МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Реализация и анализ алгоритмов для работы с медицинскими данными

Студент С.Р. Хакимов
(И.О. Фамилия) (личная подпись)

Руководитель канд. пед. наук, доцент, О. М. Гущина
(ученая степень, звание, И.О. Фамилия)

Консультант М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

## Аннотация

Темой бакалаврской работы является «Реализация и анализ алгоритмов для работы с медицинскими данными».

Объектом работы является процесс функционирования нейронных сетей. Предметом выпускной квалификационной работы является алгоритм распознавания COVID-19 на изображении.

Выпускная квалификационная работа состоит из введения, трех глав, заключения и списка литературы.

В ведении описывается актуальность данной работы.

В первой главе будет рассмотрена болезнь COVID-19, так же теоретические сведения об отдельных видах искусственных нейронных сетей.

Во второй главе описывается про выбор данных, их нормализацию, после этого теоретическая схема работы алгоритма, а также её модернизация, и под конец программная реализация системы.

В третьей главе проводиться тестирование методом кросс-валидации. Так де приводиться график к данному тестированию.

В заключении вынесены выводы о проделанной работе.

Данная выпускная квалификационная работа содержит в себе пояснительную записку, состоящую из 43 страниц, 18 рисунков,1 графика, 9 формул и списка литературы из 23 источников.

## Abstract

The topic of the bachelor's thesis is "Implementation and analysis of algorithms for working with medical data".

The object of the work is the process of functioning of neural networks. The subject of the graduate qualification work is an algorithm for COVID-19 recognition on the image.

Graduate qualification work consists of an introduction, three chapters, a conclusion and a list of references.

The introduction describes the relevance of this work.

The first chapter will be considered disease COVID-19, as well as theoretical information about individual types of artificial neural networks.

The second chapter describes the selection of data, their normalization, then the theoretical scheme of the algorithm, as well as its modernization, and at the end of the software implementation of the system.

In the third chapter the cross-validation testing is carried out. So, the graph for this testing is given.

In the conclusion of the conclusions about the work done.

This graduate qualification work contains an explanatory note, consisting of 43 pages, 18 pictures, 1 graphic, 9 formulas and a reference list of 23 sources.

## Оглавление

# Введение

В современном мире коронавирус, семейство вирусов, вызывающее как следствие в начале обычную простуду, а заканчивающееся такими тяжкими заболеваниями, как тяжёлый острый респираторный синдром и ближневосточный респираторный синдром. От этого все чаще встречается необходимость быстрого анализа рентгеновского снимка на наличие заболевания. Данная задача значительно облегчает жизнь человека при автоматизации с помощью алгоритма.

Реализовать алгоритм распознавания болезни возможно с помощью искусственных нейронных сетей [0]. Нейронная сеть – программное или аппаратное воплощение, построенная по принципу организма и функционирования биологических нейронных сетей, а также математическая модель. По итогу, в конце получаем продукт, имитирующий человеческое мышление.

Актуальность данной работы состоит в том в разработке более точного алгоритма распознавания болезни, который исправит несовершенства ряд, существующий моделей, а также сократит затраченное время вместе с используемыми ресурсами.

Новизна бакалаврской работы состоит в разработке более точного алгоритма распознавания COVID-19.

Практическая ценность бакалаврской работы заключается в совершенствовании алгоритма распознавания болезни путем уменьшения затрачиваемых ресурсов. Возможность автоматизировать огромное количество умственного труда в обнаружении и в следствии выставления диагноза процесса, несет огромную практическую роль.

Предметом исследования является точность алгоритма.

Цель исследования заключается в разработке более точного алгоритма распознавания заболевания на изображении.

Для достижения цели поставлены следующие задачи:

1. Проанализировать существующие нейронные сети.
2. Предобработать входные данные для подачи их в модель.
3. Выбрать нейронную сеть для создания модели и изучить её строение.
4. Выбрать Фреймворк для работы с нейронной сетью.
5. Реализовать выбранную нейронную сеть.
6. Обучить выбранную нейронную сеть.
7. Протестировать алгоритм на примерах реальных изображений.

Бакалаврская работа состоит из введения, трех глав, заключения и списка используемой литературы.

В первой главе рассматривается биологические понятия нейрона, сама болезнь COVID-19, принципы организации искусственных нейронных сетей, виды Фреймворков для работы с искусственными нейронными сетями, поговорим про их плюсы и минусы.

Во второй главе рассмотрены общие принципы организации алгоритмов распознавания COVID-19 по изображению, осуществлён и обоснован выбор средств для реализации алгоритма, а также непосредственно описана его разработка.

В третьей главе осуществлено тестирование данными, итоговое подведение эффективности, скорости, а также продуктивности работы алгоритма.

В заключении формируются и описываются выводы, а также подводятся итоговые результаты исследования.

# 1 Разновидности нейронных сетей и их виды

## 1.1 Описание болезни covid-19

Первым, что требуется сделать, до того, как начнем писать алгоритм, разберем что из себя представляет болезнь, как она отражается на рентгеновских снимках, какие симптомы ей присуще.

В 2019 году весь мир столкнулся с чрезвычайной ситуацией в области здравоохранения в связи с появлением нового коронавируса (КОВИД-19). Почти 196 стран были затронуты covid-19, в то время как США, Италия, Китай, Испания, Иран и Франция имеют максимальную активность случаев заболевания.

Коронавирус — это острое респираторное заболевание дыхательных путей, ассоциированное с вирусом SARS-CoV-2 [2]. Данная болезнь, как и болезнь «пневмония», поражает большое количество участка легких. Тем самым в работе будем рассматривать и данный вид болезни, так как их часто путали во время диагностики.

К сожалению, данный вид томографии, хоть дает нам большой процент, того, что мы выявим состояние пациента на то, болен он или нет данной болезнью, но на данный момент мы имеем малый процент людей, которые воспользовались столь ярым примером. Это связанно из-за недостатка аппаратуры.

При пневмонии, связанной с COVID-19, «матовое стекло» [3], изображение которого можно увидеть на рисунке 1 область находится ближе к обоим легким, то есть к нижним и боковым областям, перибронхиальной области или плевре. Компьютерная томография может определить степень поражения легких коронавирусом. Менее трех «матовых стаканов» соответствуют легкому заболеванию, а более трех - среднему повреждению легких.

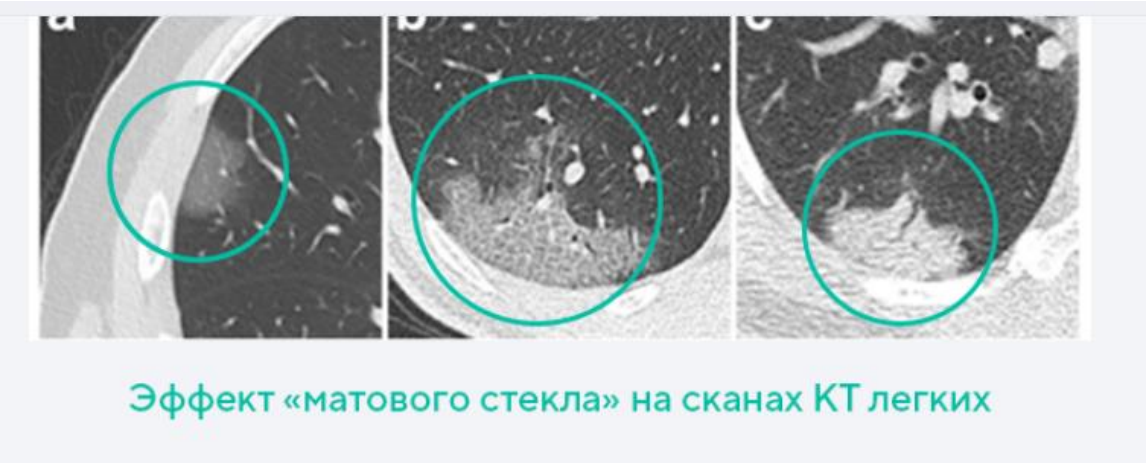


Рисунок 1 – Поражение легких COVID-19

Чтобы оценить степень поражения в процентах, легкие делятся на пять долей (три справа и две слева). Радиолог исследует каждую долю и оценивает степень повреждения каждой доли по 5-балльной шкале. Здесь 1 балл соответствует 5% или менее препятствий и 5 баллов (75% или более). Также сложите все баллы и умножьте на 4. Полученное число представляет степень поражения легких коронавирусом в процентах. Если дыхательная система функционирует ниже 50%, это уже основа национального питания.

Врачи видят клинически значимые признаки пневмонии в дополнение кроме «матовых стекло» на компьютерной томографии легких пациента с коронавирусом и в других особенностях, таких, как:

Синдром «брусчатки» или «лоскутного шитья» [4] - если уплотнение распространяется и на перегородку между долями легкого (примерно на 3-й день пневмонии), текстура легочной ткани на компьютерной томографии напоминает брусчатку.

Матовое стекло – это участки уплотнения легочной ткани, которые в свою очередь не пропускают лучи x-ray. По мере прогрессирования заболевания (обычно через 5-8 дней) легочная ткань уплотняется и пропускает больше рентгеновских лучей, при этом в газообмен вовлечено меньше функциональных областей. Так же их называют очагами уплотнения легочной ткани. Если данная общая площадь будет увеличивать, то это свидетельствует,

что инфекция прогрессирует.

Симптомом обратного ореола или синдрома ободка является кольцевидная область («матовое стекло») вокруг места инфекции. Встречается более чем у 50% больных коронавирусом.

Симптом воздушной бронхографии - наличие воздуха в просвете бронхов с выраженной интеграцией «матовых стекол».

Однако недавние исследования показали, что, хоть КТ хорошо выявляет изменения в легких, она не диагностирует инфекции, так же причина недостатка аппаратов в больницах. В результате был выбран рентген легких, изображение которого вместе с выделением болезни показан на рисунке 2.

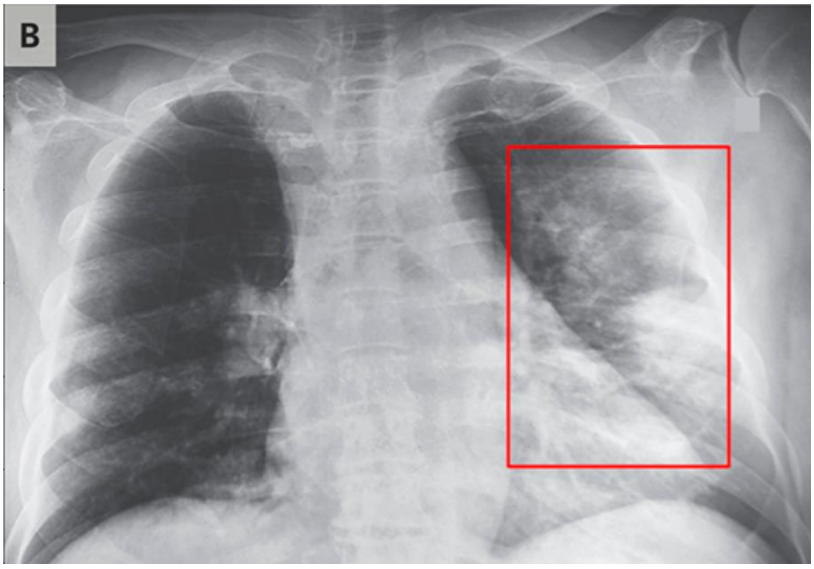


Рисунок 2 – Поражение легких COVID-19

COVID-19 атакует эпителий дыхательного тракта [6], а рентгеновские лучи помогают изучить состояние легких пациента. Из-за того, что аппараты для получение рентгеновского снимка расположены во всех больницах, логичным шагом будет использовать флюорографию для тестов.

По итогу, можно сказать, что данное заболевание достаточно быстро развивается, а на данный момент борьба с ней идет не в лучшую сторону, а имеющие алгоритмы работают достаточно медленно, тем самым было решено

улучшить один из нынешних алгоритмов, убрав все минусы и ускорив работу в несколько раза.

### 1.2 Виды искусственных нейронных сетей и их применение

Изучив болезнь и поняв на что, стоит обращать внимание можно приступить к следующему шагу. Но для того, чтобы мы смогли распознать ту или иную болезнь нам потребуется рассмотреть популярные архитектуры нейронных сетей и среди них выбрать ту, что подходит для нашей задачи, а именно распознать на изображении ту или иную особенность связанную с COVID-19.

1. Многослойный перцептрон.

Данный вид нейронной сети состоит от 3 и больше слоев [7]. То есть каждый слой нейронной сети состоит из элементов. Данные нейроны бывают трех типов: входные (сенсорные), обучаемые (ассоциативные) и входные (реагирующие) [8]. Название было получено в следствии того, что нейрон содержит несколько обучаемых слове. В теории не модель получает сигнал, помножает сигналы на веса и суммирует получившиеся величины, после чего передает результат на выход сети или к другому нейрону

Задачи, которые она решает относятся к бизнес-аналитике, то есть к анализу данных;

2. Рекурсивная нейронная сеть

Данный вид работает с данными переменной длины, при этом использует иерархические структуры образцов во время обучения [10]. В рекурсивных сетях нейроны с одинаковыми весами, рекурсивно активизируются в соответствии с архитектурой сети. Применяется в деревьях и последовательных структурах при обучении в задачах по обработки естественного языка. Архитектура данной сети продемонстрирована на рисунке 3.

Решает задачи: обработка текста на естественном языке, обработка

аудио, обработка видео, обработка изображения.

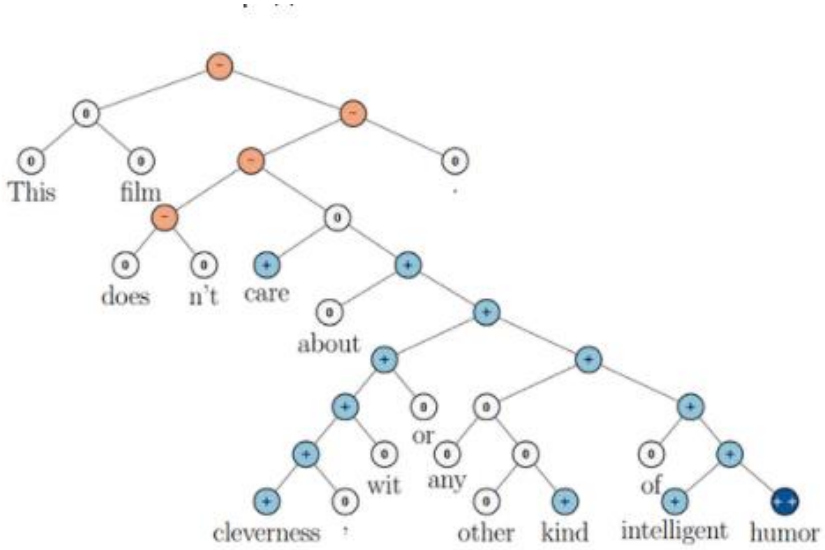


Рисунок 3 – Рекурсивная нейронная сеть

3. Рекуррентная нейронная сеть

Оно представляет из себя класс которых отлично подходят для моделирования последовательных данных, таких как естественные языки или же временные ряды. Плюсом от многослойного перцептрона будет, то, что рекуррентная сеть может использовать свою внутреннюю память для обработки задач, то есть последовательностей произвольной длины. Проблемой появляется при работе с большими данными, по мере увеличения этого нейрон теряет связь между информацией.

Задачей является распознавание образцов в виде символов текстов, образцов звуков. Образцом является представляется в виде вектора значений признаков. Архитектура данной сети продемонстрирована на рисунке 4.

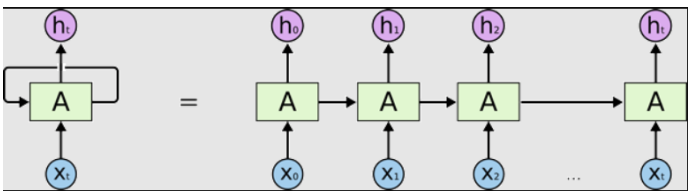


Рисунок 4 - Рекуррентная нейронная сеть

4. LSTM или же сеть долгой краткосрочной памяти.

Одна из популярнейших разновидностей рекуррентной сети. Была создана специально, в целях более точного моделирования временных последовательностей и их долгосрочных зависимостей. Все рекуррентные нейронные сети имеют форму цепочки, которые подражают моделям нейронной сети. Ключевое понятием работы LSTM сети будет состояние ячейки, где состояние ячейки напоминает конвейерную ленту [11]. Проходя через всю длину цепочки, она постепенно, незначительно преобразуется. И в зависимости от потребностей увеличивает или уменьшает количество информации в состоянии ячейки. Для этого используются гейты, которые представляют из себя ворота, тщательно настраиваемые, эти “ворота” могут как пропустить, так и наоборот, не пропустить информацию [12]. Архитектура данной сети продемонстрирована на рисунке 5.

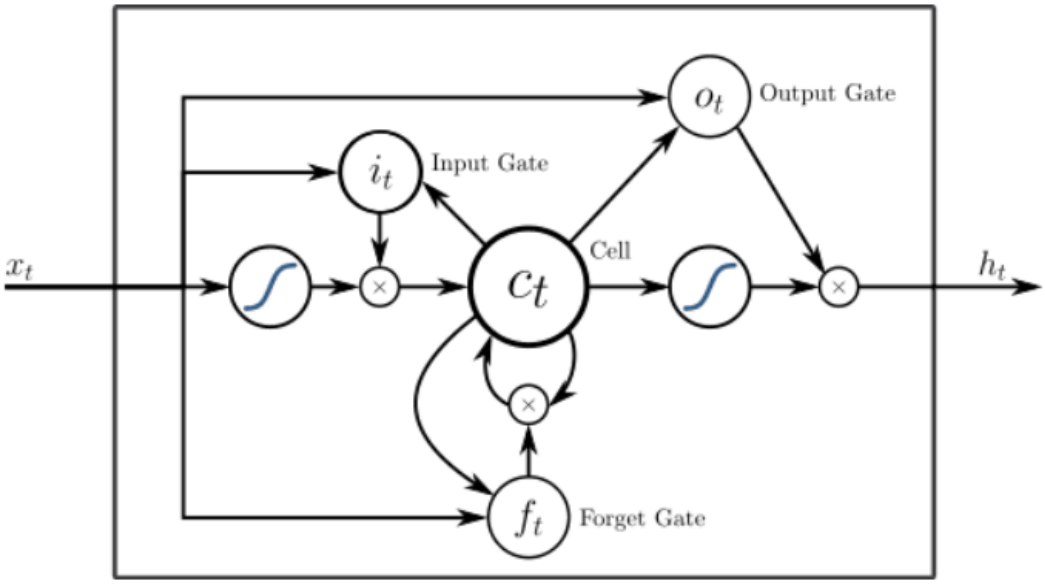


Рисунок 5 – LSTM

Области применения LSTM: в робототехнике, для анализа временных рядов, для распознавания речи, в ритмическом обучении, для генерации музыкальных композиций, для задач распознавания человеческой активности.

5. Модели Sequence-to-Sequence

Модель от TensorFlow состоящий из двух рекуррентных нейронных сетей. Первая обрабатывает данные, его называют encoder, а второй генерирует данные вывода – decoder [14]. Базовая архитектура представлена ниже

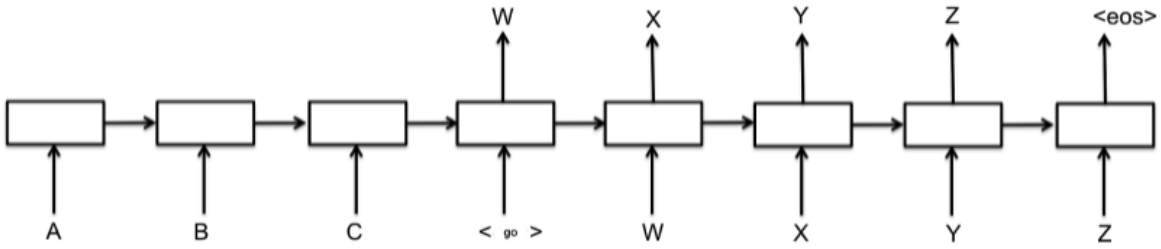


Рисунок 6 - Архитектура модели Sequence-to- Sequence

Прямоугольнике на картинке 6 представляют из себя клетки GRU или же LSTM, который был рассмотрен выше. Каждый ввод в базовой форме модели, должен быть закодирован в вектор состояния фиксированного размера, из-за того, что это единственное, что передается декодеру. Данная нейронная сеть достигла своей популярности за счет машинного перевода, аннотации изображения и др.

Область применения, распознавание изображения, подходит для перевода текста.

6. Сверточная нейронная сеть.

Данная модель показала себя в лучшем свете при работе с распознаванием лиц. Победой над многослойным персептроном обусловлен тем, что он учитывает двумерную топологию изображения. Так же светочная нейронная сеть обеспечивает частичную устойчивость к изменениям на изображении, таким как: масштабам, поворотам, смещениям, смене ракурса и т.п. искажениям. И все это благодаря тому, что текущая архитектура объединяет три архитектурных идеи:

- Локальная двумерная связность.

- Детектирование некоторых черт, что приводит к уменьшению весовых коэффициентов.
- Хаотическая организация с пространственными под выборками

По структуре сверточная сеть состоит из нескольких слоев разных по виду: слои субдискретизирующие, слои сверточные, слои персептрона, то есть обычной нейронной сети [15]. Входной вектор создается с помощью первых двух слоев, это происходит путем их чередования. Архитектура данной сети продемонстрирована на рисунке 7.

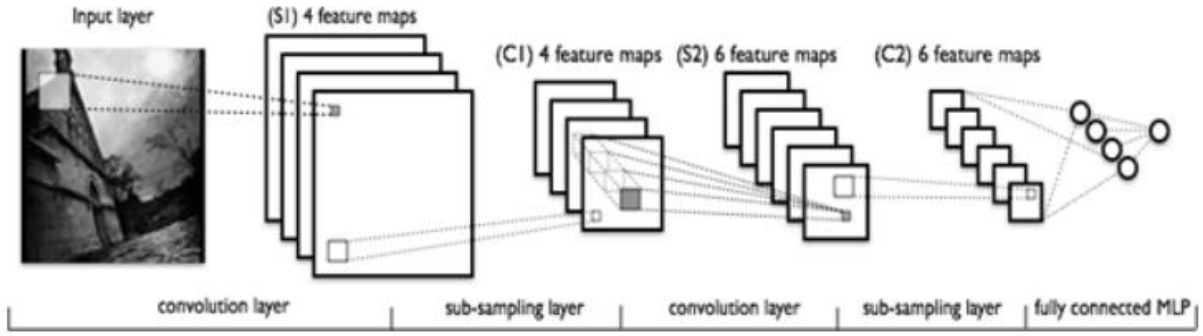


Рисунок 7 – Структура сверточной сети

Область применения является, распознавание изображения, что в данном проекте нам и нужно.

В результате рассмотрения каждого из видов искусственных нейронных сетей, их минусов и плюсов, а также их применений было решено остановить свой взор на сверточной нейронной сети. Данная сеть подходит для нас идеально, так же имеет в себе достаточно моделей, таких как resNet, VGG, Inception [16].

# 2 Проектирование алгоритма распознавания болезни на рентгеновском изображении

## 2.1 Исходные данные для работы с алгоритмом

Имея понимания с какой моделью, будем работать, надо найти подходящие для этой процедуры данные. Это деталь является одной из ключевых моментов и от того, сколько, и как мы обработаем наши данные будет зависеть работоспособность нашего алгоритма.

Данные для работы будут взяты из сайта Kaggle [17]. Данный веб сервис представляет из себя онлайн сообщество специально по машинному обучению, которое позволяет находить и публиковать данные. Для реализации задачи был выбран набор данных рентгеновских снимков за последний квартал. Этот массив содержит свыше 3000 рентгеновских снимков в формате png. На рисунке 8 можно увидеть представление данных изображений.

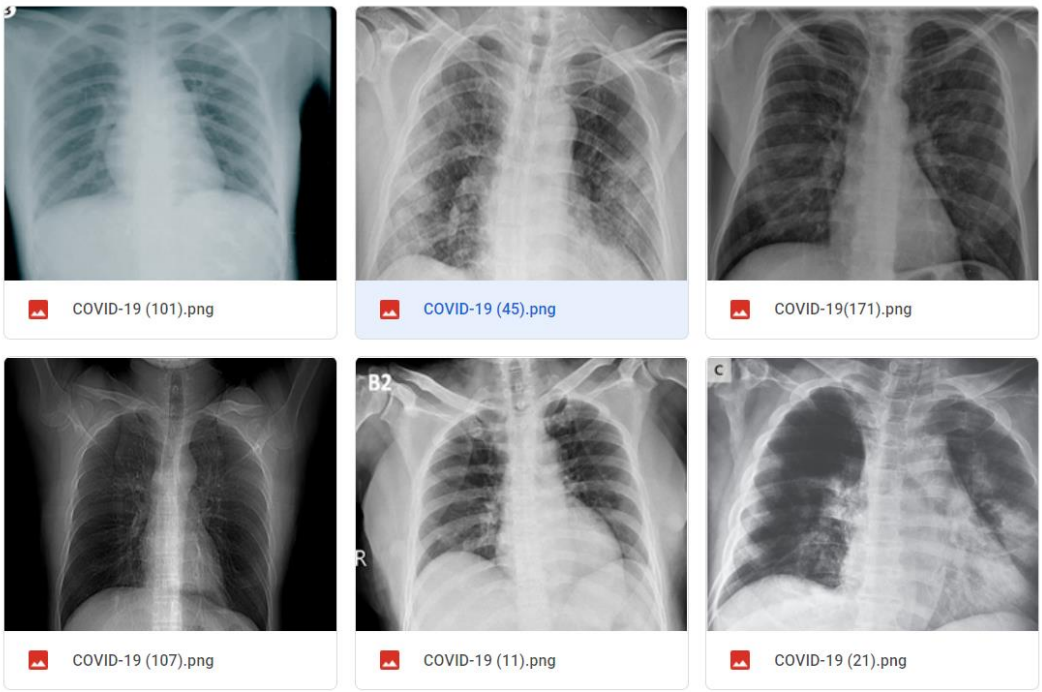


Рисунок 8 – Набор данных рентгеновских снимков

Все эти данные были разбиты на две группы, где 20% используется для тестов и 80% на обучение нашей нейронной сети. Но наша нейронная сеть не поймет как с этим работать, и для этого нам нужно эти данные предобработать Она нужна для того, чтобы заранее упростить работу для нашей нейронной сети. Если бы были не изображения, а таблица, то можно будет убрать лишние параметры, которые не входят в работу. В нашем же случае будет несколько этапов предобработки. По причине отсутствия доступа к мощным машинам, нет возможности обучать сложные модели, которые с высокой точностью могли бы выделить ключевые точки на снимках флюорографии. Поэтому требуется обработка изображения, которая позволила бы упростить работу для машины.

```
train_transform = torchvision.transforms.Compose([
    torchvision.transforms.Resize(size=(224, 224)),
    torchvision.transforms.RandomHorizontalFlip(),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean = [0.485, 0.456, 0.406],
                                     std= [0.229, 0.224, 0.225])

])
```

Рисунок 9 – Нормализация данных

На рисунке 9 демонстрируется участок кода, в котором происходит предобработка данных с помощью библиотеки torchvision, каждый из них отвечает за особое изменение данных:

1. Torchvision.transforms.Resize(size=(224, 224)) – этап в котором мы конвертируем наши изображения под один размер который указывается в парметрах.

2. Torchvision.transforms.RandomHorizontalFlip()- этап для обобщения модели, под названым расширение данных при котором будет переворот данного изображения по горизонтали. Это нужно для сбора еще

большего количества данных, из имеющих. Суть данного метода, состоит в том, что при перевороте изображения из нормального положения, класс останется в том же положение даже после переворачивания.

3. Torchvision.transforms.ToTensor() – шаг преобразования нашего изображения в тензоры. Тензоры — это специализированная структура данных, очень похожая на массивы и матрицы [18]. В PyTorch мы используем тензоры для кодирования входных и выходных данных модели, а также параметров модели. Тензоры похожи на ndarrays NumPy, за исключением того, что тензоры могут работать на графических процессорах или другом специализированном оборудовании для ускорения вычислений.

4. torchvision.transforms.Normalize - заключительный шаг, является нормализацией, достаточно важная часть предобработки изображения, с помощью которого стандартизируем диапазон значений независимых переменных которые были получены при переводе наших данных в тензоры . Данная процедура ускоряет обучение нейронной сети.

По итогу, имея данные, а также представление с какой сетью будем работать нужно определить модель, то есть алгоритм, по которому будет работа нейронная сеть.

## 2.2 Выбор модели нейронной сети

Имя понимание того, с какой сетью будем работать, а также данные для этого, следующим этапом будет выбор модели. Для этого проверим модели, которые работают на наборе данных ImageNet, которая включает в себя свыше 1.3 миллионов тренировочных, а также 50 тысяч проверочных данных, вместе с тем, что внутри себя он включает 1000классов. Это нужно для точности нашего выбора. Имея одинаковые данные, мы поймаем, какая модель более эффективна и по итогу выберем подходящий для нас.

Так как сети выполняют задачу классификации и используют на конце SoftMax-слой результатом сети будет является вектор $(x_1, x_2, x_3 \dots x_n)$, где n-

количество классов, а $x_i$ является вероятностью отношений входного изображения к i-му классу [19]. Точность измеряется в данном случае будет измеряться двумя вариантами top-1 и top-5. Top-1 будет обозначать, что наибольшее количество классов соответствуют $x_i$, а второй вариант, говорит, нам что правильный класс принадлежит пяти самым высоким значениям в выходном векторе сети. На таблице 1 указаны результаты проведенной проверки по метрике accuracy [20]. Данная метрика является отношением правильных ответов к общему их количеству.

Таблица 1 – Результат тестирования моделей

| Модели | Кол-во параметров | Top-1 | Top-5 |
|---|---|---|---|
| VGG-16 | 138 357 544 | 71.3% | 90.1% |
| VGG-19 | 143 667 240 | 71.3% | 90.0% |
| ResNet-18 | 10 857 784 | 77.8% | 93.6% |
| ResNet-50 | 44 675 560 | 78.2% | 93.9% |
| ResNet-101 | 60 380 560 | 78.3% | 94.5% |
| DenceNet-121 | 8 062 504 | 75.7% | 91.9% |
| DenceNet-169 | 20 242 984 | 77.4% | 92.3% |

Как можно видеть на таблице: сети VGG имеет самое большое количество параметров, но в ту же очередь имеют самую низкую точность. А сети DanceNet показывают лучшее соотношение точности и количества параметров. Но лучшим показал себя ResNet, а именно 18, в ней имеется меньшее количество параметров, что дает большую скорость, а также точность.

## 2.3 Описание работы алгоритма распознавания

Имея представление с какой моделью, будем работать, надо понять структуру, что происходит на каждом слое обучения, а также между ними.

Входной сигнал изображения подается только на прохождение области по правилу квадрата, например 3n 3 пикселей, область сдвигается на 1 пиксель, сигнал контента покрывается вторым пикселем и т.д [21] . Шаг изображение, как в примере 1px, будет отсканировано. И после этого каждая маска 3 на 3 пикселя подается в скрытый нейрон, при чем весовые коэффициенты одинаковы для всех нейронов от первой скрытой группы нейронов. То есть, когда мы сканируем изображение то мы сканируем это с одними и теми же весовыми коэффициентами. После того, как мы отсканировали один раз изображение цикл повторяется, но уже с другим набором весовых коэффициентов. И данный новый набор весовых коэффициентов подается уже на другу. Группу нейронов, которые обрабатывают этот входной сигнал. В итоге у нас получается слои нейронов со своими весовыми коэффициентами. Таких групп может быть n количество. Они идентичны друг другу, кроме своих весовых коэффициентов.

Теперь нужно рассмотреть, что получается на входах каждой отдельной группы, так как весовые коэффициенты в пределах группы не меняются, то мы фактически имеем окно, в нашем случае как в примере 3 на 3 пикселя с набором определенных чисел [22]. Эти числа умножатся на соответствующие входные значения потом суммируются между собой и получается выходной сигнал математически это можно записать по формуле 1:

$$\nu_{0,0} = \sum_{i=1}^{3} \sum_{j=1}^{3} x_{i,j} * \omega_{i,j} + \omega_0 \,, \qquad (1)$$

где $\nu_{0,0}$ – входной сигнал в начальной точке

$x$ – значение тензора

$\omega$ – весовые коэффициенты

После происходит смещение, тем самым берутся совершенно другие данные и формируется уже другая формула 2:

$$\nu_{0,1} = \sum_{i=1}^{3} \sum_{j=1}^{3} x_{i,j+1} * \omega_{i,j} + \omega_0 \,, \qquad (2)$$

где значения те же что и в формуле 1.

И по итогу мы получаем множество значений $\nu$ , в общей формуле это будет выглядеть, как на формуле 3:

$$\nu_{k,m} = \sum_{i=1}^{3} \sum_{j=1}^{3} x_{i+k,j+m} * \omega_{i,j} + \omega_0 \; k, m = 0,1,2, \ldots, \qquad (3)$$

где где у нас k, m меняются, смотря какое дано нам изображение.

И мы получаем общее число настраиваемых параметров в пределах одной группы нейронов равная в нашем случае n*(3*3+1) (1 = смещение bias). Сумма $\nu_{k,m}$ в цифровой обработке называется сверткой. Каждое ядро выделяет свои характерные закономерности на изображениях сканируя его по всему изображению и на выходе будет формироваться набор каждых признаков, который называется каналами [23]. То есть у нас n групп нейронов и на выходе получаем n каналов. Значимые величины в каждой карте показывает наличие признака в строго определённом месте изображения. И далее используя эти карты признаков следующие группы нейронов может их обрабатывать и делать дальнейшее обобщение, но, например выделять более сложный признаки, такие как: эллипсы, прямоугольники, различные пересечения линий и т.п. Данная схему можно рассмотреть повнимательнее на картинке 10.

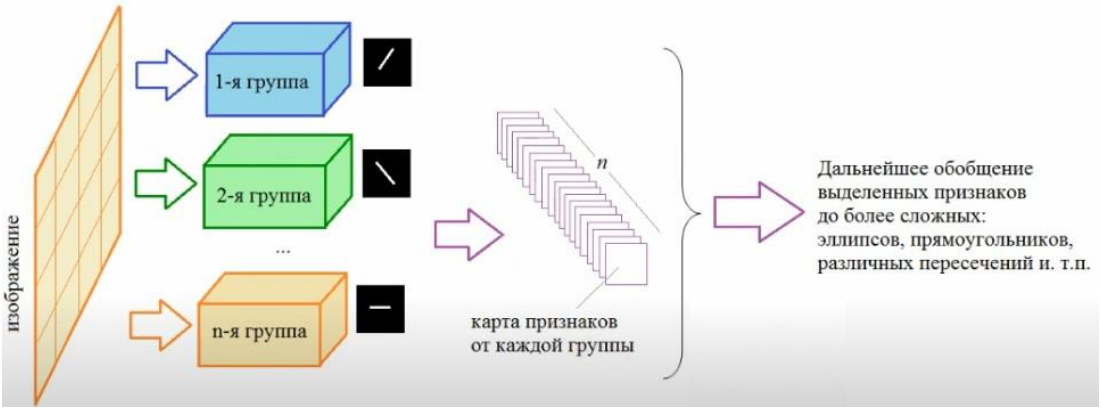


Рисунок 10 - Концепция нейронной сети

Это общая концепция каналов сверточной нейронной сети, но это простейший вариант, когда на вход подавалось одноканальное изображение, как пример, в оттенках серого, если же обрабатывается полноценное цветное изображение, представленное тремя цветными компонентами RGB, то каждая цветовая компонента обрабатывается своим отдельным фильтром и для каждого цветовой компоненты у нас образуется своя карта признака. После они складываются между собой, то есть три карты признаков каждого цвета и к ним после сложения добавляется bias, то есть прибавляется к общей сумме.

## 2.4 Выбор средств для реализации программы

Для реализации, которую рассмотрели походят лишь 2 языка это Python и MatLab, при помощи которых можно написать программный код. И в следствии большего количества информации был выбран язык python также с ним проще работать, так же имеется, достаточное количество библиотек для работы вместе с ней. Так же немаловажным плюсом, который стоит отметить, так эта скорость работы, которая важна, при обучении.

Быстрой разработке на данном языке способствуют простой синтаксис языка, не содержащий сложных конструкций, а также большое количество библиотек, созданных сообществом, встроенные функции которых уменьшают количество написанного кода.

Питон предоставляет достаточно много возможностей, от этого мы можем использовать, а точнее запускать программы на большинстве современных аппаратах. Данная деталь, будет нам плюсом, в том, что не придется заботиться об аппаратуре в мед учреждениях.

Сам по себе язык программирования, написан на одном из мощных языков - c++. Данная деталь дала то, что теперь питон работает в разы быстрее конкурентов. Но есть свои направления, где он слаб. Это математика, в данной области он отстает, как раз для таких моментов, лучше переходить на главный источник, а точнее на язык, с помощью которого он изначально был написан.

Для полноценной работы понадобятся библиотеки, в большинстве проектов они повторяются, так что кратко опишем их:

- Random - библиотек, которая используется для генерации произвольного числа в зависимости от метода в определенных значениях. Полезен для генерации массивов из произвольных чисел. Имеет плюс в мобильности и большом количестве методов.
- NumPy - библиотека, которая используется для научных вычислений. Содержит мощный N-мерный объект массива, а также полезен в линейной алгебре и может использоваться в качестве эффективного многомерного контейнера для общих данных.
- MatPlotLib/PyPlot – библиотека для построения 2-D графиков. Используется для визуализации при анализе данных.
- Torchvision – библиотека из фреймворка PyTorch. Используется для трансформации изображении. Имеет обширное количество инструментов для работы с изображениями.
- PIL – библиотека изображений, нужна для обработки графиков в pyton. Имеет обширный инструментарий, что позволяет создавать любого рода графики, что нам в дальнейшем понадобится.

Следующим этапом после подключения библиотек, является выбор

Фреймворка, который послужит шаблоном для нашей работы, разберем плюсы и минусы трех Фреймворков и выберем один из них, с которым будем работать:

1. PyTorch

Плюсы:

- благодаря архитектуре фреймворка, процесс создания модели достаточно прост и прозрачен;
- режим по умолчанию “define-by-run” – отсылка к традиционному программированию. Фреймворк поддерживает популярные инструменты для дебага, такие как pdb, ipdb или дебаггер PyCharm;
- он поддерживает декларативный параллелизм данных;

- он имеет много предварительно обученных моделей и готовых модульных частей, которые легко комбинировать;

Минусы:

- недостаточная поддержка моделей;
- он еще не готов для полноценного выхода в продакшн, однако дорожная карта к версии 1.0 выглядят действительно впечатляюще;
- недостает интерфейсов для мониторинга и визуализации, как TensorBoard – однако он имеет внешнее подключение к Tensorboard.

2. TensorFlow

Плюсы:

- для нее написано большое количество руководств и документации;
- она предлагает мощные средства мониторинга процесса обучения моделей и визуализации (Tensorboard);
- она поддерживается большим сообществом разработчиков и техническими компаниями;
- она обеспечивает обслуживание моделей;
- она поддерживает распределенное обучение;
- TensorFlow Lite обеспечивает вывод на устройства с низкой задержкой для мобильных устройств;

Минусы:

- Она проигрывает по скорости работы в эталонных тестах, в сравнении с CNTK и MXNet, например; она имеет более высокий входной порог для начинающих, чем PyTorch или Keras. Голая Tensorflow достаточно низкоуровневая и требует много шаблонного кода, и режим «определить и запустить» для Tensorflow значительно усложняет процесс дебага.

3. Keras

Плюсы:

Прототипирование действительно быстрое и простое;

- Он достаточно маловесный для построения моделей глубокого обучения для множества слоев; Имеет полностью конфигурируемые модули;
- Имеет простой и интуитивно-понятный интерфейс, соответственно, хорош для новичков;
- Имеет встроенную поддержку для обучения на нескольких GPU;
- Может быть настроен в качестве оценщиков для TensorFlow и обучен на кластерах GPU на платформе Google Cloud;
- Запускается на Spark;
- Поддерживает GPU от NVIDIA, TPU от Google, GPU с Open-CL, такие как AMD.

Минусы:

- Может оказаться слишком высокоуровневым и не всегда легко кастомизируется;
- Он ограничен бэкэндами Tensorflow, CNTK и Theano.

В итоге был выбран PyTorch [24]. Начнем с того, что это одна из самых популярных на данный момент фреймворков, от неё были получены самые эффективные результаты. Подключение всех библиотек и Фреймворка показано на изображении 11.

```
import shutil
import random
import os
import torchvision
import numpy as np

from PIL import Image
from matplotlib import pyplot as plt

import torch
from  torch.utils.tensorboard import SummaryWriter
torch.manual_seed(0)

print('Using PyTorch version', torch.__version__)
```

```
Using PyTorch version 1.8.1+cu101
```

Рисунок 11 – Вызов библиотек и Фреймворка

На основании данных исследований, было заключено использовать фреймворк PyTorch, который предоставит нам структуру, а также библиотеки для работы с нашей задачей.

Для первичного дата сета картинок был выбран сайт Kaggle, где проходят соревнования по программированию, и дается в бесплатный доступ изображения, так же таблицы с нужной информацией. В будущем привязкой к одному дата сету не даст хороших показательней, но для начала будет достаточно.

## 2.5 Реализация программы

После этапа с предобработкой изображений мы получи наши данные в виде математического тензора. Данный объект представляет собой математический объект, при этом он не зависит от внешних фактор, как пример система координат. Но при этом, при смене координат они по определенному математическому закону, что не нарушит целостность данных. В трехмерном пространстве, тензор в данном случае второго ранга, представляет собой python матрицу.

Следящим шагом является инициализация модели ResNet18 [25], она послужит нам каркасом для нашего алгоритма. На рисунке 12 показан кусок кода. В последующих действиях мы будем менять, а также добавлять функции в определённые слои.

```
resnet18 = torchvision.models.resnet18(pretrained=True)
print(resnet18)
```

```
resnet18.fc = torch.nn.Linear(in_features=512, out_features=3)
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet18.parameters(), lr=3e-5)
```

Рисунок 12 – Запуск модели

Resnet – произошел от residual нейро сети, то есть нейросеть построенная на остатках. Архитектура модели была вдохновлена VGG и суть ResNet в том, что мы не просто там создаем массив из сотен слоев, но мы добавляем промежуточные связи между слоями и таким образом обучение получается легче. То есть у нас нет, как допустим в VGG затухания [24]. Изображение модели показана на рисунке 13. В целом данная модель идет средне и дает достаточно хорошую точность. Тем самым он побеждал на соревнованиях, даже таких гигантов, как google.

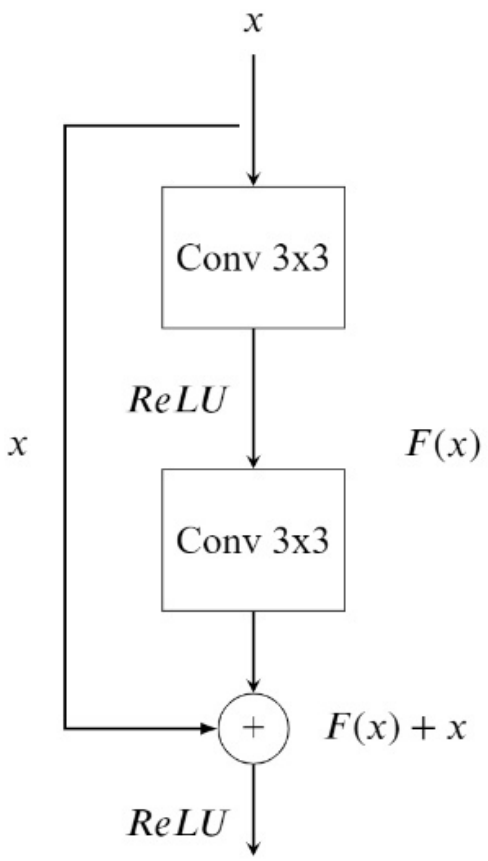


Рисунок 13 – Блок остаточной модели

Теперь поподробнее про архитектуру, она имеет ствол, который состоит из 64 фильтров размерами 7 на 7 с шагом 2(то есть он выполняет роль и свертки и под выборки) и он сужает изображение в 2 раза после получаем

некоторый базовый набор особенностей. Далее все выделенные нам пиксели мы отправляем через шаг и пропускаем половину для обработки в блоке 3 на 3 а остальную часть оставляем не обработанной для того, чтобы в следующем блоке нейронная сеть разбиралась, среди информации об изображениях, что ей важно, то есть нужны ли ей фильтры 3 на 3. Таким образом она частична сражается с проблемой градиентов и позволяет обучать чуть больше глубоких слоев.

Существует достаточно различные модели resnet, но resnet18 – это более компактная модель среди всех остальных, которую можно быстро обучить и при этом дает достаточно хорошие результаты. При запуске и выводе resnet модели было выявлено, что resnet была обучена на наборе из 1000 классов. Таким образом данная модель нам не подойдет, так что было принято решение изменить полностью последний слой, который есть у resnet18 на 3 выходных признака. Как видно на изображении n fc последний слой. Поскольку мы будем использовать классификацией, мы будем использовать соответствующую функцию потерь CrossEntropyLoss() [25]. Данный метод в математическом представлении показан на формуле 4:

$$loss(x, class) = \log(\sum j\, e^{x[j]} * e^{x[class]})) = \\ = -x[class] + \log(\mathrm{j} \sum e^{(x[j])}) \qquad (4)$$

А также будет использоваться метод Adam. Данный алгоритм. На вход будут подаваться следующие значения: Начальные значения $x_1$, параметры $\beta_1, \beta_2 \in [0,1)$, шаг $\alpha$, константа $\epsilon$. После получения значений проходим по циклу, чтобы пройти каждый слой. Алгоритм выглядит следующим образом:

1. Вычисление значений функций стоимости по формуле 5:

$$g_t = \nabla f(x_t) \qquad (5)$$

2. Вычисление значения ошибок выходного слоя:

$$m_t = \beta_1 m_{t-1} - (1 - \beta_1) g_t \quad (7)$$

3. Вычисление ошибки для каждого предыдущего слоя

$$\nu_t = \beta_2 \nu_{t-1} - (1 - \beta_2) g_t^2 \quad (8)$$

4. Вычисление градиента функции стоимости

$$x_{t+1} = x_t - \alpha \frac{m_t}{\sqrt{\upsilon_t + \epsilon}} g_t \quad (9)$$

5. Обновление весов связи

По итогу было изменен последний слой нейронной сети, что за собой подвиг ускорение данной нейронной сети, а также повышение точности. Что мы в самом начале хотели получить. По итогу было получено совершенно новая система, эффективность которой будет протестирована в следующей главе.

Следующим шагом стало создание метода для вывода пяти изображения из нашего списка, где мы могли бы увидеть правильно ли, спрогнозировала модель. Программный код указан чуть ниже, на рисунке 15.

Первым действием задаем размер фигуры, для корректного вывода 6 изображений было оптимально выбрать размеры 8на4. А теперь отдельно поговорим про каждый метод, находящийся внутри цикла, про все методы над каждым изображением:

- pyplot.subplots - создает фигуру и сетку подзаголовков с помощью одного вызова, обеспечивая разумный контроль над тем, как создаются отдельные графики.
- image.numpy().transpose() – Преобразует тензорное представление

изображения в многомерный массив numpy и при этом транспонировка происходит так, чтобы ось 0 была последней, поскольку у ResNet18 канал первый. После идут параметры ширины и высоты.

```
  ,
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

Рисунок 14 – Информация о модели ResNet18

```
def show_images(images, labels, preds):
    plt.figure(figsize=(8,4))
    for i, image in enumerate(images):
        plt.subplot(1,6, i + 1, xticks = [], yticks = [])
        image = image.numpy().transpose((1, 2, 0))
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        image = image * std + mean
        image = np.clip(image, 0., 1.)
        plt.imshow(image)
        col = 'green' if preds[i] == labels[i] else 'red'
        plt.xlabel(f'{class_names[int(labels[i].numpy())]}')
        plt.ylabel(f'{class_names[int(preds[i].numpy())]}', color=col)
    plt.tight_layout()
    plt.show()
```

Рисунок 15 – Метод по выводу изображений

Далее были выведены значения, при которых мы нормализовали наши данные, это было сделано для обратной нормализации, чтобы вернуть пиксельное изображение в исходное состояние, каким оно было до нормализации.

Последним шагом проходила проверка прогноза. То есть мы использовали метку для отображения истины или лжи прогноза. Если предсказание верно, то цвет текста отобразиться зеленым. Если оно не верно, то оно будет отображено красным.

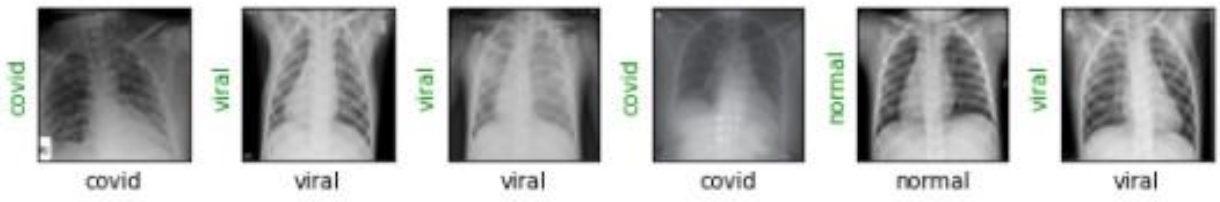


Рисунок 16 – Пример вывода изображений

А теперь переходим к главной части, к части обучения модели.

```
def train(epochs):
  print('Starting training..')
  for e in range(0, epochs):
        print('='*20)
        print(f'Starting epoch {e + 1}/ {epochs}')
        print('='*20)
        train_loss = 0
        resnet18.train()
        for train_step, (images, labels) in enumerate(dl_train):
            optimizer.zero_grad()
            outputs = resnet18(images)
            loss = loss_fn(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            if train_step % 20 == 0:
                print('Evaluating at step:', train_step)
                acc = 0.
                plt = 0.
                val_loss = 0.
                resnet18.eval()
                for val_step, (images, labels) in enumerate(dl_test):
                    outputs = resnet18(images)
                    loss = loss_fn(outputs, labels)
                    val_loss += loss.item()


                    _, preds = torch.max(outputs, 1)
                    acc += sum((preds == labels).numpy())
                    plt += sum((preds != labels).numpy())
                    val_loss /= (val_step + 1)
                acc = acc / len(test_dataset)
                plt = plt / len(test_dataset)
                print(f'Val loss:{val_loss: .4f}, Acc:{acc: .4f},
                plt:{plt: .4f}')
                show_preds()
                resnet18.train()

                if  acc > .97:
                    print('Performance condition satisfied')
                    return
        train_loss /= (train_step + 1)
        print(f'Training loss: {train_loss: .4f}')
        %%time
```

Рисунок 17 – Метод обучения

Метод принимает лишь один параметр — это количество эпох. Внутри каждой последующей эпохи будет больше итераций для каждого шага

обучения. И после каждого шага обучения в каждой эпохе оценим модель и посмотрим какую точность мы получаем. После в цикле оптимизатор устанавливаем на значение 0 и в дальнейшем, когда начнем получать потери мы реализуем градиентный шаг. Данный метод представляет из себя оптимизацию, которая может найти минимум целевой функции. Но он достаточно жадный, который находит оптимальное решение, путем того, что делаем шаг к максимальной скорости уменьшения функции.

Так же было добавлено условие, при котором мы узнаем оценку модели при каждом двадцатом шаге. После этого будет расчёт точности прогноза. И по итогу resnet с помощью метода eval перейдет в режим оценки.

Далее переходим в очередной цикл, в котором происходит прогноз. В качестве прогноза мы берем индекс, который получаем с tourch.max. Следующим этапом добавляем к точности количество правильных предсказаний, которые мы имеем. То есть для оценки сначала делаем прогнозы на тестовых наборе, сохраняя индексы прогнозов в tourch.max, а затем создаем и выводим отчет о точности

И в конечном итоге мы ставим отметку точности, в данной работы были выбраны значения 95%.

В данной главе был описан процесс создания алгоритма, способного распознать COVID-19 на изображении, выбор средств для реализации данного алгоритма и сама реализация. В конечном итоге в данной главе был описан алгоритм, который способен получать на вход изображение, обработать и подать его на вход в нейронную сеть, которая распознает, что на картинке и выведет на экран результат.

# 3 Тестирование измененного алгоритма

## 3.1 Тестирование разработанного алгоритма с использованием различных изображений

После получения модель ResNet с модернизированным последним слоем, модно с уверенностью переходить к этапу тестирования и понять, на сколько были эффективны изменения.

Но перед этим вспомним, какие функции выполняет программа:

1. Предобработка данных.
2. Использование кросс-валидацию.
3. Применение для обучающих данных data augmentation.
4. Создание экземпляра нейронной сети с её настройкой.
5. Запуск обучение и передача ей картинок.
6. Вычисление матрицы неточностей.
7. Вывод графика точностей.
8. Вывод через каждый 20 шагов процент потерь, а также точность.
9. Вывод на экран прогноза по картинке.

Данные для тестов были взяты из вышеупомянутого сайта Kaggle в размере 3 тысяч штук, так же для тестов будет использоваться процедура кросс валидации, о которой сейчас поговорим.

Данная процедура предоставляет нам, то, что мы узнаем точность не с помощью одних тестовых данных, а сразу с нескольких. То есть мы имеем некое количество данных, допустим в нынешней ситуации у нас имеется 3000 снимков, которые мы разделим на n частей. В нашем случае я взял за n = 10, так как посчитал это рациональным значением, то есть не больше и не меньше. После этого поочередно каждый кусочек из 300 изображений проходит тестирование, а остальные 2700 рентгеновских снимков проходят обучение и по итогу мы получаем первую оценку точности нейронной сети. Так проходит

еще девять раз, где меняются данные для тестов. После данной процедуры высчитывается среднее значение, которое в данном случае предоставит нам точное значение точности нейронной сети. Но у данного применения есть свои минусы, одним из которых так же, как и самый главный, это то, что на оценку нейронной сети потратиться достаточно большое время, хоть данных и достаточно, но при большем объеме данное время еще сильнее увеличиться, но оценка с помощью такого метода будет все же превыше всего. Особенно в тот момент, когда от этого зависят жизни сотен тысяч людей.

Но до этого давайте протестируем без нее, прогоним тесты лишь одни раз, где при запуске программы было решено, что одной эпохи для достижения задачи хватит. На изображении 18 можно первые шаги и проценты точности, вместе с потерями. Так же можно увидеть постепенный спады и подъемы точностей, что является обычным делом в данной процедуре и не является ничем примечательным.

На изображении можно увидеть, как манятся при каждом двадцатом шаге количество точности и потерь, но после 40 шага, данная точность равна 90%, но, чтобы достичь желаемых 95% по условию пришлось дойти до 220 шага, что можно будет доработать в дальнейшем. А так на графике 18 можно увидеть, качество модели спустя 10 итерацией с помощью кросс валидации.

И итоговое качество высчитывается по формуле n и получаем значение 0,909

$$E = \sum_{i=1}^{10} E_i \quad (5)$$

Как пример будет показана один из шагов проходящий в кросс валидации, по итогу которого нам выдается на 220 шаге информацию, что точность вышла 97% и программа завершена, так же дополнительно к этому изображение 6 рентгеновских снимков.

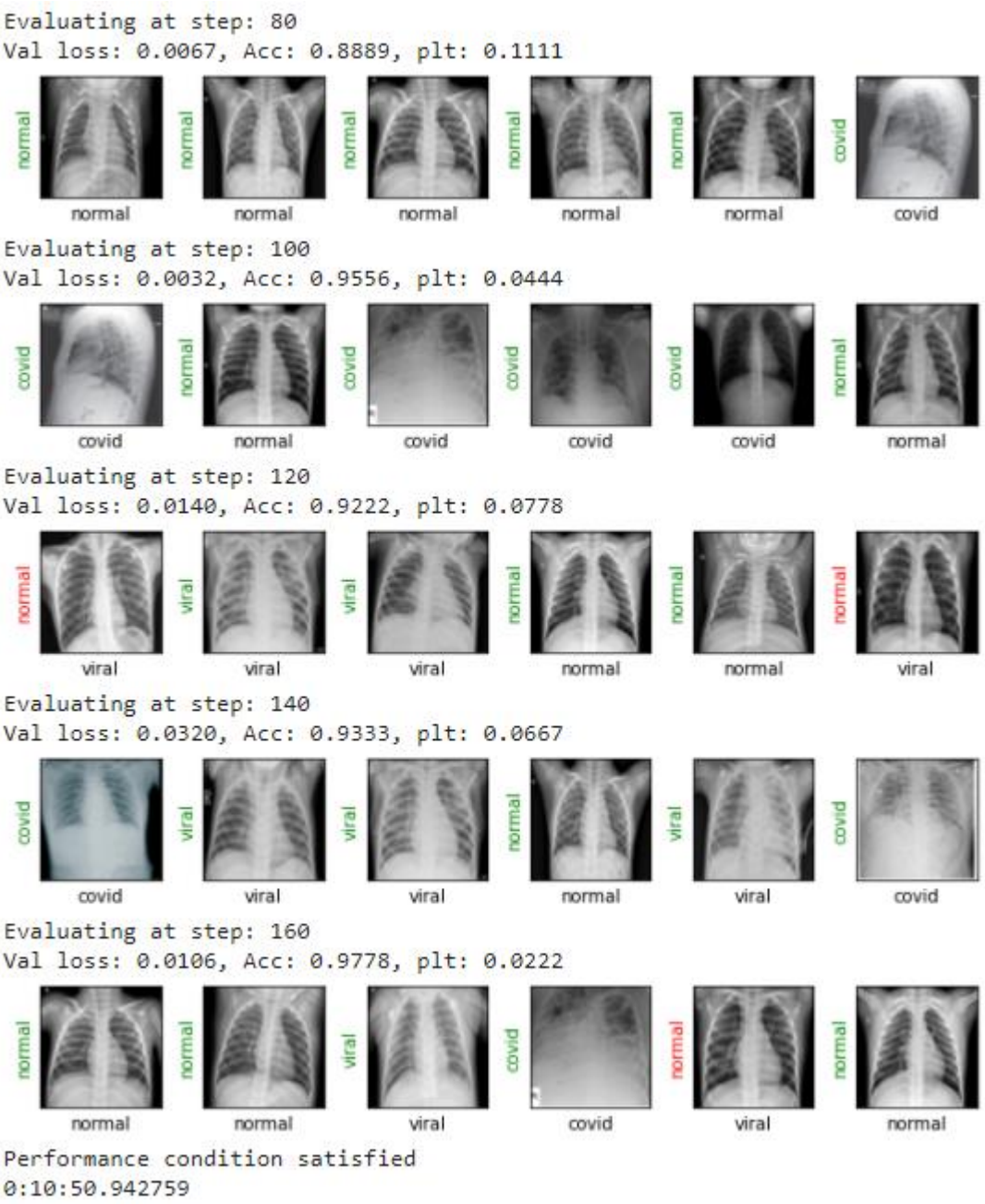


Рисунок 18 – Вывод обучения

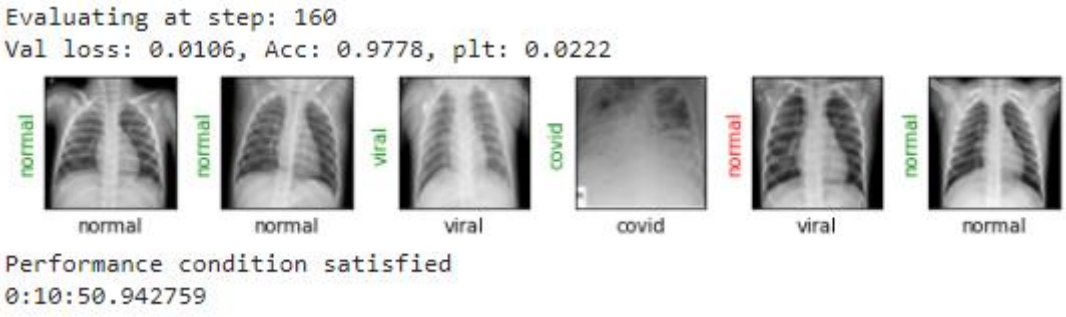


Рисунок 19 – Конечный вывод в консоли обучения

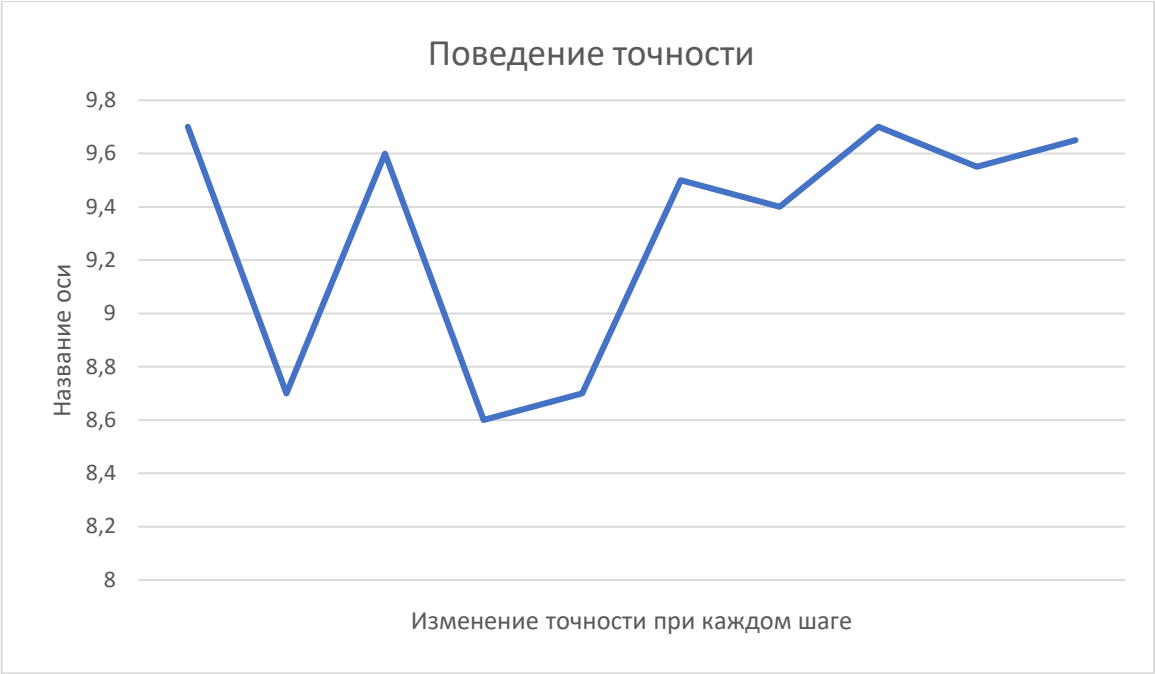


График 1 – Качество кросс валидации

В ходе тестов были взяты не одинаковые изображения, а также изображения с пневмонией, что данный алгоритм, тоже может распознать. Так же каждый день база данных Keggle обновляется и появляются новые соревнования по ковид-19 с свежими дата сетами. Так же в процессе тестов был еще один запуск с теми же данными, что и прежде, но на этот раз шагов стало меньше, а процент точности увеличился. На картинке 20 вывод последующего теста

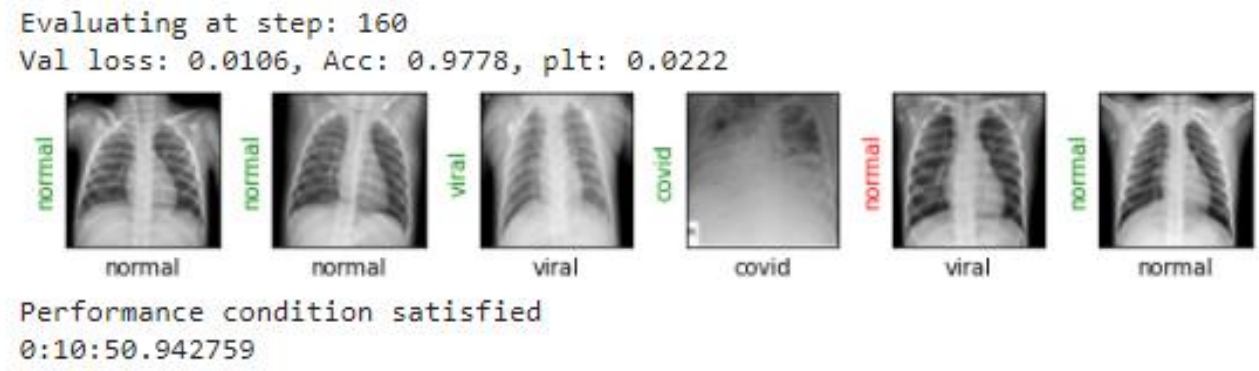


Рисунок 20 – Конечный вывод в консоли обучения при повторном тестировании

Из проведенных тестов можно сказать, что программа работает корректно и справляется с поставленной задачей.

## 3.2 Подведение итогов тестирования разработанного алгоритма

Во время работы выполнения бакалаврской работы было проведено тестирование алгоритма распознавания COVID-19 на флюорографических снимках. Во время тестирования на вход алгоритма подавались разные изображения, имеющие различия между собой. Так же данные хранятся в открытом доступе на гугл диске. Первые 6 картинок из данного дата сета представлены на изображении 21

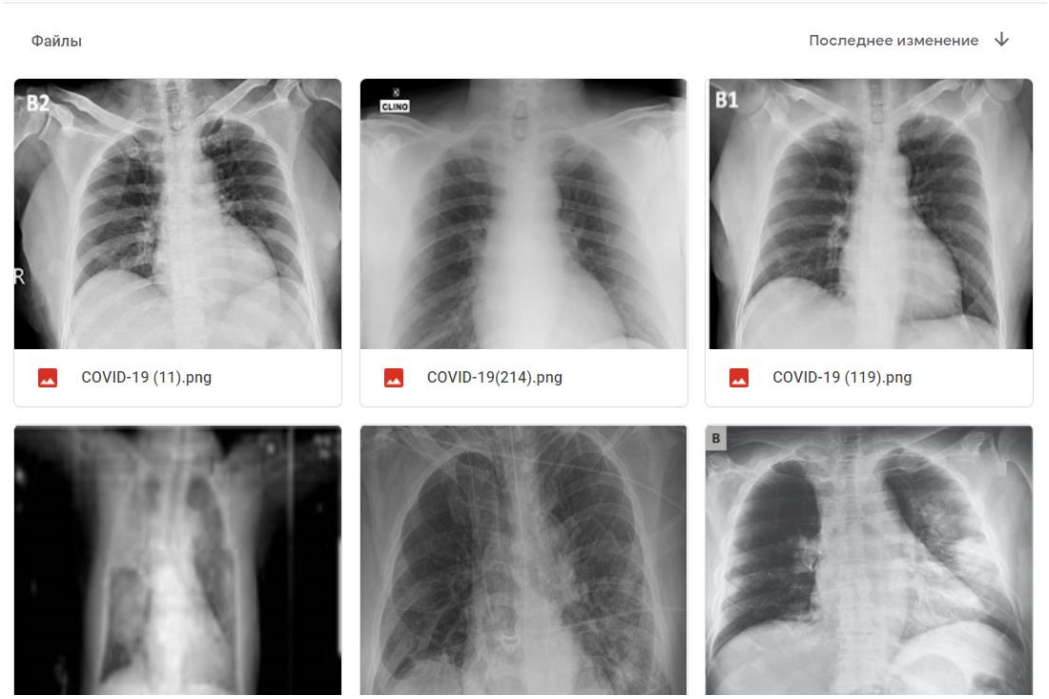


Рисунок 21 – Изображения рентгеновских снимков.

На данный момент используется только изображения в то время, как детекторы кодвид-19 в идеале должны использовать несколько источников данных, не ограничивая себя лишь изображениями. И как бы это было не желанно, но время на обучение тратиться существенное. Для данного действия понадобилось установить дополнительную библиотеку datetime

```
from datetime import datetime
import time

start_time = datetime.now()

train(epochs=2)

print(datetime.now() - start_time)
```

Рисунок 22 – Код по просчету времени

Теперь при завершении работы алгоритма с теми же данными мы можем получить время, при котором происходит обучение, данный рисунок с временем находиться на изображении 23.

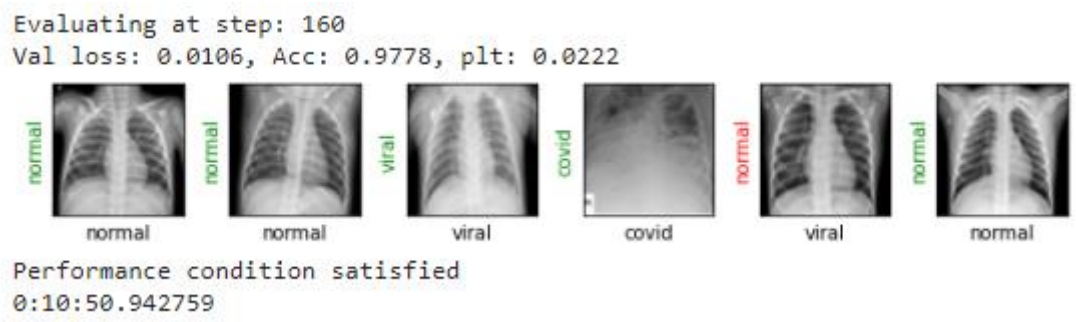


Рисунок 23 – Вывод в консоль времени выполнения работы

Так же настоящий детектор COVID-19 должен пройти тщательное тестирование подготовленными медицинскими специалистами, работающими рука об руку с опытными практиками глубокого обучения. Описанный выше алгоритм более подходит для образовательных целей, нежели для практики.

При учете того, что каждый день по статистике появляются все больше и больше больных пациентов, данная работа будет актуальной. Так как быстрота и удобство данного алгоритма сыграют не первостепенную важность за счет автоматизировать подачу изображений, путем добавления UI интерфейса для пользователя. Что облегчит задачу предоставления данных, а также упростит взаимодействие пользователя с программой.

## Заключение

В первой главе рассматривалась болезнь, как её обнаружить на снимках легких, какие есть трудности, а также решения. Также проговорили какие бывают искусственные нейронные сети, а также выбрали одну из них с помощью которой будет происходить глубокое обучение.

Были описаны виды нейронных сетей, их архитектура, а также плюсы и мигну каждого из них. Поговорили в каких целях они используются и по итогу определили какую нейронную сеть будем использовать для работы.

Во второй главе первым шагом стала нормализация данных, которые получили с ресурса Kaggle. Процесс, который производится сильно влияет на обучение модели. После происходил выбор программного языка, также и библиотек, и Фреймворка для работы с поставленной задачей. В этой же главе происходило создание алгоритма по модели ResNet18, где мы удачно изменили последний слой модели добавив модель потерь, а также метод Адама.

В третьей главе, было осуществлено тестирование алгоритма путем кросс-валидации. Для большей точности качества модели использовался алгоритм кросс валидации. По итогу которого были получены точные данные того, на сколько алгоритм является успешным.

В ходе разработки программы были оправданы ожидания, полученный результат был удовлетворительным.

## Список используемой литературы


1. Астахова И. Ф. Компьютерные науки. Операционные системы, сети [Электронный ресурс]: учебное пособие / И. Ф. Астахова [и др.]. - Москва : ФИЗМАТЛИТ, 2013. - 88 с.

2. Васильева, Е.В., Левит, Б.Ю., Лившиц, В.Н. Нелинейные транспортные задачи на сетях. – М: Финансы и статистика, 1981.

3. Вильсон, А. Дж. Энтропийные методы моделирования сложных систем. – М: Наука, 1978.

4. Головко О.В. Высшая математика. Часть I. Матрицы и определители. Системы линейных уравнений. Векторная алгебра и аналитическая геометрия [Электронный ресурс]: учебное пособие/ Головко О.В., Дадаева Г.Н., Салтанова Е.В.— Электрон. текстовые данные.— Кемерово: Кемеровская государственная медицинская академия, 2006.— 56 с.— Режим доступа: http://www.iprbookshop.ru/6111.html.— ЭБС «IPRbooks»

5. Документация по Python [Электронный ресурс] / URL: https://www.python.org/doc/ (дата обращения 10.02.21)

6. Лацис А. О. Параллельная обработка данных : учеб. пособие для студ. вузов, обуч. по спец. «Прикладная математика и информатика» / А. О. Лацис. - Гриф УМО. - Москва : Академия, 2010. - 335 с.

7. Левин М.П. Параллельное программирование с использованием OpenMP [Электронный ресурс]/ Левин М.П.— Электрон. текстовые данные. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 133 с.— Режим доступа: http://www.iprbookshop.ru/52216.html.— ЭБС «IPRbooks»

8. Партыка Т. Л. Операционные системы, среды и оболочки : учеб. пособие для студ. учреждений сред. проф. образования, обуч. по спец. информатики и техники / Т. Л. Партыка, И. И. Попов. - 3-е изд., перераб. и доп. ; Гриф МО. - Москва : ФОРУМ, 2010. - 543 с.

9. Тарик Рашид «Создаем нейронную сеть». – Вильямс, 2017 – ISBN:

978-5-9909445-7-2.

10. Умнов А. Е. Аналитическая геометрия и линейная алгебра: учеб. пособие/ А. Е. Умнов. – 3-е изд., испр. и доп. - М.: МФТИ, 2011. – 544 с.

11. Шолле «Глубокое обучение на Python». – Питер, 2018 –ISBN: 978-5-4461-0770-4

12. Что такое свёрточная нейронная сеть [Электронный ресурс]

Режим доступа: https://habr.com/ru/post/309508/ (дата обращения 15.05.20)

13. Якимов, М. Р. Транспортное планирование: создание транспортных моделей городов: монография / М.Р. Якимов. – М.: Логос, 2013. – 188 с.

14. Keras Documentation [Электронный ресурс]. – Режим доступа: https://keras.io/api/ (дата обращения: 15.02.20)

15. Mohammad Abdur Razzaque PhD, Md. Rezaul Karim. Hands-On Deep Learning for IoT: Train neural network models to develop intelligent IoT applications // Packt Publishing, 2019

16. OpenCV: API Documentation [Электронный ресурс]. – Режим доступа: https://docs.opencv.org/2.4/modules/refman.html (дата обращения: 15.02.20)

17. Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5) [Электронный ресурс]. – Режим доступа: https://arxiv.org/pdf/1311.2524.pdf (дата обращения: 10.02.20)

18. Sudharsan Ravichandiran, Sean, Saito, Rajalingappaa Shanmugamani, Yang Wenzhuo. Python Reinforcement Learning: Solve complex real-world problems by mastering reinforcement learning algorithms using OpenAI Gym and Tensorflow. // Paclt Publishing, 2019

19. Suzuki S, Abe K. Topological Structural Analysis of Digitized Binary Images by Border Following // CVGIP. 1985. Vol. 1. P. 32-46.

20. Tensorflow API Documentation [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org/api_docs/python/tf (дата

обращения:15.02.20)

21. Sick, B. Dynamishe Effekte be in ichtlinearer Wellengleichungenmit Anwendungen in der Verkehrsfkub theorie: Master Thesis. University of Ulm, 1989

22. Rödiger, M. Chaotische Lösungennichtlinearer Wellengleichungenmit Anwendungen in der Verkehrsfkub theorie: Master Thesis. University of Münster, 1990.

23. Fotheringham, A.S. Modelling hierarchical destination choice // Envir. & Plan. A. 1986. V. 18. P. 401 – 418.

24. Olstam, J. J. Comparison of Car-following models / J. J. Olstam, A. Tapani // Swedish National Road and Transpost Research Institute. – 2004. – 45 с.

25. Stouffer, S. A. Intervening opportunities: a theory relating mobility and distance // Amer. Sociolog. Rev. 1940. V. 5. P. 845 – 867.

Приложение А

## Исходный код программы

Исходный код программы находиться в открытом доступе на платформе google colab:

https://colab.research.google.com/drive/1Qq-5P_D_thcAUqcJzVc8OAkhtm8n-bNZ#scrollTo=L61W6J5B4TkC