,
                    validation_data=(valid_data, valid_labels), callbacks=[es, chkpt],
                    class_weight={0:1.0, 1:0.4})
model.save("../models/pneumo_model3.h5")
```

Рисунок 2.11 – Запуск обучения



В параметр `class_weight` передается примерное соотношение классов. После обучения модель лучше сразу сохранить методом `save()`, чтобы использовать при следующем запуске (метод `load_model()`). Вес данной модели 1,16 Гб.

Использование GPU для обучения существенно ускоряет этот процесс, относительно использования CPU. Обучение данной нейронной сети на видеокарте NVIDIA RTX 2060 6gb завершается в среднем за 15 минут. Этот факт обусловлен более удачной для таких вычислений архитектурой ядра, позволяющей GPU эффективно справляться 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.

## **Оценка результатов**

После нескольких запусков обучения модели выяснилось, что точность (Accuracy) на обучающей выборке достигает 98% примерно на десятой эпохе, после чего срабатывает ранняя остановка, так как точность дальше либо не увеличивается, либо увеличивается крайне медленно.

Для проверки на тестовом наборе вызывается метод `evaluate(test_data, test_labels, batch_size)`. Точность на тестовом наборе ожидаемо оказалась меньше и в среднем составила 80%.

Расчет матрицы ошибок на основе тестового набора:

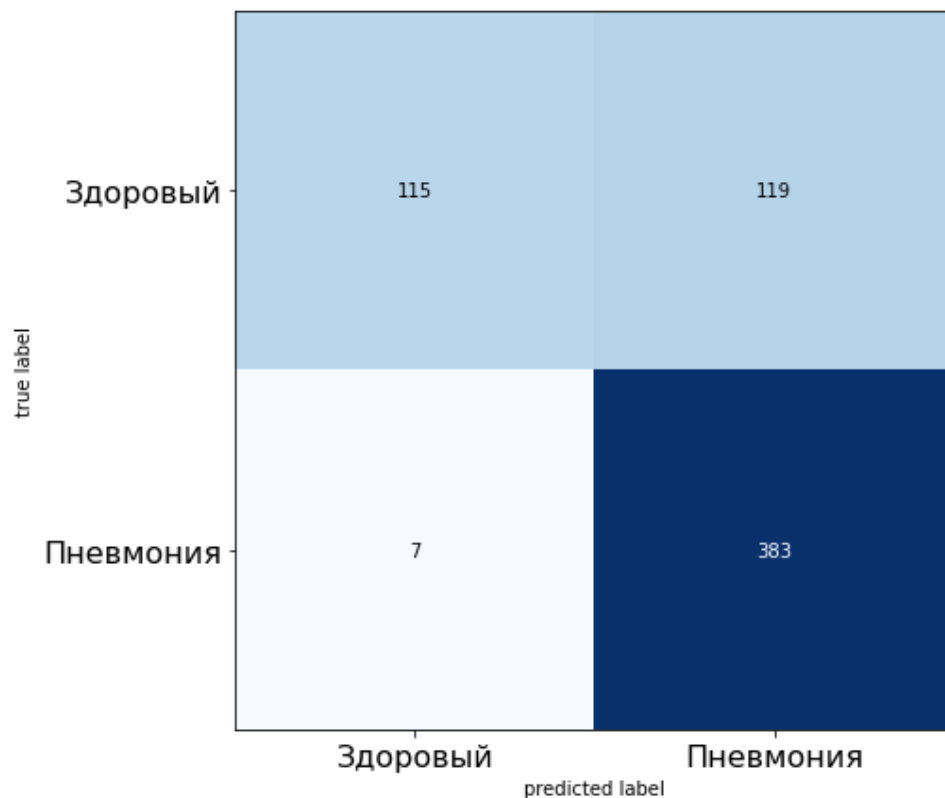


Рисунок 2.12 – Матрица ошибок

Как видно из матрицы ошибок, модель редко принимает больного за здорового (7, 1%) но часто принимает здорового за больного (119, 19%).

Когда набор данных несбалансированный, Ассигасу не является объективным показателем. Например, если набор данных содержит 95 отрицательных и 5 положительных примеров, наличие модели с точностью 95% ни о чем не говорит. Классификатор может помечать каждый пример как отрицательный и все равно достигать 95% точности исходя из формулы расчета этой метрики. Следовательно, нужно использовать другие показатели – Precision и Recall.

Расчет метрик нейронной сети:

- Accuracy – 0.80
- Precision – 0.76
- Recall – 0.98

В данном случае можно полагать, что обученная нейросеть будет верно классифицировать изображения с вероятностью 76%.

Для классификации новых изображений вызывается метод `predict()`. Функция, классифицирующая все изображения из указанной директории:

```
# загрузка изображений
def predict(path):
    counter = 1
    img_dir = Path(path) #pathlib
    types = ('*.jpg', '*.jpeg', '*.png')
    glob_images = []

    #добавляем в list изображения всех типов
    for img in types:
        glob_images.extend(img_dir.glob(img))

    pred_img = []

    #унифицируем
    for img in glob_images:
        img = cv2.imread(str(img))
        img = cv2.resize(img, (224,224))
        if img.shape[2] == 1:
            img = np.dstack([img, img, img])
        else:
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            img = img.astype(np.float32)/255.
            label = to_categorical(0, num_classes=2)
            pred_img.append(img)

    #переводим в numpy массив для дальнейшей работы
    images = np.array(pred_img)

    for img in images:
        #классифицируем
        predictions_single = model.predict(images)
        #вывод названия изображения
        print(glob_images[counter-1])
        #вывод вероятностей
        print("{} | Здоровый: {:.2f}".format(counter, predictions_single[counter-1][0]),
              " Больной: {:.2f}".format(predictions_single[counter-1][1]))
        #отрисовка изображений matplotlib
        img = mimg.imread(glob_images[counter-1])
        imgplot = plt.imshow(img)
        plt.show()
        counter += 1
```

Рисунок 2.13 – Классификация загруженных изображений

Для классификации одного нового изображения была написана небольшая программа с визуальным интерфейсом на Python с использованием

библиотеки PyQt. Программа позволяет выбрать изображение из указанной директории и получить вероятностную оценку состояния легких:

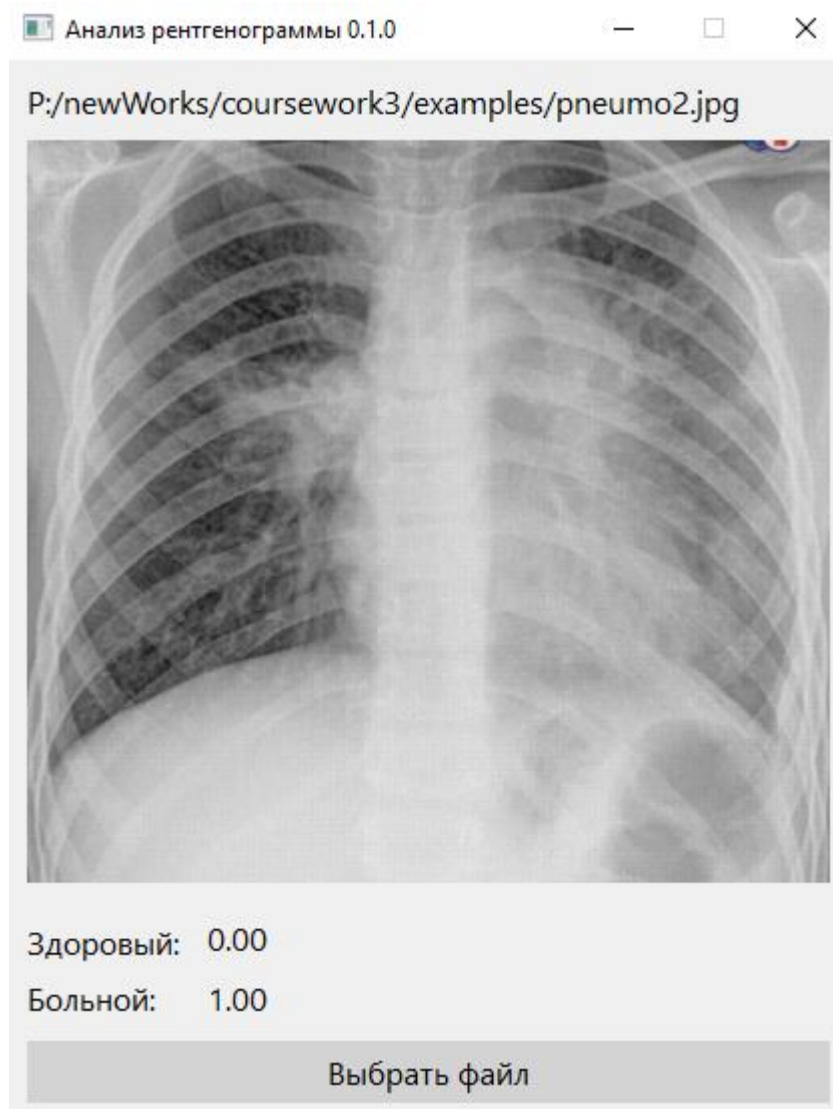


Рисунок 2.14 – Программа для классификации изображения

Для запуска данной программы на любом компьютере без Python и установленных библиотек она была скомпилирована в исполняемый файл .EXE при помощи библиотеки "auto-py-to-exe". При этом все необходимы библиотеки автоматически включаются в директорию с исполняемым файлом.

## Заключение

Машинное обучение широко распространено в здравоохранении, но применять его не просто, так как вопрос касается здоровья и жизней людей, но его осторожное применение может принести огромную пользу. На данный момент заключение врача гораздо значимей результата классификации нейронной сети. Тем не менее нейросети несравнимо быстрее обрабатывают большие объемы информации и могут дать представление о ситуации в целом или указать пациентов, на исследования которых следует обратить особое внимание.

В ходе выполнения курсовой работы была спроектирована и реализована нейронная сеть, классифицирующая рентгеновские снимки легких на здоровые и с признаками пневмонии. Вероятность верной классификации после нескольких циклов обучения составила порядка 76%.

Результаты исследования показывают, что использование глубоких нейронных сетей в медицине возможно уже сейчас: при наличии качественного размеченного набора данных, оптимально настроенных гиперпараметров, удачно выбранной архитектуре модели с подходящими слоями можно добиться хороших результатов даже в сложных задачах. Однако, сильно полагаться на результаты работы нейронных сетей пока не стоит, но точность предсказаний нейросетей увеличивается с каждым годом, и, можно заключить, что в какой-то момент она превзойдет точность человека.

Цель работы – создание нейронной сети и ее обучение – была достигнута, поставленные задачи выполнены.

Направлением для дальнейшей работы может быть увеличение точности предсказаний путем изменения модели, гиперпараметров или повышения качества изображений, написание полноценной программы с визуальным интерфейсом.

## 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