

# Оглавление

<b>Искусственный интеллект</b>	<b>7</b>
Введение.	7
Определение понятия искусственный интеллект. Классификация разделов ИИ.	9
<b>Машинное обучение.</b>	<b>11</b>
История:	11
Сферы применения машинного обучения:	12
Общая постановка задачи машинного обучения (обучения по прецедентам):	14
Способы машинного обучения:	14
Классические задачи, решаемые с помощью машинного обучения:	15
<b>Глубокое обучение</b>	<b>16</b>
Строение нейрона и функции активации:	16
Нейронные сети - основы:	18
Классификация нейронных сетей.	20
Архитектуры нейронных сетей:	20
Платформы для глубокого обучения:	25
<b>Создание нейронной сети.</b>	<b>27</b>
Введение, область применения, постановка задачи	27
Подключение необходимых библиотек	28
Изменение ускорителя на TPU	28
Загрузка данных.	29
Обработка данных	30
Визуализация датасета	31
Построение и компиляция модели	33
Обучение модели	34
Проверка точности распознавания на тестовых данных	36
Улучшение и переобучение модели	37
Повторная проверка точности распознавания на тестовых данных:	40
<b>Вывод:</b>	<b>41</b>

## РЕЗЮМЕ

**Курсовой проект:** 39 стр., 11 рис., 5 источников.

МАШИННОЕ ОБУЧЕНИЕ С ПОМОЩЬЮ БИБЛИОТЕКИ  
TENSORFLOW.

КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ, СОЗДАНИЕ И ОБУЧЕНИЕ  
НЕЙРОСЕТИ.

**Объект исследования:** нейронные сети для распознавания и классификации.

**Предмет исследования:** особенности создания и обучения нейронной сети с помощью библиотеки TensorFlow.

**Цель работы:** изучить способы создания и обучения нейронной сети с помощью библиотеки TensorFlow.

**Методы исследования:** графический, наблюдения, сравнительного анализа.

**Исследования и разработки:** изучены способы создания и обучения нейронной сети с помощью библиотеки TensorFlow.

**Область возможного практического применения:** учебный процесс факультета математики и информатики ГрГУ им. Я.Купалы (ФаМИ ГрГУ) и аналогичных организаций.

Автор работы подтверждает, что приведенный в курсовом проекте расчётно-аналитический материал правильно и объективно отражает состояние исследуемого процесса, а все заимствованные из литературных источников теоретические, методологические и методические положения и концепции сопровождаются ссылками на их авторов.

## SUMMARY

**Course project:** 39 pages, 11 drawings, 5 sources.

MACHINE LEARNING WITH TENSORFLOW LIBRARY.  
IMAGECLASSIFICATION, CREATION AND TRAINING OF A NEURAL  
NETWORK.

**Object of study:** neural networks for recognition and  
classification.

**Subject of study:** features of creating and training a neural network  
using the TensorFlow library.

**The purpose of the work:** to study how to create and train a neural  
network using the TensorFlow library.

**Research methods:** graphic, observation, comparative analysis.

**R&D:** Learned how to build and train a neural network using the  
TensorFlow library.

**Area of possible practical application:** the educational process of  
the Faculty of Mathematics and Informatics of the GrSU. Y. Kupala (FaMI  
GrSU) and similar organizations.

The author of the work confirms that the calculation and  
analytical material given in the course project correctly and  
objectively reflects the state of the process under study, and all  
theoretical, methodological and methodological provisions and  
concepts borrowed from literary sources are accompanied by  
references to their authors.

# Искусственный интеллект

## Введение.

Темпы развития отрасли искусственного интеллекта в целом и машинного обучения как подобласти ИИ в 21 веке можно поистине назвать революционными. Этому безусловно способствовали как огромный объём знаний, собранный за несколько столетий, так и быстрыми темпами растущая производительность вычислительной техники. Благодаря существенному росту производительности процессоров и видеокарт стало возможным дальнейшее активное развитие отрасли ИИ. По прогнозам специалистов, за машинным обучением — будущее. По мере того, как люди становятся все более зависимыми от техники, машин и гаджетов, грянет мировая технологическая революция, благодаря которой появятся новые профессии и исчезнут старые.

На данный момент область применения машинного обучения существенно ограничена из-за относительной «молодости» отрасли. Сейчас мы находимся на этапе слабого (иногда еще называют узким) искусственного интеллекта, при котором машина может справляться лучше человека только с ограниченным видом задач (например, распознать что изображено на картинке, перевести аудио в текст, научиться играть в шахматы и т.д.). Основная задача специалистов в области ИИ – достичь следующего этапа развития – этапа общего искусственного интеллекта, главной чертой которого является возможность компьютера решить любую интеллектуальную задачу также, как её бы выполнил среднестатистический человек. При этом важно выделить, что именно интеллектуальных задач, то есть выполнение творческих функций, которые традиционно считаются прерогативой человека. Конечной целью является достижение этапа сильного искусственного интеллекта, при котором компьютер может решить большинство поставленных задач лучше человека – в разы качественней, быстрее, результативней.

Таким образом перспективы и задачи искусственного интеллекта расписаны на столетия вперед и заключаются в решении проблем, связанных с приближением искусственного интеллекта к возможностям человека, а также в создании искусственного разума, представляющего объединение уже созданных систем искусственного интеллекта в единую систему, способную решать глобальные проблемы.

Для изучения машинного обучения с помощью библиотеки TensorFlow мы на основе выбранного датасета создадим модель, которую впоследствии обучим, таким образом изучив все этапы решения актуальных и полезных для общества задач – в моем случае, в области медицины, являющейся сейчас как никогда важной. Также необходимым минимумом для знакомства с библиотекой будем считать краткий экскурс в историю ее создания, обзор конкурентов в отрасли, необходимые в ходе работы дополнительные библиотеки, а также обзор возможностей, предоставляемых библиотекой TensorFlow.

Для более качественного и быстрого обучения необходимо будет качественно спроектировать модель для обучения, правильно выбрав тип нейронной сети, количество слоев и количество нейронов в каждом слое.

## Определение понятия искусственный интеллект. Классификация разделов ИИ.

Искусственный интеллект – научное направление, разрабатывающее методы, позволяющие электронно-вычислительным машинам решать интеллектуальные задачи, другими словами, заниматься моделированием разумного поведения.

Стоит отметить, что ИИ – это достаточно обширная область знаний, включающая в себя множество разделов, таких как:

- 1) Обработка естественного языка (извлечение информации, классификация, машинный перевод, генерирование текста и т.д.) – компьютер должен распознать написанное, обработав, выдать правильный и релевантный ответ.
- 2) Экспертные системы - компьютерные системы, которые имитируют способность принятия решений человеком, основываясь на заданных правилах в формате «если ..., то ...». Количество правил экспертной системы прямо пропорционально зависит как от количества учитываемых входных параметров, так и от желаемой точности каждого из входных параметров. Точность экспертной системы зачастую в большей степени зависит от количества диапазонов классификации выходного параметра, нежели от точности входных параметров. ЭС нашли применение в большом количестве отраслей, таких как медицина, вычислительная техника, военное дело, микроэлектроника, радиоэлектроника, экономика, экология, математика и т.д.
- 3) Зрение (машинное зрение, распознавание изображений) – компьютер распознает те или иные объекты на изображении, может на лету их классифицировать, имеет возможность распознать один и тот же объект в разной обстановке и с разных ракурсов. Применяется в системах видеонаблюдения, автопилотах различных транспортных средств и т.д.

- 4) Речь (Перевод речи в текст, перевод текста в речь) – компьютер должен распознавать речь людей и имеет возможность разговаривать.
- 5) Робототехника – создание роботов для выполнения конкретных функций. Применяется в промышленности, военном секторе, в космосе, выполняя сложную, рутинную и зачастую опасную для жизни человека работу.
- 6) Автоматическое планирование – обычно используется автономными роботами и беспилотными аппаратами, когда им необходимо выполнять последовательность действий при решении комплексных задач.
- 7) Машинное обучение (глубокое обучение, обучение с учителем, обучение без учителя) – обучение происходит не за счет правил, минусом которых является невозможность или трудозатратность применения в ряде сложных задач, а на основе опыта, полученного в ходе обучения на большом наборе данных. То есть, алгоритмы в ходе тренировки на наборе данных становятся все более эффективнее и точнее.

Для лучшего понимания отличий искусственного интеллекта от экспертной системы, приведем пример их обучения игре в шахматы.

Так, экспертная система будет весьма сложной и трудозатратной – ведь разработчику ЭС придётся продумать и учесть все возможные ходы и их комбинации и написать внушительное количество правил, которыми в последствии будет руководствоваться компьютер.

В случае с машинным обучением, компьютер на основе набора данных о прошлых играх сам изучает и анализирует комбинации ходов и их конечные результаты, и на основе этих примеров, а также прошлого опыта, сам создает алгоритмы, нацеленные на выигрыш.

## **Машинное обучение.**

### **Краткая история развития, области применения и классификация.**

Машинное обучение - класс методов искусственного интеллекта, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, математического анализа, методов оптимизации, теории вероятностей, теории графов, различные техники работы с данными в цифровой форме.

#### **История:**

1946: Появился компьютер ЭНИАК — сверхсекретный проект армии США.

1950: Алан Тьюринг создает “Тьюринг тест” для оценки интеллекта компьютера.

1958: Фрэнк Розенблатт придумал Персептрон — первую искусственную нейронную сеть и создал первый нейрокомпьютер «Марк-1».

1959: Марвин Минский создал первую машину SNARC со случайно связанной нейросетью.

1959: Артур Самуэль, исследователь искусственного интеллекта, ввел термин «машинное обучение». Он изобрел первую самообучающуюся компьютерную программу по игре в шашки. Самуэль определил машинное обучение как процесс, в результате которого компьютеры способны показать такое поведение, которое в них не было запрограммировано изначально.



1967: Написан метрический алгоритм по классификации данных. Алгоритм позволил компьютерам применять простые шаблоны распознавания.

1985: Терренс Сейновски создает NetTalk — искусственную нейронную сеть.

1997: Компьютер Deep Blue обыграл чемпиона мира, Гарри Каспарова, в шахматы.

2006: Джеффри Хинтон, ученый в области искусственных нейросетей, ввел термин «Глубинное обучение» (Deep learning).

2011: Эндрю Энг и Джефф Дин основали Google Brain.

2012: Google запускает облачный сервис Google Prediction API для машинного обучения. Он помогает анализировать неструктурированные данные.

2014: В Facebook изобрели DeepFace для распознавания лиц. Точность алгоритма 97%.

2015: Amazon запустила собственную платформу машинного обучения — Amazon Machine Learning.

2015: Microsoft создает платформу Distributed Learning Machine Toolkit, предназначенную для децентрализованного машинного обучения.

2020: Технологии искусственного интеллекта применяются практически в каждом программном продукте.

## **Сферы применения машинного обучения:**

### **1.Образование.**

Благодаря внедрению искусственного интеллекта, разработчики создали обучающие системы, симулирующие поведение учителя. Они могут выявлять уровень знаний учащихся, анализировать их ответы, ставить оценки и даже определять персональный план обучения.

К примеру, AutoTutor, обучает студентов компьютерной грамотности, физике и критическому мышлению. Knewton учитывает характеристику обучения каждого студента и разрабатывает для него уникальную

учебную программу. BBC США используют систему SHERLOCK, чтобы обучить пилотов находить технические неисправности в самолетах.

## 2.Поисковики.

Поисковые системы используют машинное обучение, чтобы улучшить свои функции. Например, Google внедрила машинное обучение в распознавание голоса и поиск изображений. В 2019 году Google представила Teachable Machine 2.0 - самообучающуюся нейронную сеть, способную распознавать звуки речи, интонации и позы. С помощью веб-камеры и микрофона пользователь обучает нейронные сети без написания кода и экспортирует их в сторонние приложения, носители или на веб-сайты.

## 3.Digital-маркетинг.

Машинное обучение в данной сфере обеспечивает глубокую персонализацию клиента. Таким образом, компании могут взаимодействовать с клиентом на личном уровне, становясь к нему ближе. Благодаря алгоритмам сложной сегментации, машина фокусируется на “нужном клиенте в нужное время”, чтобы эффективно продавать продукты. Кроме того, благодаря правильным данным о клиентах, компании располагают информацией, которую можно использовать для изучения их поведения и реакций.

Например, Nova использует машинное обучение для написания электронной рассылки клиентам, делая письма при этом персонализированными. Машина знает, у каких электронных писем ранее была высокая конверсия, и, соответственно, предлагает изменения в рассылках для лучших продаж.

## 4.Здравоохранение.

У IBM есть разработка Watson. Это суперкомпьютер для медицинских исследований, основанный на машинном обучении. Технология Watson for Oncology обрабатывает большой объем медицинских данных, в том числе изображения, на которых можно точно диагностировать рак. Watson for Oncology сейчас используется в

больницах Нью-Йорка, Бангкока и Индии. В июле 2016 года IBM стала сотрудничать с 16 медицинскими центрами и технологическими стартапами, чтобы ускорить развитие программ для точной диагностики.

1. А также:

- распознавание речи, жестов, образов;
- техническая, медицинская диагностика;
- обнаружение спама, мошенничества;
- биржевой технический анализ;

### **Общая постановка задачи машинного обучения (обучения по прецедентам):**

Имеется множество объектов (ситуаций) и множество возможных ответов (откликов, реакций). Существует некоторая зависимость между ответами и объектами, которая неизвестна, но известна конечная совокупность прецедентов — пар «объект, ответ», называемая обучающей выборкой. На основе этих данных требуется восстановить неявную зависимость - построить алгоритм, способный для любого входного объекта выдать достаточно точный классифицирующий ответ. Эта зависимость не всегда выражается аналитически, и здесь нейросети реализуют принцип эмпирически формируемого решения. Важной особенностью при этом является способность обучаемой системы к обобщению, к адекватной обработке данных, выходящих за пределы имеющейся обучающей выборки. Для измерения точности ответов вводится оценочный функционал качества.

### **Способы машинного обучения:**

1.Обучение с учителем — для каждого прецедента задаётся пара «ситуация, требуемое решение»:

- а) Искусственная нейронная сеть - Глубокое обучение
- в) Метод коррекции ошибки
- г) Метод обратного распространения ошибки

2.Обучение без учителя — для каждого прецедента задаётся только «ситуация», требуется сгруппировать объекты в кластеры, используя данные о попарном сходстве объектов, и/или понизить размерность данных:

а) Альфа-система подкрепления

б) Гамма-система подкрепления

в) Метод ближайших соседей

3.Обучение с подкреплением — для каждого прецедента имеется пара «ситуация, принятое решение»: Генетический алгоритм.

4.Активное обучение — отличается тем, что обучаемый алгоритм имеет возможность самостоятельно назначать следующую исследуемую ситуацию, на которой станет известен верный ответ:

5.Обучение с частичным привлечением учителя — для части прецедентов задается пара «ситуация, требуемое решение», а для части — только «ситуация»

6.Трансдуктивное обучение — обучение с частичным привлечением учителя, когда прогноз предполагается делать только для прецедентов из тестовой выборки

7.Многозадачное обучение — одновременное обучение группе взаимосвязанных задач, для каждой из которых задаются свои пары «ситуация, требуемое решение»

8.Многовариантное обучение— обучение, когда прецеденты могут быть объединены в группы, в каждой из которых для всех прецедентов имеется «ситуация», но только для одного из них (причем, неизвестно какого) имеется пара «ситуация, требуемое решение»

9.Бустинг — это процедура последовательного построения композиции алгоритмов машинного обучения, когда каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих алгоритмов.

10.Байесовская сеть

### **Классические задачи, решаемые с помощью машинного обучения:**

1. Классификация, как правило, выполняется с помощью обучения с учителем на этапе собственно обучения.
2. Кластеризация, как правило, выполняется с помощью обучения без учителя
3. Регрессия, как правило, выполняется с помощью обучения с учителем на этапе тестирования, является частным случаем задач прогнозирования.
4. Понижение размерности данных и их визуализация (выполняется с помощью обучения без учителя).
5. Восстановление плотности распределения вероятности по набору данных.
6. Одноклассовая классификация и выявление новизны.
7. Построение ранговых зависимостей.

### **Глубокое обучение. Строение нейрона, нейронные сети, области применения и классификация.**

Глубокое обучение — это разновидность машинного обучения на основе искусственных нейронных сетей. Процесс обучения называется глубоким, так как структура искусственных нейронных сетей состоит из нескольких входных, выходных и скрытых слоев. Каждый слой содержит единицы, преобразующие входные данные в сведения, которые следующий слой может использовать для определенной задачи прогнозирования.

#### **Строение нейрона и функции активации:**

Искусственный нейрон – базовый элемент нейронной сети. Математическая модель искусственного нейрона построена по подобию биологических нейронов (рис.1), но с явными упрощениями.

Схема искусственного нейрона представлена на рис. 2, где  $X_1 \dots X_N$  – входы нейрона,  $W_1 \dots W_N$  – синаптические веса связей нейрона,  $S$  – взвешенная сумма входных значений нейрона,  $F(S)$  – функция активации, значением которой является  $Y$  – выходное значение нейрона.

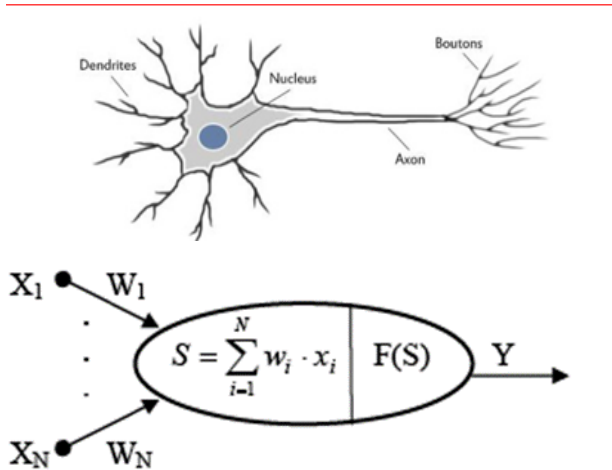


Рис.1. Строение нейрона

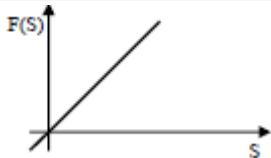
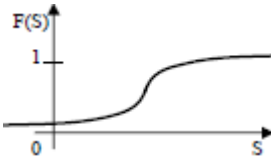
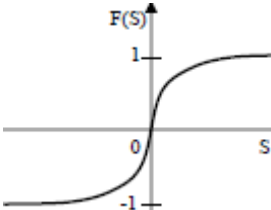
Рис. 2. Формальный нейрон (мат.. модель)

Алгоритм можно описать следующим образом: входы нейрона  $X_1 \dots X_N$ , имеющие некоторые значения, перемножаются на синаптические веса их связей; далее полученные значения суммируются в сумматоре  $S$ , после чего полученное значения подается на в функцию активации, а конечный результат преобразований  $Y$  – выходное значение нейрона.

Функции активации могут различные, наиболее часто используемые представлены в таблице 1 и рисунке 3.

Таблица 1. Основные функции активации

Название	Формула	График (пример)
Пороговая	$F(S) = \begin{cases} 1, & \text{при } S \geq T, \\ 0, & \text{при } S < T, \end{cases}$ $T = const.$	

Линейная	$F(S) = k \cdot S,$ $k = const.$	
Сигмоидальная	$F(S) = \frac{1}{1 + e^{-S \cdot k}},$ $k = const.$	
Гиперболический тангенс	$F(S) = th\left(\frac{S}{k}\right) = \frac{e^{\frac{S}{k}} + e^{-\frac{S}{k}}}{e^{\frac{S}{k}} - e^{-\frac{S}{k}}}$ $k = const.$	

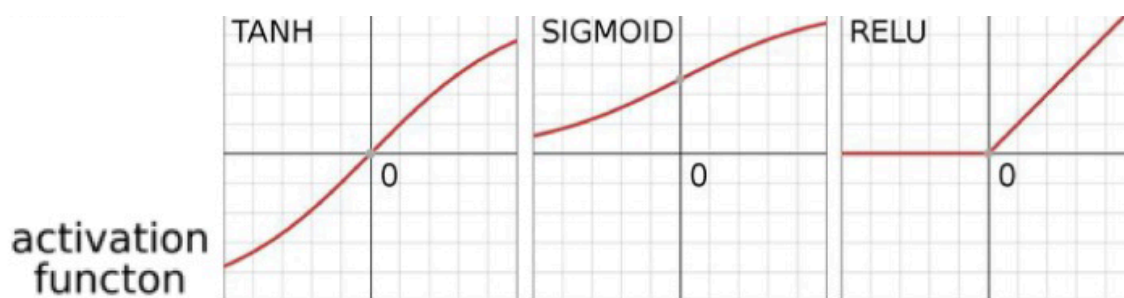


Рис.3. Функции активации нейрона

### Нейронные сети - основы:

Искусственная нейронная сеть – математическая модель, реализуемая программно или аппаратно, построенная по подобию естественных нейронных сетей (сетей нервных клеток живого организма), представляющая собой соединение простых взаимодействующих между собой процессоров - искусственных нейронов.

Схема простейшей полносвязной нейронной сети представлена на рисунке 4. Она состоит из входного слоя, одного скрытого слоя и выходного слоя.

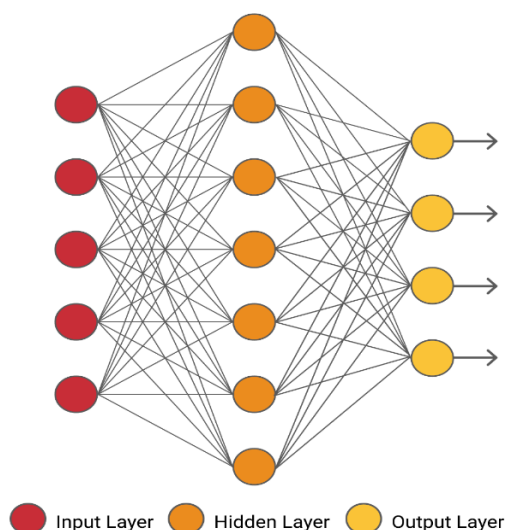


Рис.4. Схема простейшей нейронной сети

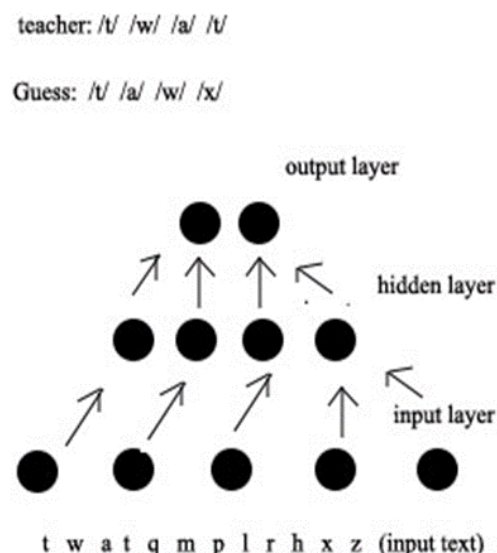


Рис.5. нейронная сеть NetTalk

До 2000-ых годов глубокое обучение было не столь популярным, как сейчас в силу неудовлетворительной производительности процессоров и видеокарт. В то время нейронные сети зачастую имели один или несколько скрытых слоев (например, как нейронная сеть Терренса Сейновски - NetTalk - результат исследований 1980-ых годов, содержащая один скрытый слой – рис. 5), что давало невнушительные результаты: с той же задачей намного успешнее справлялись «классические» по типу графовых алгоритмов. Уровень сложности поставленной перед нейронной сетью задачи, также, как и время её обучения, неразрывно зависит от производительности процессора или видеокарты.

Изучив график производительности процессоров и видеокарт, изображенный на рисунке 6, становится понятным, почему активное развитие нейронных сетей, а также их внедрение в повседневную жизнь человека началось только ближе к середине 2000-ых годов, а сам термин «Глубокое обучение» (Deep learning) был введен ученым в области искусственных нейросетей, Джеффри Хинтоном в 2006 году:

После 2005 года наблюдается быстрыми темпами экспоненциально возрастающая производительность как процессоров, так и видеокарт, но при этом с каждым годом разрыв в производительности между ними в пользу видеокарт также возрастает экспоненциально. Поэтому для обучения нейронных сетей чаще используют GPU.



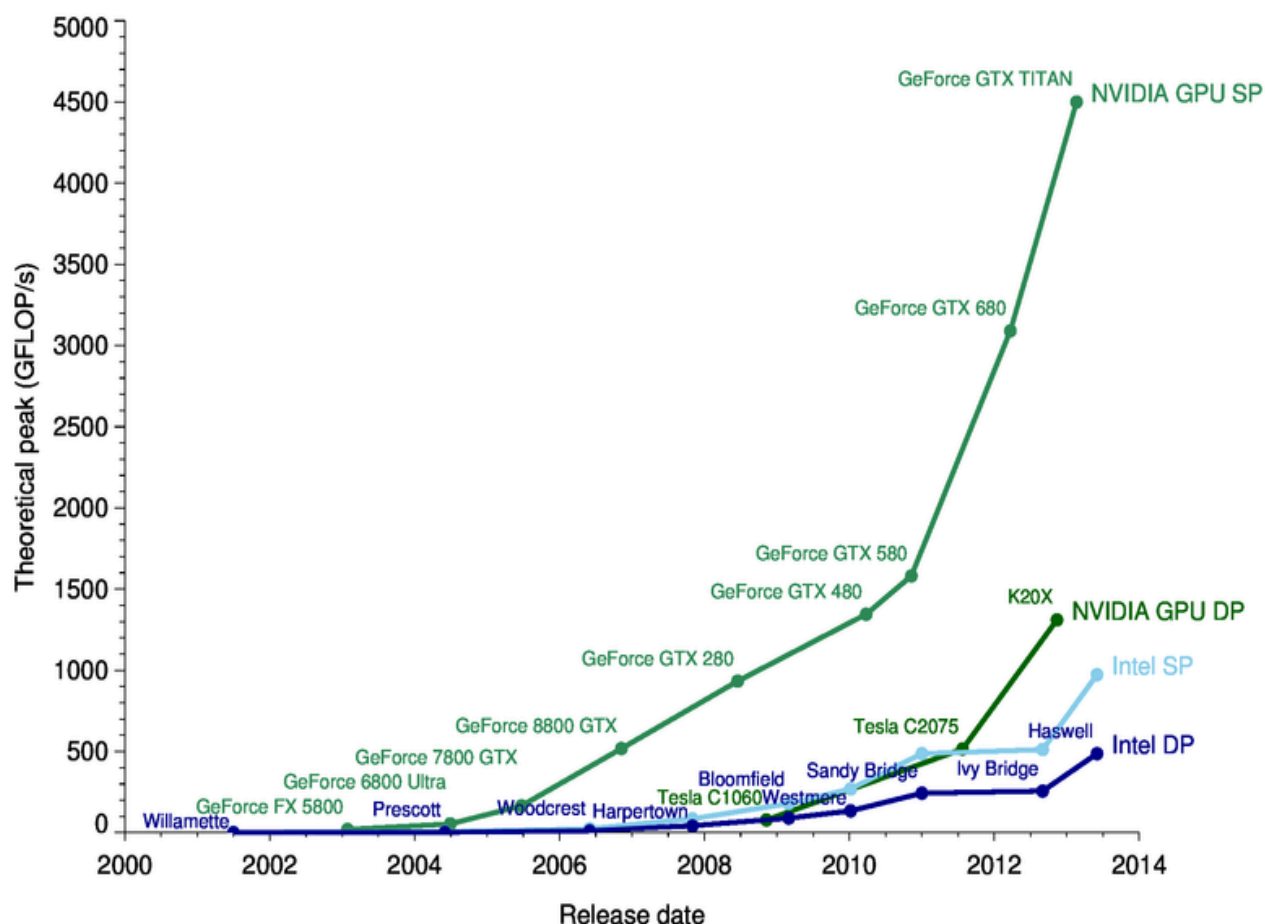


Рис.6. График производительности видеокарт и процессоров

## Классификация нейронных сетей.

Существует множество нейронных сетей, которые классифицируются по нескольким признакам (таблица 2). Наибольшее распространение получили слоистые сети прямого распространения.

Тип	Описание
<i>По топологии</i>	
Полносвязные	Каждый нейрон связан с другим нейроном в сети (из-за высокой сложности обучения не используется).
Слоистые	Нейроны располагаются слоями, каждый нейрон последующего слоя связан с нейронами предыдущего. Есть однослойные и многослойные сети.

<i><b>По типу связей</b></i>	
Прямого распространения	Все связи между нейронами идут от выходов нейронов предыдущего слоя к входам нейронов последующего.
Рекуррентные	Допускаются связи выходов нейронов последующих слоев с входами нейронов предыдущих.
<i><b>По организации обучения</b></i>	
С учителем	При обучении используются обучающие выборки, в которых определены требуемые от сети выходные значения, такие сети используют для решения задач классификации.
Без учителя	Нейронная сеть сама в процессе работы выделяет классы объектов и относит объект к определенному классу, такие сети используют для задач кластеризации.
<i><b>По типу сигнала</b></i>	
Бинарные	На вход нейронных сетей подают только нули или единицы.
Аналоговые	Подаваемые на входы нейронов сигналы могут быть произвольными (вещественными числами).
<i><b>По типу структур</b></i>	
Однородная	Все нейроны в нейронной сети используют одну функцию активации.
Неоднородная	Нейроны в нейронной сети имеют разные функции активации.

## Архитектуры нейронных сетей:

### 1. Многослойный перцептрон (рис.7)

Многослойный перцептрон состоит из 3 или более слоев. Он использует нелинейную функцию активации, часто тангенциальную или логистическую, которая позволяет классифицировать линейно неразделимые данные. Каждый узел в слое соединен с каждым узлом в последующем слое, что делает сеть полностью связанной. Такая архитектура находит применение в задачах распознавания речи и машинном переводе.

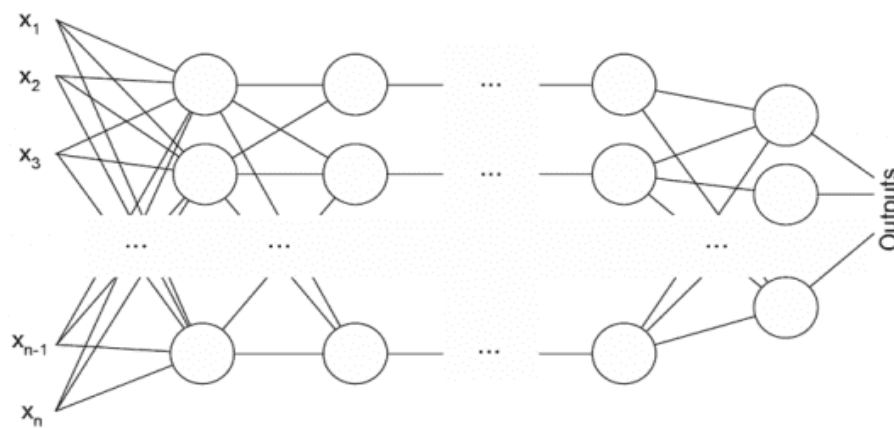


Рис.7. многослойный перцептрон.

## 2. Сверточная нейронная сеть (рис.8)

Сверточная нейронная сеть (Convolutional neural network, CNN) содержит один или более объединенных или соединенных сверточных слоев. CNN использует вариацию многослойного перцептрона, рассмотренного выше. Сверточные слои используют операцию свертки для входных данных и передают результат в следующий слой. Эта операция позволяет сети быть глубже с меньшим количеством параметров.

Сверточные сети показывают выдающиеся результаты в приложениях к картинкам и речи. В статье Convolutional Neural Networks for Sentence Classification автор описывает процесс задач классификации текста с помощью CNN. В работе представлена модель на основе word2vec, которая проводит эксперименты, тестируется на нескольких бенчмарках и демонстрирует блестящие результаты.

В работе Text Understanding from Scratch авторы показывают, что сверточная сеть достигает выдающихся результатов даже без знания слов, фраз, предложений и других синтаксических структур присущих человеческому языку. Семантический разбор, поиск парафраз, распознавание речи — тоже приложения CNN.

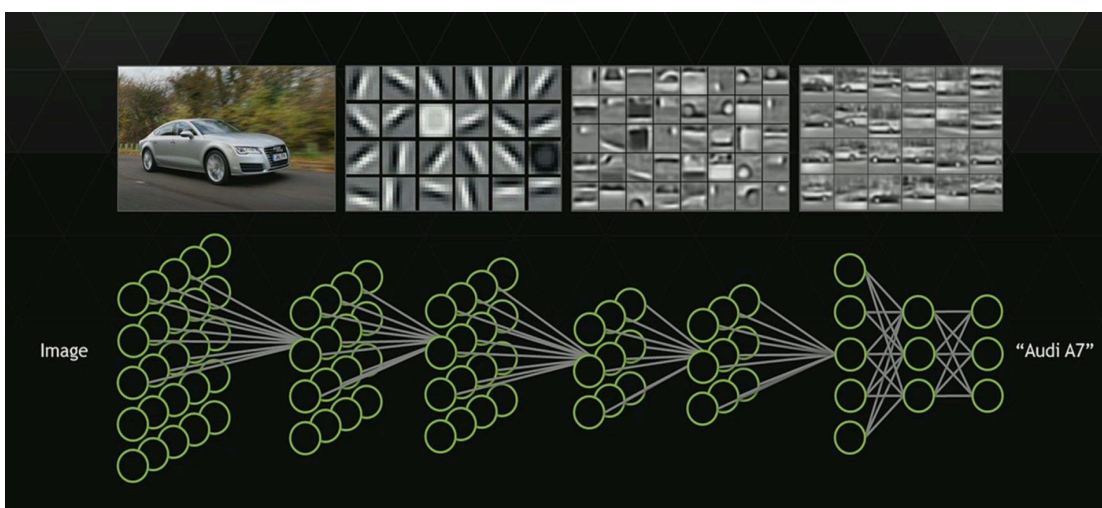


Рис.8. сверточные нейронные сети

### 3. Рекурсивная нейронная сеть (рис.9).

Рекурсивная нейронная сеть — тип глубокой нейронной сети, сформированный при применении одних и тех же наборов весов рекурсивно над структурой, чтобы сделать скалярное или структурированное предсказание над входной структурой переменного размера через активацию структуры в топологическом порядке. В простейшей архитектуре нелинейность, такая как тангенциальная функция активации, и матрица весов, разделяемая всей сетью, используются для объединения узлов в родительские объекты.

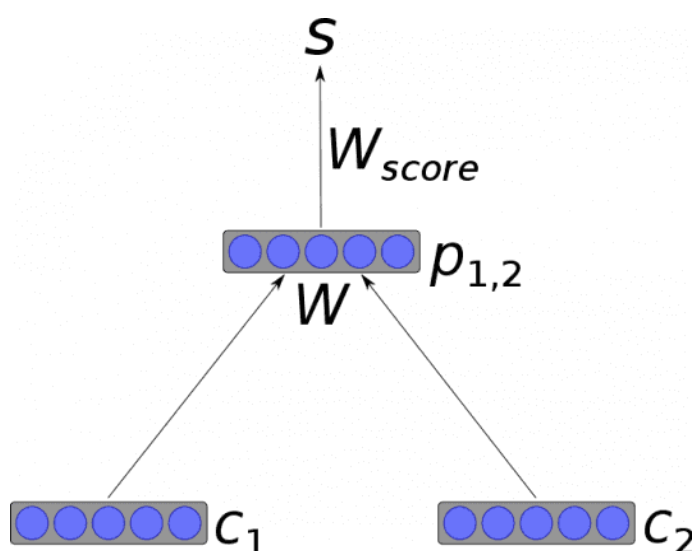


Рис.9. Рекурсивная нейронная сеть.

### 4. Рекуррентная нейронная сеть (рис.10).

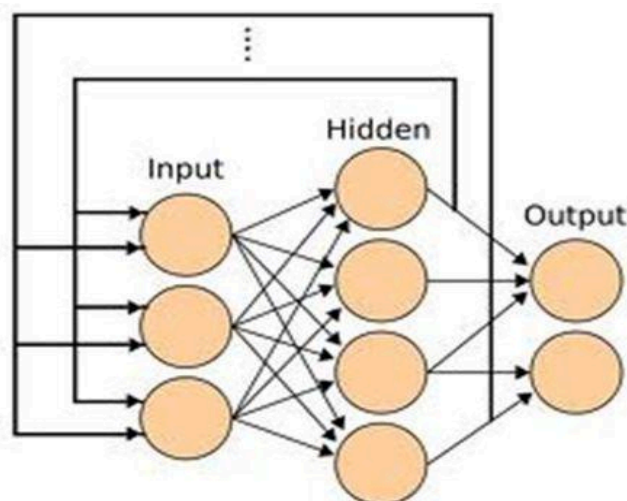


Рис.10. Рекуррентная нейронная сеть

Рекуррентная нейронная сеть, в отличие от прямой нейронной сети, является вариантом рекурсивной ИНС, в которой связи между нейронами — направленные циклы. Последнее означает, что выходная информация зависит не только от текущего входа, но также от состояний нейрона на предыдущем шаге. Такая память позволяет пользователям решать задачи NLP: распознавание рукописного текста или речи. Например, модель рекуррентной нейросети, которая генерирует новые предложения и краткое содержание текстового документа.

Siwei Lai, Liheng Xu, Kang Liu, и Jun Zhao в своей работе Recurrent Convolutional Neural Networks for Text Classification создали рекуррентную сверточную нейросеть для классификации текста без рукотворных признаков. Модель сравнивается с существующими методами классификации текста — Bag of Words, Bigrams + LR, SVM, LDA, Tree Kernels, рекурсивными и сверточными сетями. Описанная модель превосходит по качеству традиционные методы для всех используемых датасетов.

## 5. Сеть долгой краткосрочной памяти (рис.11).

Сеть долгой краткосрочной памяти (Long Short-Term Memory, LSTM) — разновидность архитектуры рекуррентной нейросети, созданная для более точного моделирования временных последовательностей и их долгосрочных зависимостей, чем традиционная рекуррентная сеть

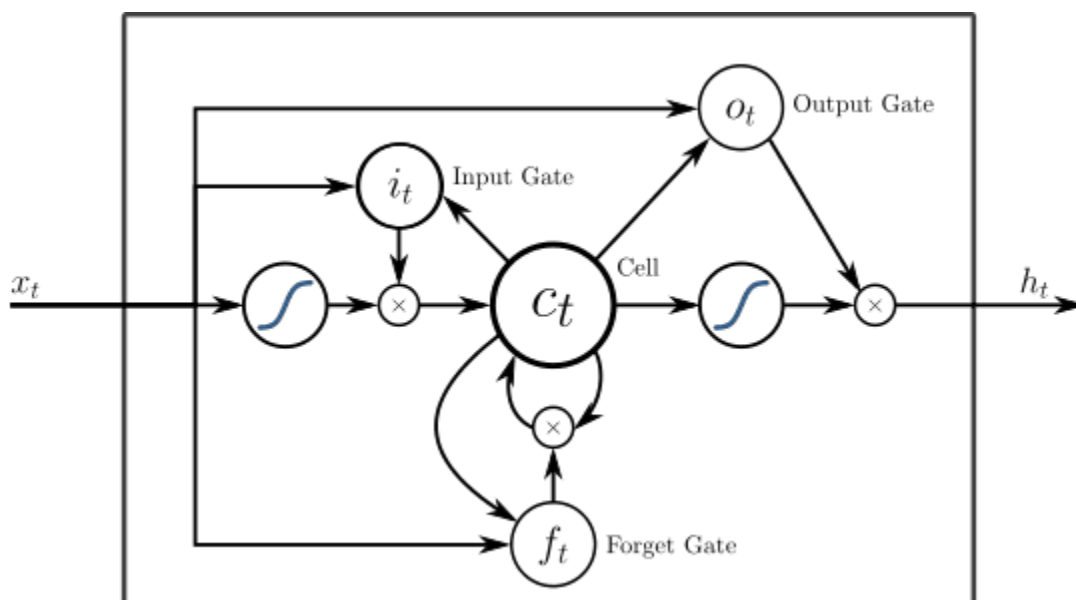


Рис.11. LSTM блок с входным, выходным и гейтом забывания.

LSTM-сеть не использует функцию активации в рекуррентных компонентах, сохраненные значения не модифицируются, а градиент не стремится исчезнуть во время тренировки. Часто LSTM применяется в блоках по несколько элементов. Эти блоки состоят из 3 или 4 затворов (например, входного, выходного и гейта забывания), которые контролируют построение информационного потока по логистической функции.

В Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling авторы показывают архитектуру глубокой LSTM рекуррентной сети, которая достигает хороших результатов для крупномасштабного акустического моделирования.

Apple, Amazon, Google, Microsoft и другие компании внедрили в продукты LSTM-сети как фундаментальный элемент.

## 6. Sequence-to-sequence модель

Часто Sequence-to-sequence модели состоят из двух рекуррентных сетей: кодировщика, который обрабатывает входные данные, и декодера, который осуществляет вывод.

Sequence-to-Sequence модели часто используются в вопросно-ответных системах, чат-ботах и машинном переводе. Такие многослойные ячейки успешно использовались в sequence-to-sequence

моделях для перевода в статье Sequence to Sequence Learning with Neural Networks study.

В Paraphrase Detection Using Recursive Autoencoder представлена новая рекурсивная архитектура автокодировщика, в которой представления — вектора в  $n$ -мерном семантическом пространстве, где фразы с похожими значением близки друг к другу.

## 7. Неглубокие (shallow) нейронные сети

Неглубокие модели, как и глубокие нейронные сети, тоже популярные и полезные инструменты. Например, word2vec — группа неглубоких двухслойных моделей, которая используется для создания векторных представлений слов (word embeddings). Представленная в Efficient Estimation of Word Representations in Vector Space, word2vec принимает на входе большой корпус текста и создает векторное пространство. Каждому слову в этом корпусе приписывается соответствующий вектор в этом пространстве. Отличительное свойство — слова из общих текстов в корпусе расположены близко друг к другу в векторном пространстве.

В статье описаны архитектуры нейронных сетей: глубокий многослойный перцептрон, сверточная, рекурсивная, рекуррентная сети, нейросети долгой краткосрочной памяти, sequence-to-sequence модели и неглубокие (shallow) сети, word2vec для векторных представлений слов. Кроме того, было показано, как функционируют эти сети, и как различные модели справляются с задачами обработки естественного языка. Также отмечено, что сверточные нейронные сети в основном используются для задач классификации текста, в то время как рекуррентные сети хорошо работают с воспроизведением естественного языка или машинным переводом. В следующих части серии будут описаны существующие инструменты и библиотеки для реализации описанных типов нейросетей.

## Платформы для глубокого обучения:

### 1. Torch.

Torch был разработан с использованием языка LUA и реализован на языке C. Реализация на языке Python называется PyTorch.

### 2. Keras.

Keras — это фреймворк Python для глубокого обучения. Его большим плюсом является возможность использовать код как на CPU, так и на GPU. Она была разработана инженером из Google Франсуа Шолле и представлена в марте 2015 года. Keras работает поверх TensorFlow, CNTK и Theano. Более того, Keras интегрирован в библиотеку TensorFlow, и устанавливается вместе с ней. Именно поэтому для обучения нейронной сети выберем связку TensorFlow + Keras.

### 3. TensorFlow

TensorFlow — это библиотека глубокого обучения с открытым исходным кодом, созданная Google. Как и большинство фреймворков глубокого обучения, TensorFlow имеет API на Python поверх механизма C и C++, что ускоряет его работу. Keras можно запускать поверх TensorFlow.

### 4. DL4J

Deep Learning for Java (DL4J) — первая библиотека глубокого обучения, написанная для Java и Scala. Она интегрирована с Hadoop и Apache Spark.



## Создание нейронной сети.

### Введение, область применения, постановка задачи:

Изначально рассматривались несколько вариантов проекта с разными областями применения, из которых надо было выбрать наиболее полезный и актуальный. И все же мой выбор остановился на медицине. В плане было создать классическую свёрточную или полносвязную нейронную сеть для классификации изображений с органом, пораженным болезнью или здоровым соответственно. Полезность применения такой нейросети колоссальна: она может частично взять на себя ответственность врача – пригодится в экстренных ситуациях, когда врачей катастрофически не хватает, а также может использоваться врачом в качестве повторной постановки диагноза для дополнительной проверки и исключения человеческого фактора. Еще один вариант применения – в обучающем процессе в мед. учреждениях. И хоть в этой работе нейросеть будет определять только наличия признаков для одной конкретной болезни, в силу гибкости технологии и при большем объёме данных, можно из отдельных нейросетей, обученных на определение наличия только одного заболевания путем их объединения создать более мощную нейросеть, способную определять наличие нескольких болезней.

Выделим основные этапы по созданию готовой нейросети:

1. Сбор данных и их упорядочивание.
2. Анализ и обработка данных под конкретную задачу.
3. Создание модели нейросети опытным путем.
4. Обучение модели на тренировочных данных.
5. Анализ процесса обучения. В случае его неудовлетворительности, переходим к пункту 6.

6.Улучшение модели, повторное обучение. Если результат неудовлетворительный, повторять пункты 5-6.

7.Визуализация процесса обучения.

8.Предсказывание результата на тестовых данных.

Помимо обучения нейронной сети, не менее важным этапом является сбор и структурирования данных в датасет, который занимает большую часть времени и усилий. Для большей концентрации на реализации проекта, было принято решение выбрать один из уже существующих датасетов наиболее подходящий. В чем мне помог ресурс Kaggle.

В качестве данных для обучения выберем датасет рентгеновских снимков, на части которых обнаружена пневмония, а на другой нет. После обучения мы сможем предсказать на тестовых данных наличие пневмонии, что в особенности полезно во время периодических волн COVID-19, вызывающего пневмонию.

Теперь, перейдем непосредственно к листингу программы, который разбит поэтапно с подробными пояснениями. Программа написана и обучена в Colab, но также весь необходимый процесс подготовки данных, обучения и тестирования нейросети можно сделать локально в любой IDE Python:

### 1.Подключаем необходимые библиотеки.

Импортируем все необходимые для глубокого обучения библиотеки, такие как numpy, pandas, tensorflow, keras, matplotlib для построения графиков, sklearn. Перед этим скачаем и разархивируем датасет:

Листинг 1:

Ввод:

```
import re
import os
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```
print(tf.__version__)
```

Вывод: 2.9.2

## 2. Изменим ускоритель на TPU

Изменим ускоритель на TPU вместо CPU или GPU, который лучше всего подходит для машинного обучения. Если коротко, TPU – это интегральная схема ускорителя искусственного интеллекта (ASIC), разработанная Google специально для машинного обучения нейронных сетей, в частности с использованием собственного программного обеспечения Google TensorFlow. Здесь же объявим константы, которые будем использовать в дальнейшем:

Листинг2:

Ввод:

```
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print('Device:', tpu.master())
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.experimental.TPUStrategy(tpu)
except:
    strategy = tf.distribute.get_strategy()
print('Number of replicas:', strategy.num_replicas_in_sync)

BATCH_SIZE = 16 * strategy.num_replicas_in_sync
IMAGE_SIZE = [150, 150]
EPOCHS = 25
```

Вывод: Number of replicas: 1

## 3. Загрузим данные.

Данные датасета поделены на тренировочные, валидационные и тестовые. Однако разрыв в количестве файлов между тренировочным и валидационным датасетом слишком сильный. Для нивелирования данного эффекта, объединим валидационные и тренировочные данные, после чего поделим их в пропорции 4 : 1, и сразу же проверим результат. Также обратим внимание на наличие имбаланса в данных между снимками здоровых легких и легких с пневмонией:

### ЛИСТИНГ 3:

Ввод:

```
filenames = tf.io.gfile.glob(str('chest_xray/train/*/*'))
filenames.extend(tf.io.gfile.glob(str('chest_xray/val/*/*')))
filenames2 = tf.io.gfile.glob(str('chest_xray/test/*/*'))

train_filenames, val_filenames = train_test_split(filenames, test_size=0.2)
test_filenames = filenames2

print("Training and validating images count: " + str(len(filenames)))
print("Testing images count: " + str(len(filenames2)))
print("Training images count: " + str(len(train_filenames)))
print("Validating images count: " + str(len(val_filenames)))
NormalCount = len([filename for filename in train_filenames if "NORMAL" in filename])
print("Normal images count in training set: " + str(NormalCount))

PneumoniaCount = len([filename for filename in train_filenames if "PNEUMONIA" in filename])
print("Pneumonia images count in training set: " + str(PneumoniaCount))
```

Вывод:

Training and validating images count: 5232

Testing images count: 624

Training images count: 4185

Validating images count: 1047

Normal images count in training set: 1107

Pneumonia images count in training set: 3078

## 4. Обработка данные.

Удостоверимся в наличии только двух меток для всех картинок. Сейчас наш набор данных представляет собой список имен файлов. Для сопоставления каждого имени файла с соответствующей парой (изображение, метка) воспользуемся следующими методами. Поскольку у нас есть только две метки, мы перепишем метку так, чтобы 1 или True указывало на пневмонию, а 0 или False указывало на нормальное состояние. Преобразуем путь в список компонентов пути в функции `get_label()`. Изображения имеют значения от 0 до 255.

Сверточная нейросеть лучше справляется с маленькими числами, поэтому уменьшим это значение в функции `decode_img()`, сначала конвертировав строку в 3Д тензор, и далее конвертировав тензор в действительное число от 0 до 1, после чего приведя изображение к нужному размеру (в нашем случае заданному константой и равному 150 на 150):

Листинг 4:

Ввод:

```
lables = np.array([str(tf.strings.split(item, os.path.sep)[-1].numpy())[2:-1]
                   for item in tf.io.gfile.glob(str("chest_xray/train/*"))])
lables2 = np.array([str(tf.strings.split(item, os.path.sep)[-1].numpy())[2:-1]
                    for item in tf.io.gfile.glob(str("chest_xray/val/*"))])
print(lables)
print(lables2)
```

```
def get_label(file_path):
    parts = tf.strings.split(file_path, os.path.sep)
    return parts[-2] == "PNEUMONIA"
```

```
def decode_img(img):
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.convert_image_dtype(img, tf.float32)
    return tf.image.resize(img, IMAGE_SIZE)
```

```
def process_path(file_path):
    label = get_label(file_path)
    # load the raw data from the file as a string
    img = tf.io.read_file(file_path)
    img = decode_img(img)
    return img, label
```

Вывод:

```
['NORMAL' 'PNEUMONIA']
```

```
['NORMAL' 'PNEUMONIA']
```

## **5. Визуализируем датасет.**

Сначала визуализируем форму пары (изображение/метка), в результате получив (150, 150, 3) и True/False соответственно. В качестве предварительной подготовки воспользуемся буферизованной предварительной выборкой, чтобы получать данные с диска, не блокируя ввод-вывод. Так как это небольшой набор данных, загрузим его только один раз, сохранив в памяти, проведем кеширование после предварительной обработки данных, которые не помещаются в памяти и воспользуемся функцией `prefetch`, который позволяет набору данных извлекать пакеты в фоновом режиме, пока модель тренируется. Здесь же загрузим данные для тестового датасета, и применим функцию `prepare_for_training()` к валидационному и тренировочному датасету.

Листинг 5.1:

Ввод:

```
for image, label in train_ds.take(3):
    print("Image shape: ", image.numpy().shape)
    print("Label: ", label.numpy())

def prepare_for_training(ds, cache=True, shuffle_buffer_size=1000):
    if cache:
        if isinstance(cache, str):
            ds = ds.cache(cache)
        else:
            ds = ds.cache()

    ds = ds.shuffle(buffer_size=shuffle_buffer_size)
    ds = ds.repeat()
    ds = ds.batch(BATCH_SIZE)
    ds = ds.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

    return ds

test_list_ds = tf.data.Dataset.list_files(str('chest_xray/test/*/*'))
test_ds = test_list_ds.map(process_path,
num_parallel_calls=tf.data.experimental.AUTOTUNE)
test_ds = test_ds.batch(BATCH_SIZE)

train_ds = prepare_for_training(train_ds)
val_ds = prepare_for_training(val_ds)
image_batch, label_batch = next(iter(train_ds))
```

Вывод:

Image shape: (150, 150, 3) Label: True

Image shape: (150, 150, 3) Label: True

Image shape: (150, 150, 3) Label: True

0 сек.

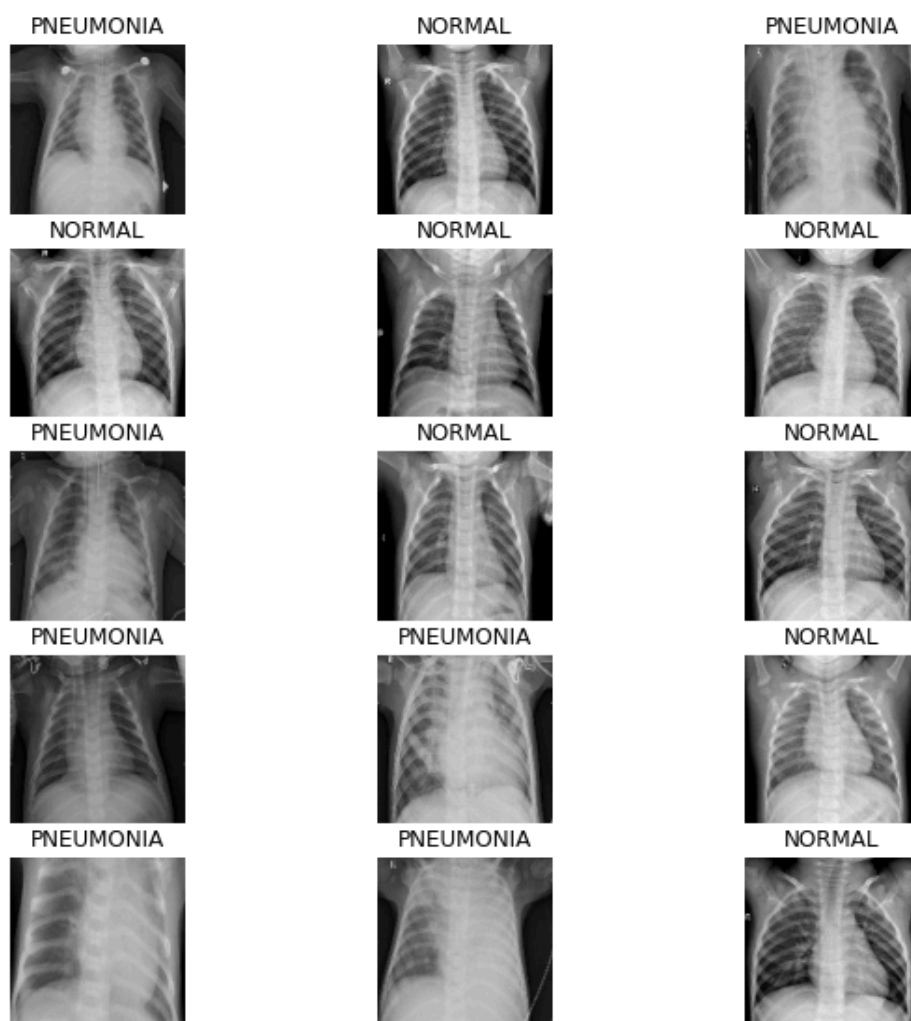
Выведем первые 15 снимков чтобы удостовериться, что все прошло успешно:

Листинг 5.2

Ввод:

```
def show_batch(image_batch, label_batch):  
    plt.figure(figsize=(10,10))  
    for n in range(25):  
        ax = plt.subplot(5,3,n+1)  
        plt.imshow(image_batch[n])  
        if label_batch[n]:  
            plt.title("PNEUMONIA")  
        else:  
            plt.title("NORMAL")  
        plt.axis("off")  
    show_batch(image_batch.numpy(), label_batch.numpy())
```

Вывод:



## 6. Построим и скомпилируем модель.

Архитектура модели содержит один Flatten-слой с входными параметрами изображения, а также несколько Dense-слоёв, содержащих нелинейные функции активации ReLU и один Dense-слой с нормализованной экспоненциальной функцией Softmax. Количество нейронов Dense-слоёв уменьшается с каждым слоем в несколько раз, а последний на выходе имеет два нейрона для классификации: или пневмония есть, или ее нет. Также немаловажной деталью является использование Dropout-слоёв между основными слоями, основная задача которых заключается в выборочном отключении некоторых нейронов в скрытых слоях для минимизации риска переобучения модели и дополнительной тренировки нейронов.

Листинг 6:

Ввод:

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(150, 150, 3)),
```



```

keras.layers.Dense(512, activation='relu'),
keras.layers.Dropout(0.3),
keras.layers.Dense(128, activation='relu'),
keras.layers.Dropout(0.2),
keras.layers.Dense(32, activation='relu'),
keras.layers.Dense(2, activation='softmax'),
])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics = ['accuracy'])

```

## 7. Обучаем модель.

Обучим нашу **полносвязную** нейронную сеть. Количество эпох возьмём равным 30, а количество проходов за эпоху – 261. На протяжении всего обучения будем следить за динамикой 4 показателей: loss – потери на тренировочной выборке, accuracy – точность на тренировочной выборке, val\_loss – потери на валидационной выборке, val\_accuracy – точность на валидационной выборке. Эти промежуточные результаты по каждой эпохе запишем в переменную history, для того чтобы на основе этих данных построить графики. Результаты обучения представлены ниже:

Листинг 7.1:

Ввод:

```

history = model.fit(
    train_ds,
    epochs = 30,
    steps_per_epoch = len(train_filenames) // BATCH_SIZE,
    validation_data=val_ds,
)

```

```
validation_steps= len(val_filenames) // BATCH_SIZE,
)
```

Вывод:

```
Epoch 16/30
261/261 [=====] - 47s 181ms/step - loss: 0.2925 - accuracy: 0.8705 - val_loss: 0.1913 - val_accuracy: 0.9396
Epoch 17/30
261/261 [=====] - 48s 182ms/step - loss: 0.2614 - accuracy: 0.8875 - val_loss: 0.1584 - val_accuracy: 0.9521
Epoch 18/30
261/261 [=====] - 48s 182ms/step - loss: 0.2700 - accuracy: 0.8748 - val_loss: 0.1696 - val_accuracy: 0.9458
Epoch 19/30
261/261 [=====] - 48s 182ms/step - loss: 0.2995 - accuracy: 0.8647 - val_loss: 0.1954 - val_accuracy: 0.9500
Epoch 20/30
261/261 [=====] - 48s 186ms/step - loss: 0.2683 - accuracy: 0.8906 - val_loss: 0.1573 - val_accuracy: 0.9500
Epoch 21/30
261/261 [=====] - 47s 180ms/step - loss: 0.2782 - accuracy: 0.8764 - val_loss: 0.1592 - val_accuracy: 0.9542
Epoch 22/30
261/261 [=====] - 49s 186ms/step - loss: 0.3012 - accuracy: 0.8657 - val_loss: 0.1982 - val_accuracy: 0.9500
Epoch 23/30
261/261 [=====] - 47s 181ms/step - loss: 0.2604 - accuracy: 0.8877 - val_loss: 0.1692 - val_accuracy: 0.9563
Epoch 24/30
261/261 [=====] - 47s 181ms/step - loss: 0.2616 - accuracy: 0.8908 - val_loss: 0.1638 - val_accuracy: 0.9438
Epoch 25/30
261/261 [=====] - 47s 181ms/step - loss: 0.2705 - accuracy: 0.8870 - val_loss: 0.1882 - val_accuracy: 0.9375
Epoch 26/30
261/261 [=====] - 47s 181ms/step - loss: 0.2706 - accuracy: 0.8831 - val_loss: 0.1744 - val_accuracy: 0.9438
Epoch 27/30
261/261 [=====] - 47s 181ms/step - loss: 0.2425 - accuracy: 0.8997 - val_loss: 0.1459 - val_accuracy: 0.9667
Epoch 28/30
261/261 [=====] - 49s 186ms/step - loss: 0.2783 - accuracy: 0.8784 - val_loss: 0.2525 - val_accuracy: 0.9083
Epoch 29/30
261/261 [=====] - 47s 181ms/step - loss: 0.2643 - accuracy: 0.8918 - val_loss: 0.2165 - val_accuracy: 0.9604
Epoch 30/30
261/261 [=====] - 48s 182ms/step - loss: 0.2555 - accuracy: 0.8968 - val_loss: 0.1568 - val_accuracy: 0.9708
```

Результаты совсем неудовлетворительные: если изучить глубже, становится понятно, что половину эпох нейросеть почти не улучшала результаты. Потери на тренировочных данных выше, чем на валидационных. Но и там, и там они весьма внушительные: 0.25 и 0.15 соответственно. Точность тоже не дотягивает: 0.89 и 0.97 для тренировочных и валидационных данных соответственно. Также стоит обратить внимание на высокую скорость обучения полносвязной нейросети: в среднем 47 секунд на эпоху. Дополним сложившуюся картину графической визуализацией, в чем нам поможет matplotlib:

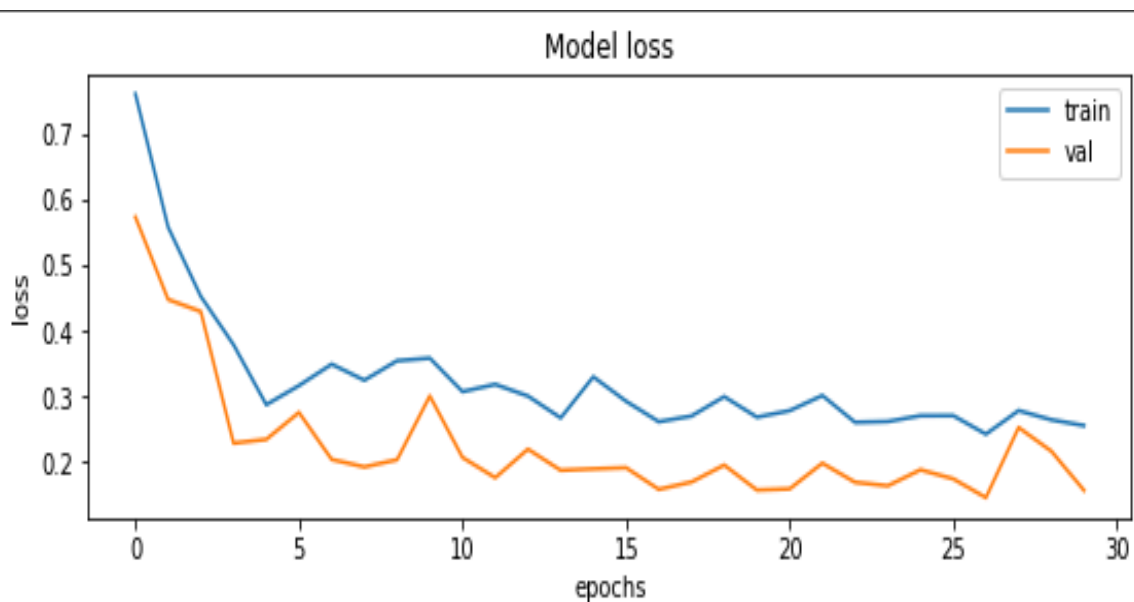
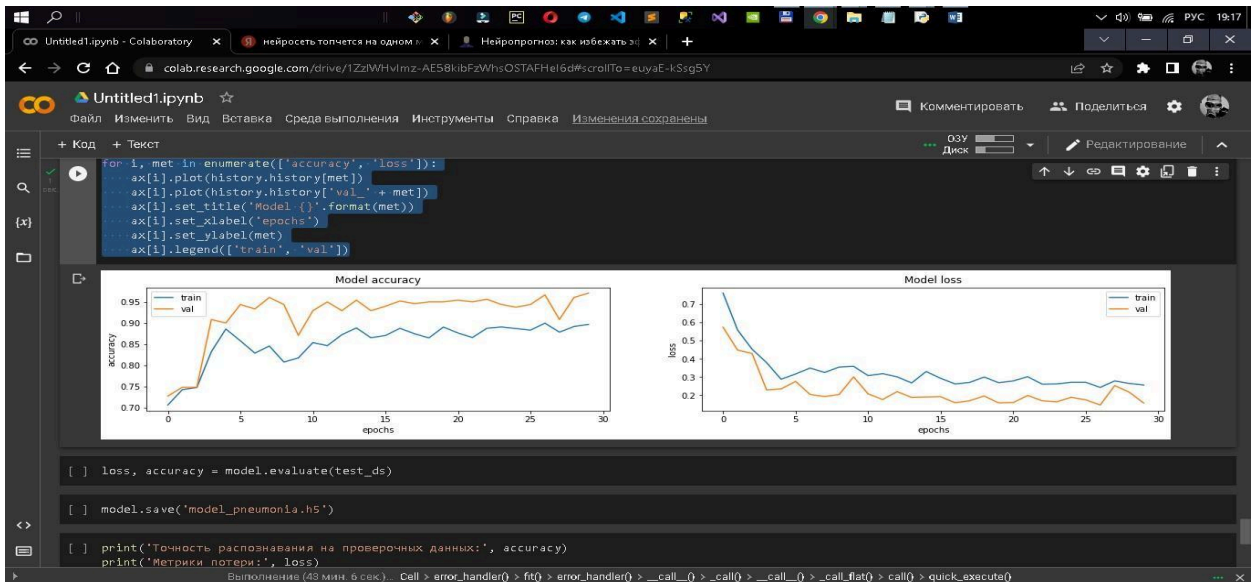
Листинг 7.2:

Ввод:

```
fig, ax = plt.subplots(1, 2, figsize= (20, 3))
ax = ax.ravel()
for i, met in enumerate(['accuracy', 'loss']):
    ax[i].plot(history.history[met])
    ax[i].plot(history.history['val_' + met])
    ax[i].set_title('Model {}'.format(met))
```

```
ax[i].set_xlabel('epochs')
ax[i].set_ylabel(met)
ax[i].legend(['train', 'val'])
```

Вывод:



## 8.Теперь проверим точность распознавания на тестовых данных:

Листинг 8:

Ввод:

```
loss, accuracy = model.evaluate(test_ds)
print('Точность распознавания на проверочных данных:', accuracy)
print('Метрики потери:', loss)
```

Вывод: 0.73

Точность на тестовых данных, как ожидалось, упала ещё больше.

Поэтому сейчас мы переделаем нашу модель и дополним полносвязную нейросеть сверточной.

## 9.Улучшим и переобучим нашу модель:

Сверточная часть: Добавим Keras Conv2D - слой двумерной свертки, создающий ядро свертки. Задаем количество ядер – 32 и размер ядра – 3 на 3 пикселя – столько пикселей обрабатываются за один шаг и подаются в один нейрон скрытого слоя нейросети. Параметр padding='same' для того, чтобы расположить ядро маски над первым пикселем снимка. Следующий слой - MaxPool2D - нужен для укрупнения полученных признаков. В нем задаем размер окна, в котором будет выбираться максимальное значение, параметр strides=2 задает шаг сканирования по осям плоскости. Далее по аналогии добавляем аналогичных два слоя, увеличив количество ядер вдвое. После первой операции Conv2D размер картинки уменьшается в 2 раза: до 75 x 75 пикселей. После логичной операции – еще в два раза, до размера 32.5 x 32.5. Поэтому на выходе четвертого слоя имеем тензор размерностью (32.5, 32.5, 64).

Далее подаем наш тензор на полносвязную нейронную сеть для конечной классификации. Чтобы вытянуть тензор в один вектор используем слой Flatten, чтобы потом подать его на полносвязную сеть.

Теперь подадим на три слоя Dense с 512, 128 и 32 нейронами соответственно. Функция активации по-прежнему ReLU. Между Dense-слоями обязательно добавим Dropout со значениями 0.5, 0.3 и 0.2. Конечный выходной слой состоит из 2 нейронов и функции активации 'softmax'.

Листинг 9.1:

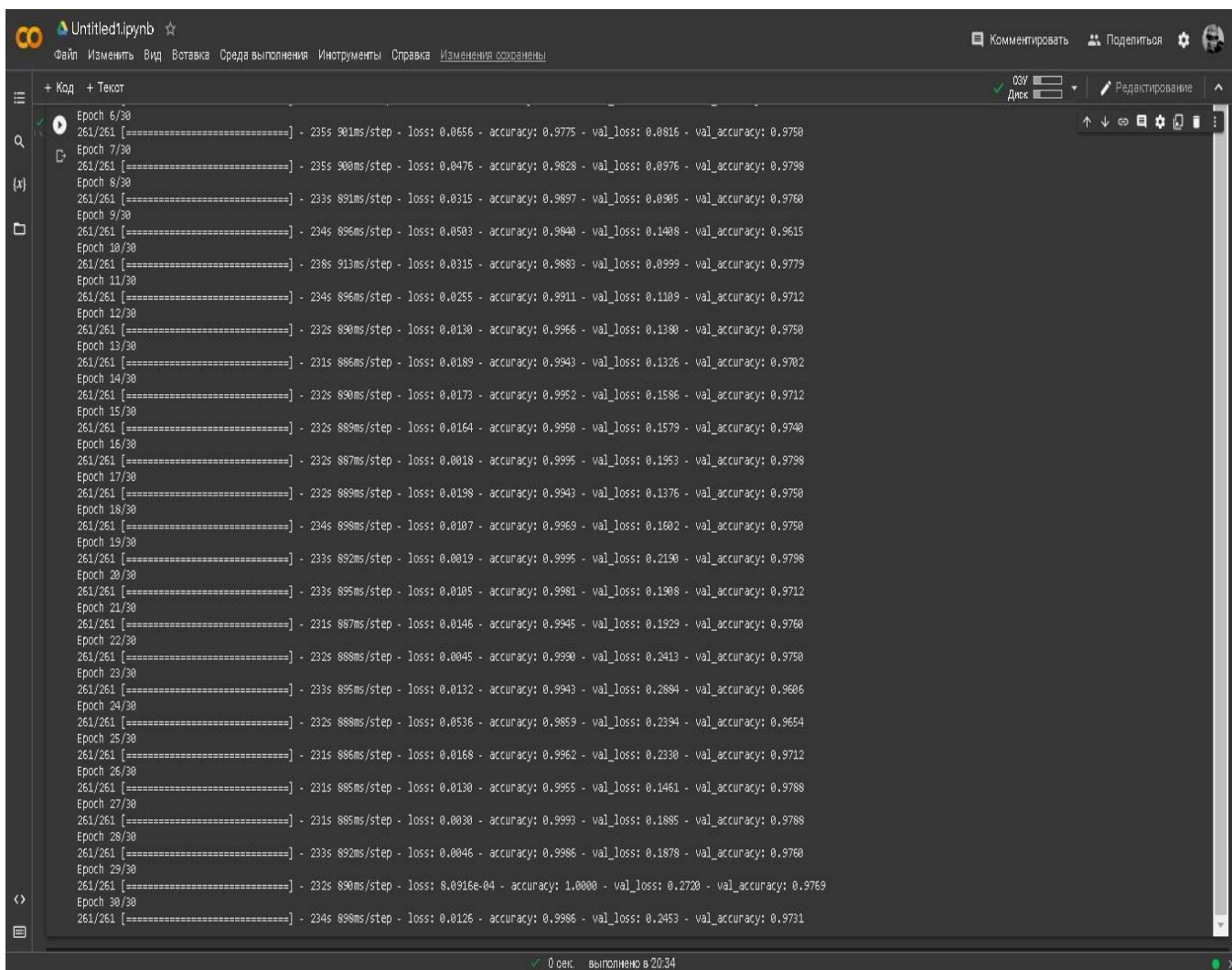
Ввод:

```
model = keras.Sequential([
    keras.layers.Conv2D(32, (3,3), activation='relu', padding='same',
        input_shape=(150,150,3)),
    keras.layers.MaxPool2D((2,2), strides=2),
    keras.layers.Conv2D(64, (3,3), activation='relu', padding='same'),
    keras.layers.MaxPool2D((2,2), strides=2),
    keras.layers.Flatten(),

    keras.layers.Dense(512, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.3),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(2, activation='softmax')
```

)

Вывод:



```
Epoch 6/30
261/261 [=====] - 235s 901ms/step - loss: 0.0656 - accuracy: 0.9775 - val_loss: 0.0816 - val_accuracy: 0.9750
Epoch 7/30
261/261 [=====] - 235s 900ms/step - loss: 0.0476 - accuracy: 0.9828 - val_loss: 0.0976 - val_accuracy: 0.9798
Epoch 8/30
261/261 [=====] - 233s 891ms/step - loss: 0.0315 - accuracy: 0.9897 - val_loss: 0.0905 - val_accuracy: 0.9760
Epoch 9/30
261/261 [=====] - 234s 896ms/step - loss: 0.0503 - accuracy: 0.9840 - val_loss: 0.1408 - val_accuracy: 0.9615
Epoch 10/30
261/261 [=====] - 238s 913ms/step - loss: 0.0315 - accuracy: 0.9683 - val_loss: 0.0999 - val_accuracy: 0.9779
Epoch 11/30
261/261 [=====] - 234s 896ms/step - loss: 0.0255 - accuracy: 0.9911 - val_loss: 0.1109 - val_accuracy: 0.9712
Epoch 12/30
261/261 [=====] - 232s 890ms/step - loss: 0.0130 - accuracy: 0.9966 - val_loss: 0.1380 - val_accuracy: 0.9750
Epoch 13/30
261/261 [=====] - 231s 885ms/step - loss: 0.0189 - accuracy: 0.9943 - val_loss: 0.1326 - val_accuracy: 0.9702
Epoch 14/30
261/261 [=====] - 232s 890ms/step - loss: 0.0173 - accuracy: 0.9952 - val_loss: 0.1586 - val_accuracy: 0.9712
Epoch 15/30
261/261 [=====] - 232s 889ms/step - loss: 0.0164 - accuracy: 0.9950 - val_loss: 0.1579 - val_accuracy: 0.9740
Epoch 16/30
261/261 [=====] - 232s 887ms/step - loss: 0.0018 - accuracy: 0.9995 - val_loss: 0.1953 - val_accuracy: 0.9798
Epoch 17/30
261/261 [=====] - 232s 889ms/step - loss: 0.0198 - accuracy: 0.9943 - val_loss: 0.1376 - val_accuracy: 0.9750
Epoch 18/30
261/261 [=====] - 234s 898ms/step - loss: 0.0107 - accuracy: 0.9969 - val_loss: 0.1602 - val_accuracy: 0.9750
Epoch 19/30
261/261 [=====] - 233s 892ms/step - loss: 0.0019 - accuracy: 0.9995 - val_loss: 0.2190 - val_accuracy: 0.9798
Epoch 20/30
261/261 [=====] - 233s 895ms/step - loss: 0.0105 - accuracy: 0.9981 - val_loss: 0.1908 - val_accuracy: 0.9712
Epoch 21/30
261/261 [=====] - 231s 887ms/step - loss: 0.0146 - accuracy: 0.9945 - val_loss: 0.1929 - val_accuracy: 0.9760
Epoch 22/30
261/261 [=====] - 232s 888ms/step - loss: 0.0045 - accuracy: 0.9990 - val_loss: 0.2413 - val_accuracy: 0.9750
Epoch 23/30
261/261 [=====] - 233s 895ms/step - loss: 0.0132 - accuracy: 0.9943 - val_loss: 0.2894 - val_accuracy: 0.9606
Epoch 24/30
261/261 [=====] - 232s 888ms/step - loss: 0.0536 - accuracy: 0.9859 - val_loss: 0.2394 - val_accuracy: 0.9654
Epoch 25/30
261/261 [=====] - 231s 886ms/step - loss: 0.0168 - accuracy: 0.9962 - val_loss: 0.2330 - val_accuracy: 0.9712
Epoch 26/30
261/261 [=====] - 231s 885ms/step - loss: 0.0130 - accuracy: 0.9955 - val_loss: 0.1461 - val_accuracy: 0.9788
Epoch 27/30
261/261 [=====] - 231s 885ms/step - loss: 0.0030 - accuracy: 0.9993 - val_loss: 0.1885 - val_accuracy: 0.9788
Epoch 28/30
261/261 [=====] - 233s 892ms/step - loss: 0.0046 - accuracy: 0.9986 - val_loss: 0.1878 - val_accuracy: 0.9760
Epoch 29/30
261/261 [=====] - 232s 890ms/step - loss: 8.0916e-04 - accuracy: 1.0000 - val_loss: 0.2720 - val_accuracy: 0.9769
Epoch 30/30
261/261 [=====] - 234s 898ms/step - loss: 0.0126 - accuracy: 0.9986 - val_loss: 0.2453 - val_accuracy: 0.9731
```

Разница между предыдущими и текущими результатами обучения видна невооруженным взглядом. Как и время и среднее время обучения. В отличие от чисто полносвязной нейросети, сверточная нейросеть с каждой эпохой улучшала свои результаты. Хотя и время обучения ощутимо выросло, полученная точность полностью компенсирует этот недостаток. Ведь каждая сотая процента ошибки и точности – это чья-то спасенная жизнь.

Итак, перейдем к результатам. Примерно после половины пройденных эпох модель стала преобучаться (видно на валидационных данных), даже дропауты здесь не помогли. Поэтому лучше остановится после 15 эпохи. Но тренировочные данные уверенно только улучшали показатели. На последней 30-ой эпохе потери на тренировочных данных 0.0126, на валидационных 0.24 при лучшем результате 0.09, точность на тренировочных данных – 0.9986, на валидационных – 0.9731. Даже такие результаты можно назвать приемлемыми.

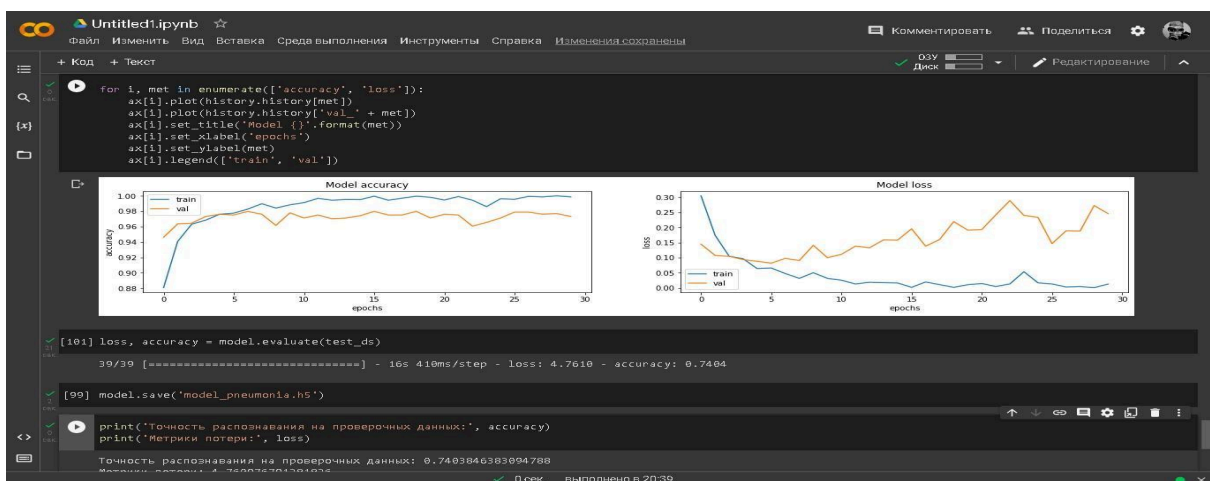
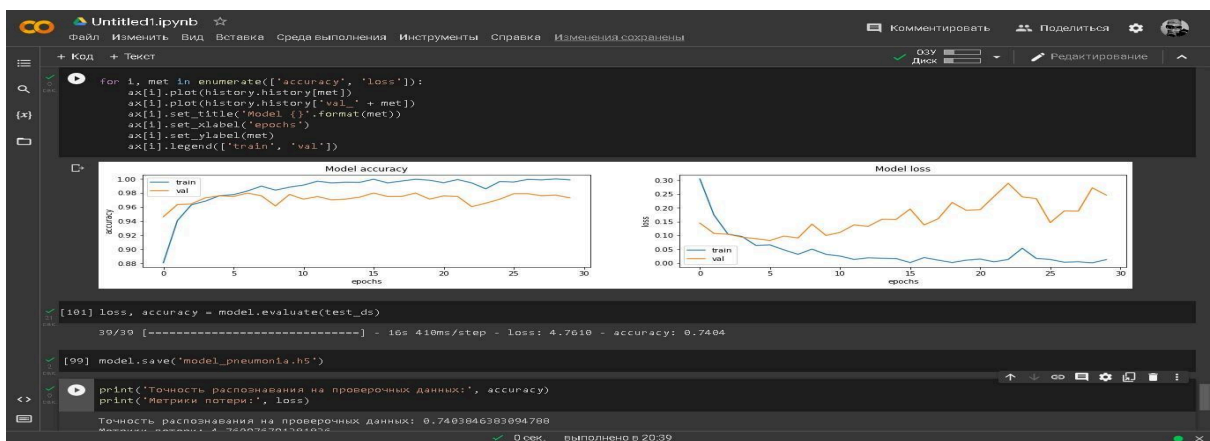
Дополним сложившуюся картину графической визуализацией, в чем нам поможет matplotlib:

## Листинг 9.2:

Ввод:

```
fig, ax = plt.subplots(1, 2, figsize= (20, 3))
ax = ax.ravel()
for i, met in enumerate(['accuracy', 'loss']):
    ax[i].plot(history.history[met])
    ax[i].plot(history.history['val_' + met])
    ax[i].set_title('Model {}'.format(met))
    ax[i].set_xlabel('epochs')
    ax[i].set_ylabel(met)
    ax[i].legend(['train', 'val'])
```

Вывод:



## 10. Теперь проверим точность распознавания на тестовых данных:

Листинг 10:

Ввод:

```
loss, accuracy = model.evaluate(test_ds)
print('Точность распознавания на проверочных данных:', accuracy)
print('Метрики потери:', loss)
```

Вывод: 0.74

Итак, конечная точность на тестовой выборке – 0.74. Безусловно, это не самый лучший результат и при долгом экспериментировании с параметрами обучения, количеством и типами слоёв есть большая вероятность показать результат выше 0.9 на тестовой выборке. Мои еще несколько попыток не дали лучшего результата на тестовой выборке. Возможные причины низкой точности: недостаточное количество слоев, слишком большое количество нейронов, недостаточное воздействие дропаута, слишком большое количество эпох, несовершенство построенной модели.



## **Вывод:**

Таким образом, при работе над курсовой по теме «Машинное обучение с помощью библиотеки TensorFlow» удалось провести анализ существующих архитектур нейронных сетей, выбрать наиболее подходящую: вначале полносвязную, после сверточную, провести анализ существующих алгоритмов классификации и распознавания, настроить ПО для обучения и тестирования нейросети, обучить и протестировать нейросеть, интерпретировать результаты проведенного тестирования, изучить возможные проблемы и их решения.

После знакомства с фреймворками TensorFlow и Keras, хочу выделить такие их преимущества как относительная простота в использовании (функции заменяют десятки строк кода, которые пришлось бы использовать, в случае, если бы мы захотели повторить аналогичную модель на языке Python или C#), простота в освоении и также скорость исполнения (код TensorFlow написан на C++, который существенно быстрее Python). К недостаткам могу отнести сложность настраивания окружения и сложность достижения совместимости версий всех библиотек. Но этот недостаток легко нивелируется возможностью создания модели нейронной сети в таких блокнотах как Kaggle, Colab.

TensorFlow в связке с Keras сейчас, несомненно, одно из лучших предложений на рынке, что обусловлено наличием всех его достоинств, описанных выше.

## **Литература:**

Гаврилов. Системы ИИ.

Франсуа Шолле – Глубокое обучение на Python.

Барский А. Б. - Нейронные сети - распознавание, управление, принятие решений.

С. Николенко, А. Кадулин, Екатерина Архангельская – Глубокое обучение. Погружение в мир нейронных сетей.

Сидоркина Системы искусственного интеллекта.

Тематические статьи на Хабр.

Ютуб-канал Selfedu, плейлист, посвященный нейросетям.