|  |
| --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования  «Национальный исследовательский университет  «Высшая школа экономики»  *Факультет социально-экономических и компьютерных наук* |
| Берсенёв Илья Иванович  **Разработка системы диагностики артроза по рентген снимкам коленного сустава**  *Курсовой проект*  студента образовательной программы «Программная инженерия» по направлению подготовки *09.03.04 Программная инженерия*  Руководитель,  приглашенный преподаватель  М.Д. Чистогов |

Пермь, 2024 год

**Оглавление**

[Оглавление 2](#_Toc130676825)

[Введение 3](#_Toc130676826)

[1 Анализ предметной области 4](#_Toc130676827)

[1.1 Анализ назначения программы 5](#_Toc130676828)

[1.2. Обзор существующих решений 6](#_Toc130676829)

[1.3. Анализ средств 7](#_Toc130676830)

[1.3.1. Сервисы для геоданных 7](#_Toc130676831)

[1.3.2. Графические сервисы 8](#_Toc130676832)

[1.4. Требования к проекту 9](#_Toc130676833)

[1.5. Диаграмма прецедентов 10](#_Toc130676834)

[2 Проектирование 11](#_Toc130676835)

[2.1 Общая информация 11](#_Toc130676836)

[2.2 Проектирование интерфейса пользователя 12](#_Toc130676837)

[2.2.1 Экран карты 13](#_Toc130676838)

[2.2.2. Экран флагов 14](#_Toc130676839)

[2.2.3 Экран формы 15](#_Toc130676840)

[2.2.4. Общий экран 17](#_Toc130676841)

[2.3 Проектирование данных 19](#_Toc130676842)

[2.3.1. Проектирование базы данных 19](#_Toc130676843)

[2.3.2. Проектирование запросов к базе данных 21](#_Toc130676844)

[2.3.3. Результаты проектирования 22](#_Toc130676845)

[3 Реализация информационной системы 24](#_Toc130676846)

[3.1. Архитектура 25](#_Toc130676847)

[3.1.1. Модуль utils.py 25](#_Toc130676848)

[3.1.2. Модуль app.py 26](#_Toc130676849)

[3.1.3. Шаблон test.html 27](#_Toc130676850)

[3.1.3.1. Создание слоя из переданного JSON объекта 27](#_Toc130676851)

[3.1.3.2. Создание слоя из переданного JSON объекта 28](#_Toc130676852)

[3.1.3.3. Создание формы 28](#_Toc130676853)

[3.2. Финальный вид приложения 30](#_Toc130676854)

[Заключение 31](#_Toc130676855)

[Список литературы 32](#_Toc130676856)

# Введение

В наше время наблюдается стремительный рост технологий с применением искусственного интеллекта, под которым часто подразумевают нейронные сети. Во многом это обусловлено тем, что нейронные сети, будучи симуляцией нейронов органического мозга, позволяют решать нетривиальные задачи, неподвластные человеку, например постановка диагноза по рентген снимку, предсказание курса акций на бирже, генерация изображений и т.д.

Ввиду востребованности данной технологии была выбрана тема «разработка системы диагностики артроза по рентген снимкам коленного сустава».

# Анализ предметной области

в данной главе осуществляется анализ предметной области, перечисляются существующие решения, а также ведётся обзор технологий для реализации сервиса.

**1.1 Анализ назначения программы**

В наше время нейросетевые технологии используются во множестве различных сфер, однако по большей части они решают задачи бизнеса, а разработок в области медицины на порядок меньше.

Сервис будет представлять из себя систему, которую можно будет интегрировать в систему ЕРИС, способную распознавать степень артроза сустава по его рентген снимку, однако хотелось бы подчеркнуть, что на первом месте стоит именно функциональность сервиса, а на втором – его способность к интеграции.

Сервис будет реализовывать единственную функцию:

1. Загрузка рентген снимка сустава и мгновенный ответ с вероятностями степеней артроза в соответствии с требованиями ЕРИС.

**1.2. Обзор существующих решений**

На момент написания курсовой работы не было выявлено аналогов данной работы. Существуют предобученные нейронные сети, но они имеют определённые недостатки: несовместимы с ЕРИС, не имеют возможности подсветки патологии либо имеют слишком низкую точность.

**1.3. Анализ средств**

Сервис для распознавания степени артроза будет иметь в себе следующие компоненты:

* Ядро, представляющее из себя нейросеть;
* Веб-сервис, служащий обёрткой над нейросетью;
* Брокер сообщений для интеграции с ЕРИС.

Были поставлены следующие задачи:

1. Изучить сервисы/технологии для разработки нейросетей;
2. Изучить подходы к разработке нейросетей;
3. Изучить сервисы, предоставляющие ресурсы для обучения нейросетей;
4. Выбрать источник данных для тренировки;
5. Выбрать веб-сервис для обертки над нейросетью;
6. Выбрать брокер сообщений;
7. Сформировать требования к продукту;
8. Спроектировать приложение.

**1.3.1. Сервисы для разработки нейросетей**

В данный момент в сфере нейросетей присутствуют три лидера: TensorFlow, PyTorch, Keras и XGBoost. После непродолжительного анализа был выбран фреймворк Keras ввиду того, что в нём присутствует возможность составлять нейросети высокоуровневыми блоками, что довольно удобно.

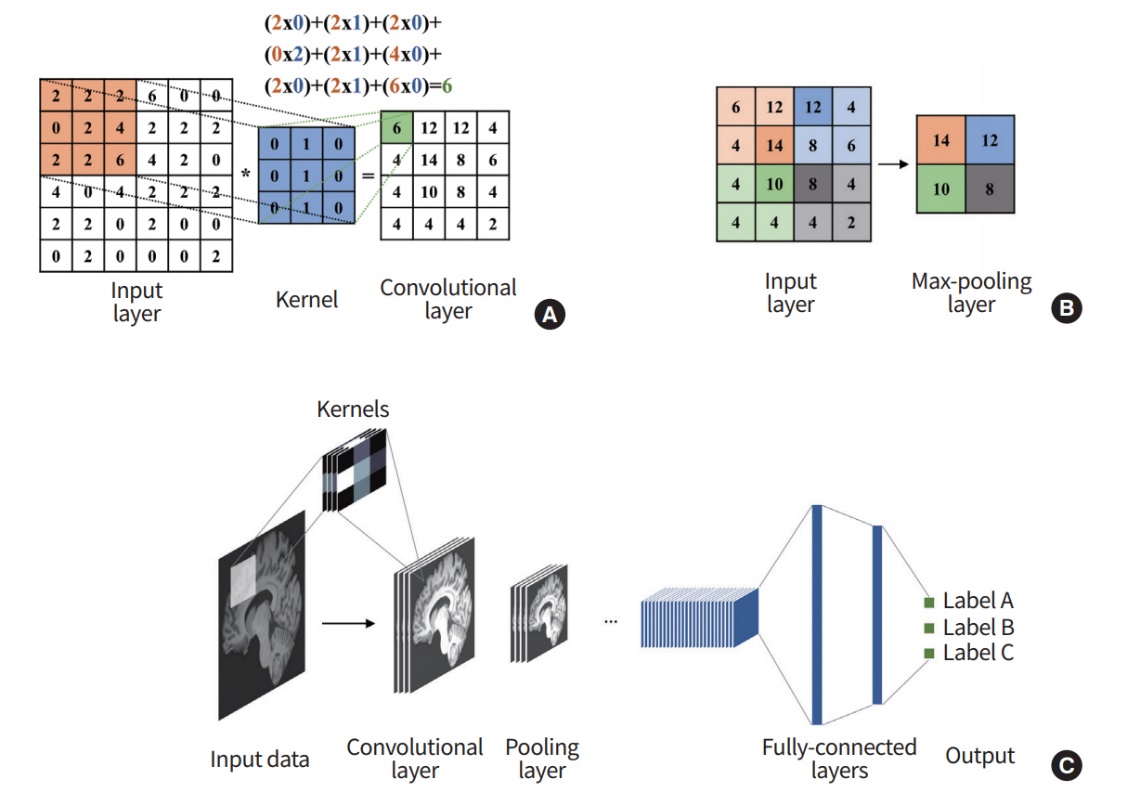
**1.3.2. Подходы к разработке нейросетей**

В процессе самообучения было выделено несколько различных подходов к разработке нейросетей, а именно:

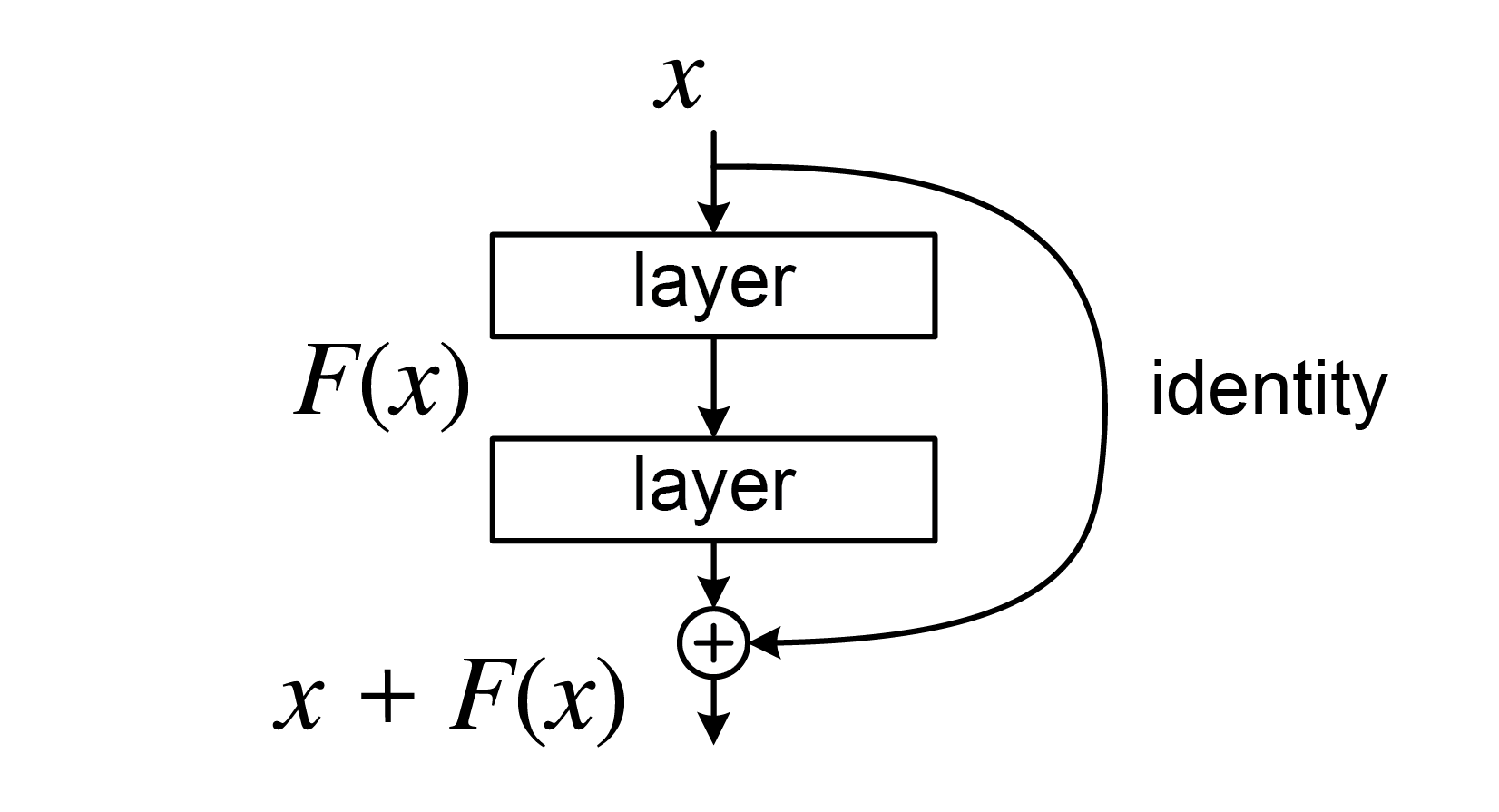
1. Свёрточные нейронные сети (convolutional neural networks);
2. Остаточные нейронные сети (residual neural networks);
3. Рекуррентные нейронные сети (recurrent neural networks).

Рассмотрим каждый из них поподробнее.

Свёрточные нейронные сети представляют из себя архитектуру нейронных сетей, при которой на вход нейронной сети подаётся определённое количество сигналов, которые сеть преобразует в некоторое количество feature maps. Feature map можно охарактеризовать как исходные данные с наложенным «фильтром». На рис. 1 сверху слева вы можете видеть вычисление feature map из исходных данных и сверточного ядра. Как мы можем видеть, это обыкновенное скалярное произведение входных данных со сверточным ядром с тем лишь нюансом, что скалярное вычисление вычисляется для каждой области входных данных. Это значит, что если мы условно обозначим на картинке первую область как [0, 2;0, 2], где 0,2 это координаты начала и конца области по вертикали и горизонтали соответственно, то также будут вычислены [0, 2; 1, 3], [0, 2; 2, 4], [0, 2; 3, 5], [1, 3; 0, 2] и так далее. Данный вид архитектуры высокоэффективен для нахождения повторяющихся закономерностей.



Остаточные нейронные сети являются типом нейросетей, в которых имеются особые остаточные блоки, в которых могут быть, например, линейные или сверточные слои, которые так же обращаются ко входу остаточного блока. Происходит это посредством суммирования обработанного каким-либо слоём и необработанного входов блока. Вы можете увидеть примерную архитектуру такого блока на рисунке 2. Данный вид архитектуры эффективен тем, что теряется меньше информации в процессе преобразования данных, чем в линейной или сверточной нейросетях.



Рекуррентные нейронные сети – это вид нейронных сетей, имеющих долгую краткосрочную память (LSTM). Чаще всего эти нейросети используются в языковых моделях, так что разбирать их не имеет особого смысла.

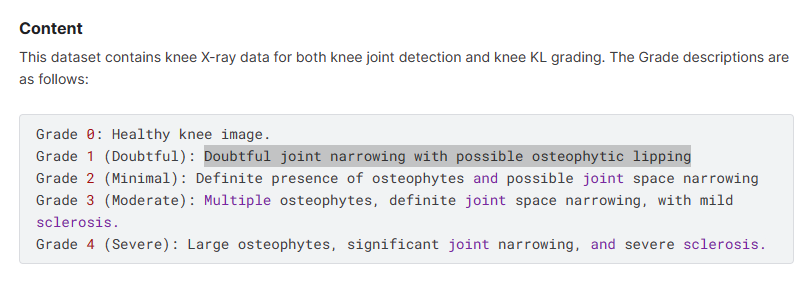
Таким образом, для разработки были выбраны 2 основных архитектуры: свёрточные нейронные сети и остаточные нейронные сети.

**1.3.3. Сервисы для обучения нейросети**

Ввиду того, что обучение нейросети – довольно сложная задача в смысле компьютерных вычислений, необходимо было выбрать бесплатный сервис для обучения. Наиболее популярные – Google Colab, Kaggle, Microsoft Azure Notebook. Ввиду того, что из-за санкций последний вариант недоступен, пришлось пользоваться Colab и Kaggle. Colab довольно удобен с точки зрения разработки, поэтому изначально работа велась в нём, однако Kaggle предоставляет бóльшие ресурсы для тренировки нейросетей, потому дальнейшая разработка велась в Kaggle.

**1.3.4. Сервисы для обучения нейросети**

В качестве источника данных был выбран датасет на сайте Kaggle под названием [Knee Osteoarthritis Dataset with KL Grading - 2018](https://www.kaggle.com/datasets/tommyngx/kneeoa), представляющий из себя рентген снимки коленей (1 колено на 1 снимке) в формате 224 на 224 пикселя. Данные размечены на 5 классов: 1 – здоровое колено, 2 – колено с возможной патологией, 3 – колено с подтверждённой патологией первой степени, 4 – колено с множественными подтверждёнными патологиями средней тяжести, 5 – колено с множественными патологиями высокой тяжести. Вы можете увидеть легенду на рисунке 3.



**1.3.5. Сервис-обёртка над нейросетью**

Для совместимости с ЕРИС необходимо, чтобы сервис был размещён в интернете, а для этого нужен веб-фреймворк. В качестве такого был выбран FastApi ввиду того, что это активно развивающийся и легкий фреймворк.

**1.3.6. Брокер сообщений**

Для совместимости с ЕРИС также необходим брокер сообщений Kafka. Именно этот брокер был выбран для системы, для взаимодействия с ним был выбран пакет aiokafka.

**1.4. Требования к проекту**

Были сформированы следующие требования к проекту:

1. Проект должен распознавать степень артроза сустава;
2. Проект должен возвращать области, в которых предположительно локализованы патологии;
3. Проект должен быть совместим с ЕРИС.

# Проектирование

в данной главе будет производиться проектирование самого сервиса и его пользовательского интерфейса.

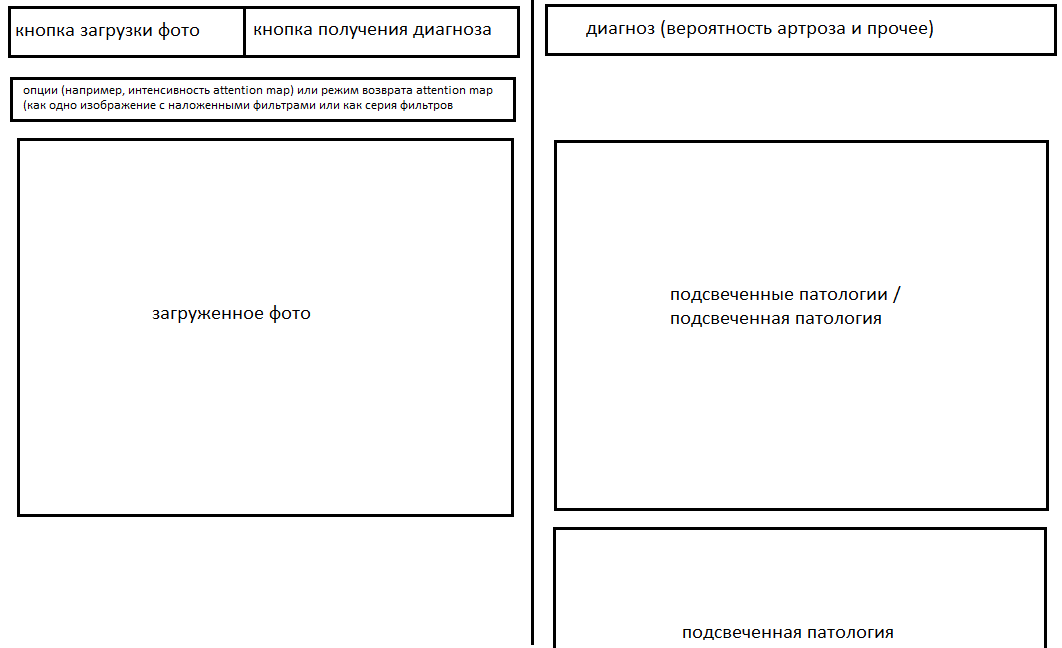
**2.1 Общая информация**

Как уже было упомянуто выше, для сервиса будут использоваться keras, fastapi, kafka. Эти средства позволяют писать емкий читаемый код, и цель – написать не только рабочий прототип сервиса, но ещё и сделать его исходный код легко читаемым и поддерживаемым.

**2.2 Проектирование интерфейса пользователя**

Перед началом работы было необходимо сверстать или придумать пользовательский интерфейс, который затем обретёт программа, являющаяся продуктом курсовой деятельности. Было принято решение рисовать эскизы в графическом редакторе под названием «Microsoft Paint», ввиду его лёгкости и быстроты.

В конечном продукте, результате курсовой работы будет всего лишь один экран с двумя секциями. Проект должен выполнять лишь одну функцию, классифицировать рентген снимки, так что нет нужды верстать усложнённый интерфейс с большим количеством различных функций. Ниже – примерный эскиз будущего интерфейса.

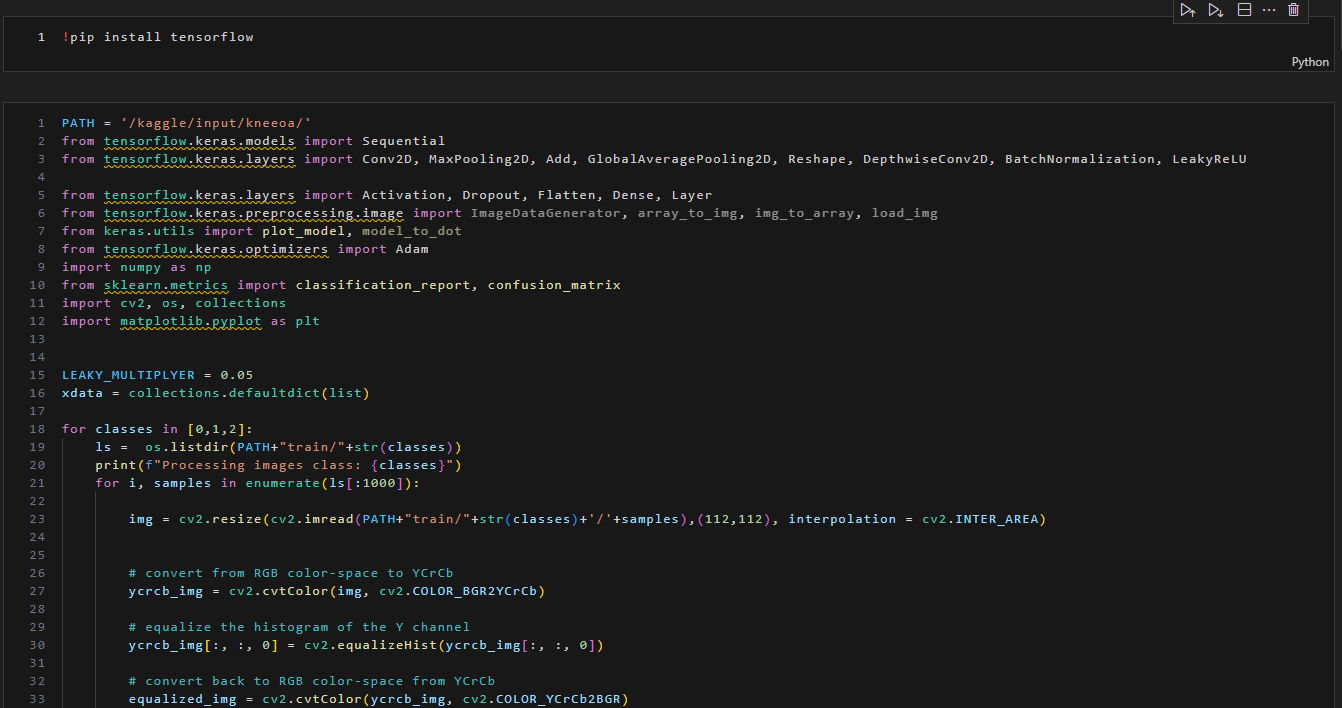


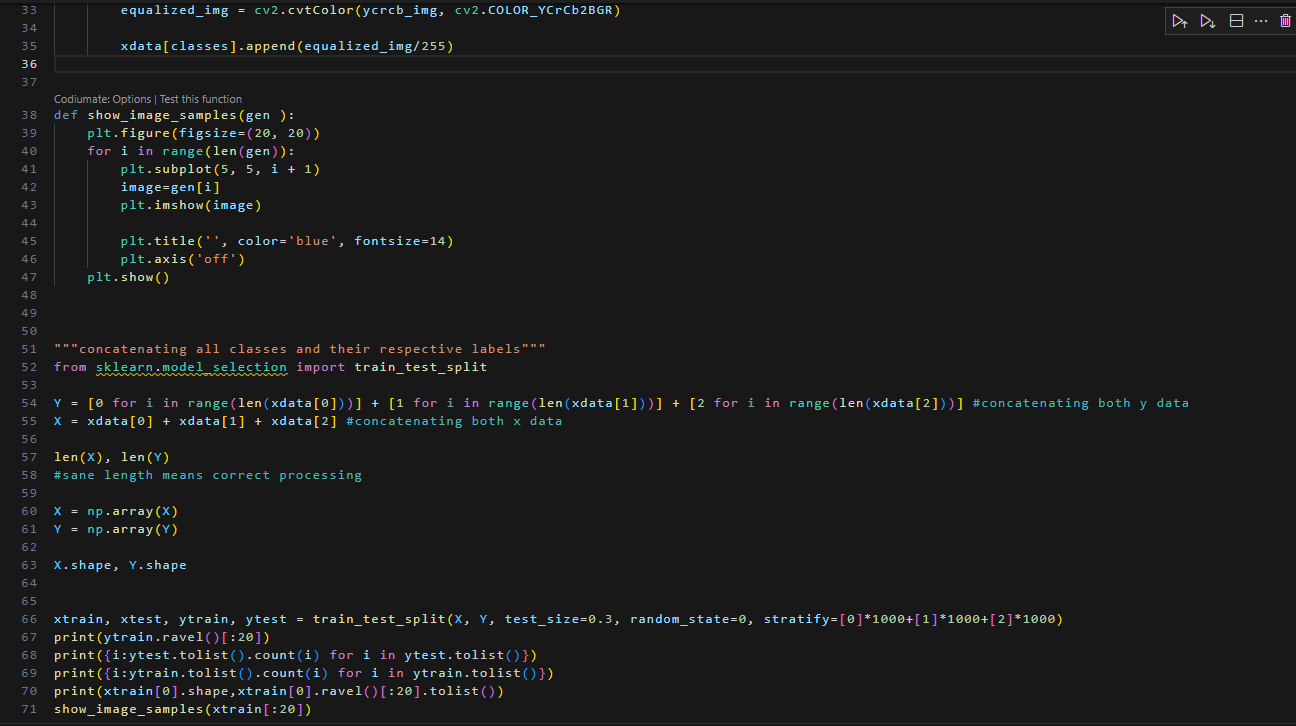
# Реализация информационной системы

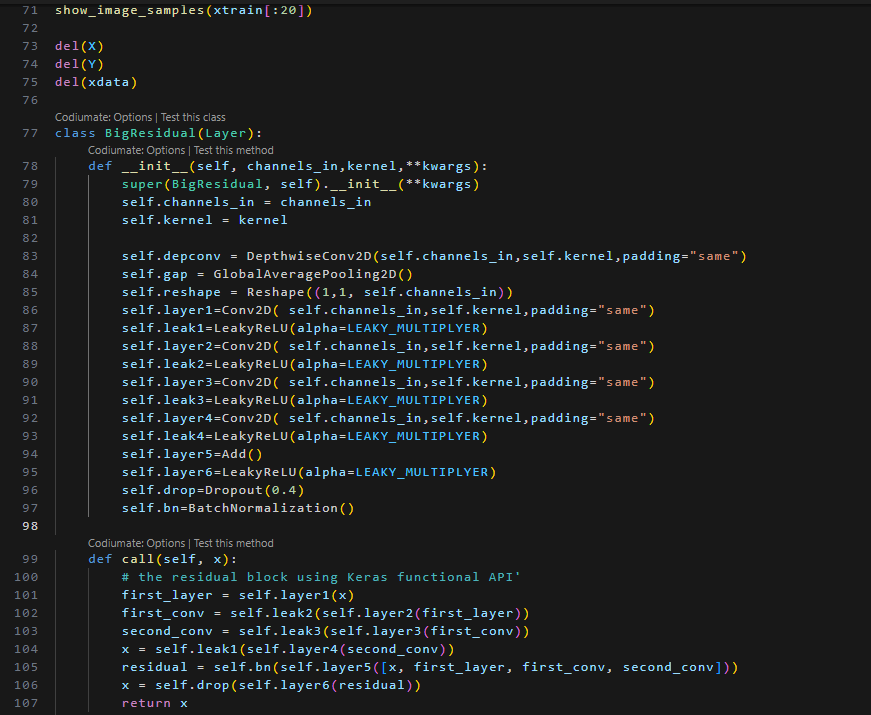
В данной главе будет описано получившееся приложение, а также его архитектура.

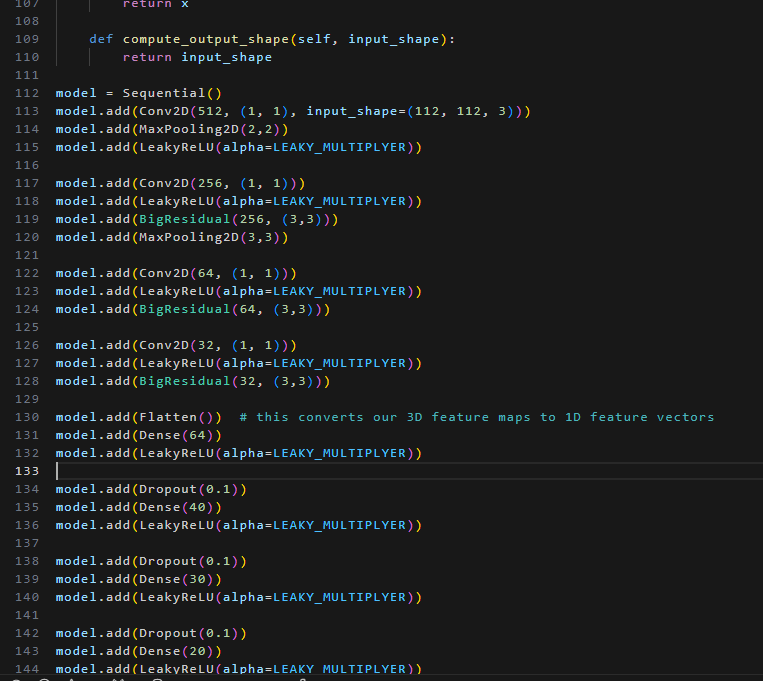
**3.1. Архитектура нейронной сети**

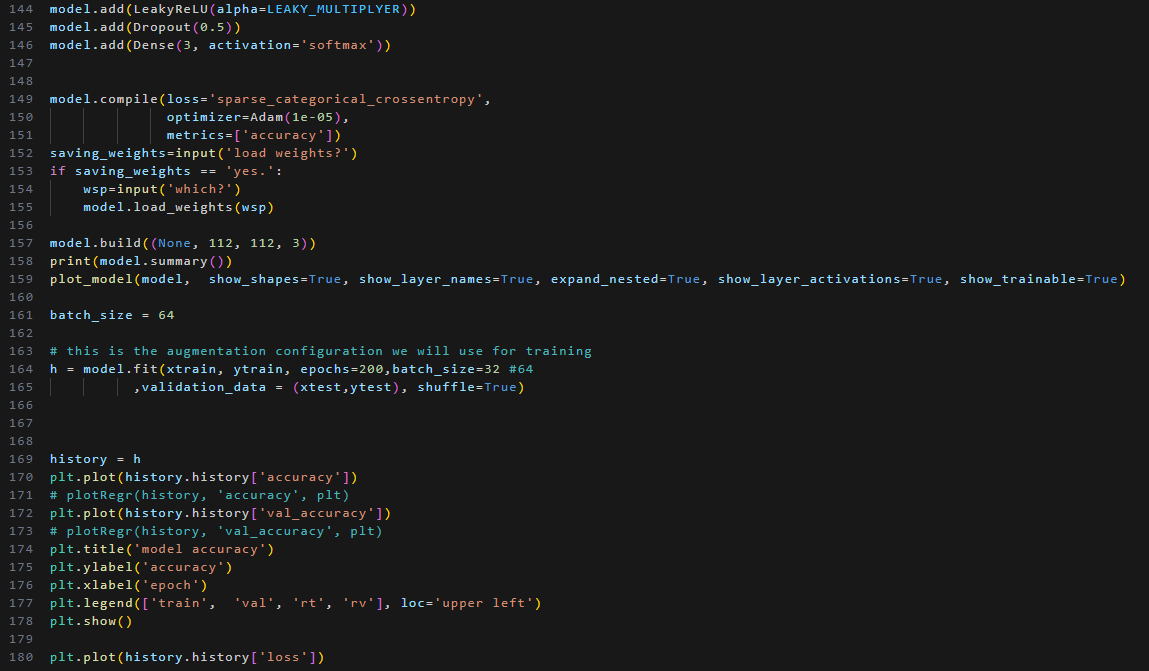
Для того, чтобы понять архитектуру получившейся нейронной сети, необходимо сначала рассмотреть блокнот, в котором велась разработка.

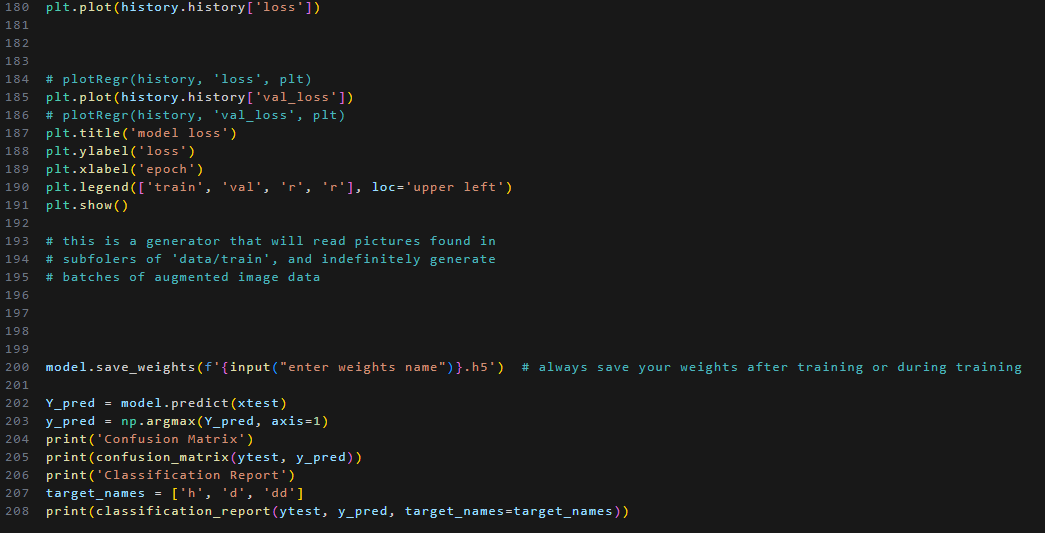








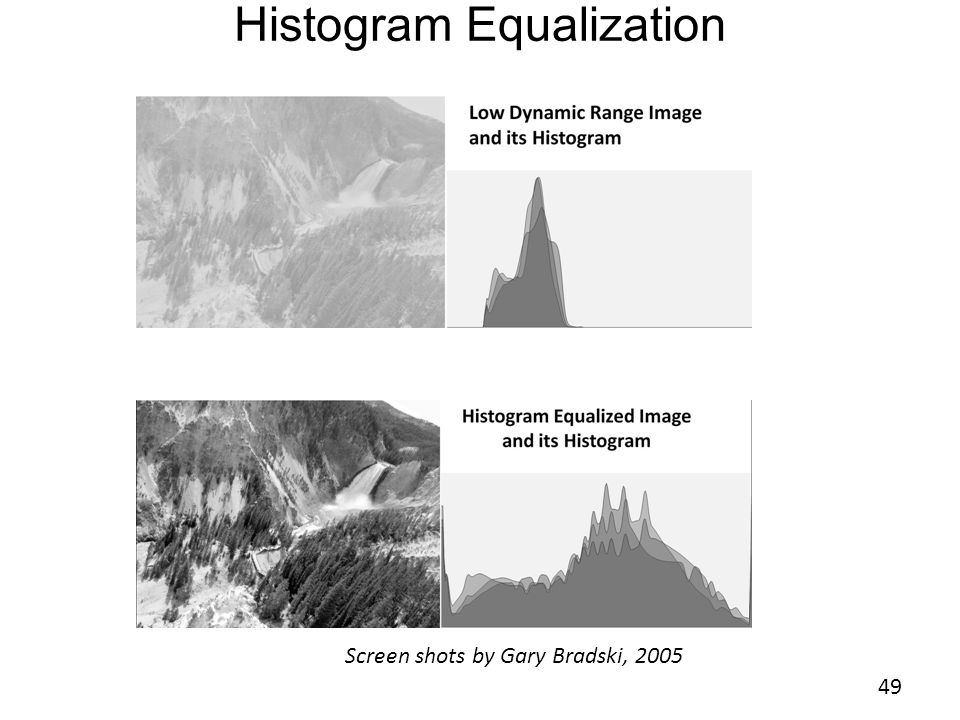




В данном блокноте можно выделить три основных блока: загрузка и препроцессинг (1-75), задание нейросетевой структуры (77-155), обучение (164-208). Рассмотрим каждый поподробнее.

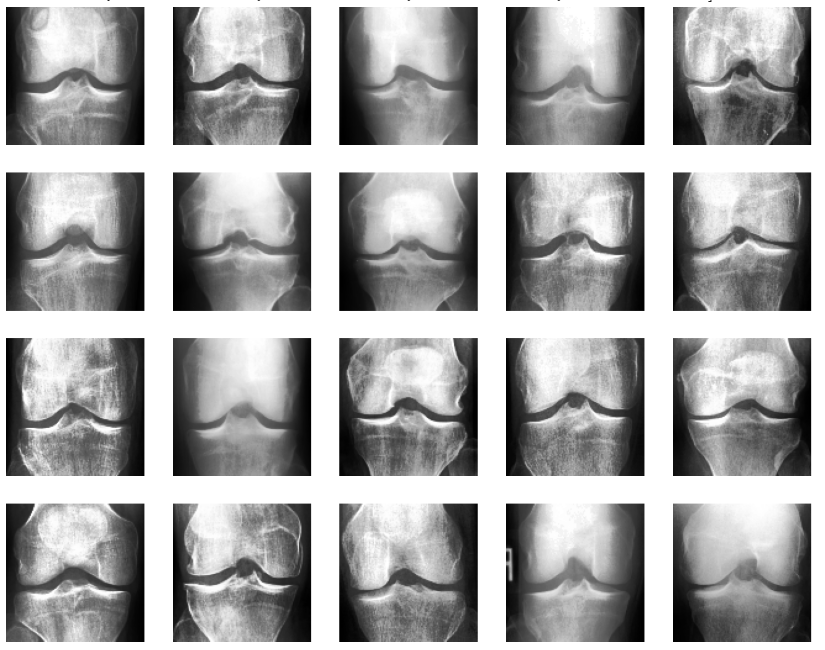
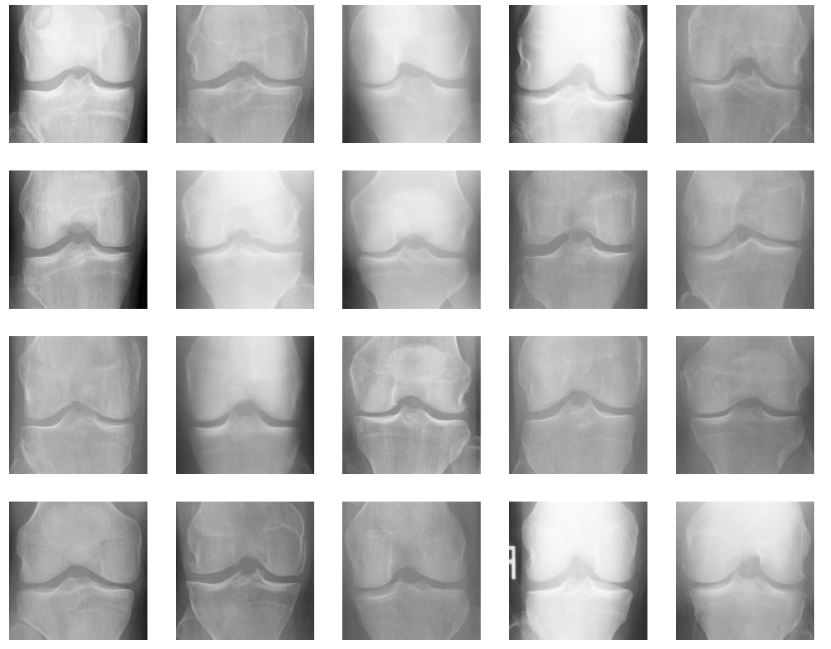
**3.1.1. Препроцессинг**

В строке 23 мы можем увидеть изменение размерности изображения до 112\*112. Это нужно для того, чтобы уменьшить размерность сети с 224\*224 для ускорения обучения. Также в выбранном датасете довольно много снимков, и они не были нормализованы. У какого-то снимка был засвет, у какого-то – наборот, слишком тёмная цветовая гамма, также были и другие отклонения вроде стальных болтов в коленях, случайно попавших на снимки букв и т.п., но было решено исправлять только цветовые отклонения. Для этого был выбран метод эквализации гистограмм из пакета Opencv. На рисунке (ПОПРАВИТЬ) можно увидеть принцип работы этого метода, его суть заключается в том, чтобы равномерно распределить распределение значений во всех каналах.



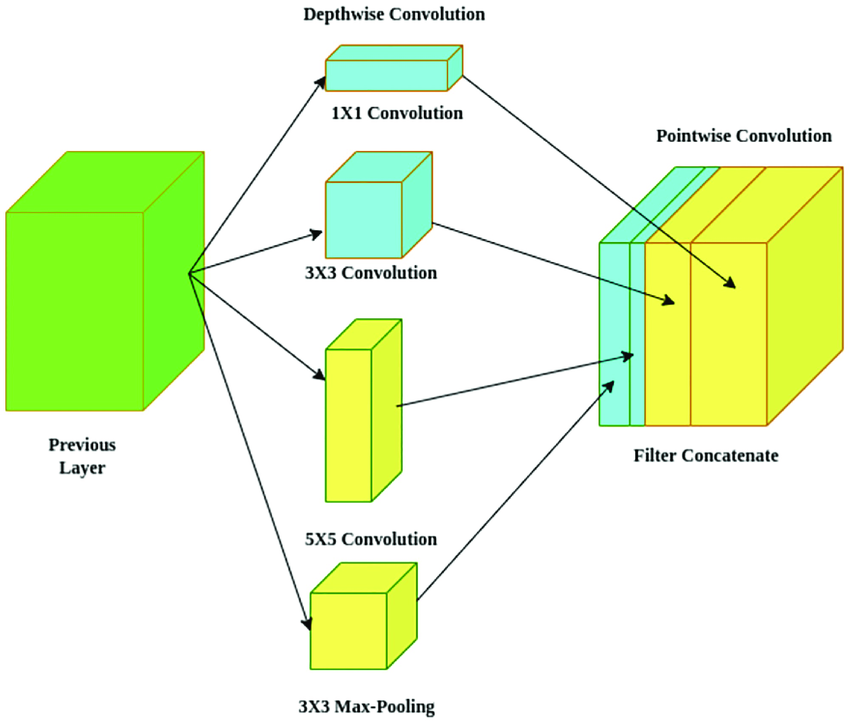
Вы можете увидеть реализацию данного метода в строках с 23 по 35. Очень важно корректно осуществить препроцессинг и убедиться что данные принимают нужный вид, потому что в процессе разработки довольно долго существовала проблема того, что по непонятным причинам нейросеть выводила лишь одно значение. Оказалось, что это – следствие того, что при препроцессинге значения принимали вид [0..255], а не [0..1], вследствие чего при обратном проходе к весам применялись слишком высокие значения и нейросеть не имела возможности обучиться.

Пример препроцессинга вы можете видеть ниже, рисунок (ПОПРАВИТЬ) – серия изображений до препроцессинга, рисунок (ПОПРАВИТЬ) – после. На обоих снимках одни и те же фотографии.



**3.1.2. Нейросетевая структура**

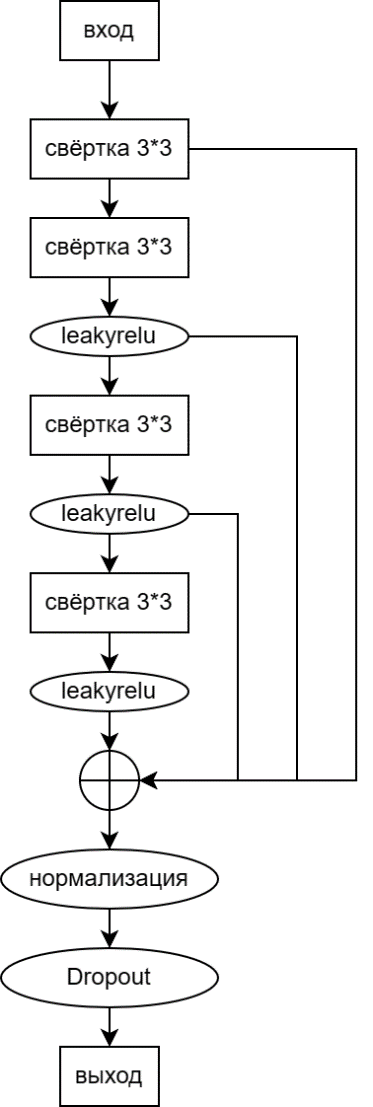
Тренировка нейросети и выбор подходящей для неё структуры – самый сложный и трудоёмкий этап во всей курсовой работе. Ранее уже говорилось о том, что будут использованы свёрточные и остаточные слои. Их можно увидеть в кастомном блоке под названием BigResidual в строке 77. Данный блок – очень важная архитектурная часть сети. Он вдохновлён Xception блоком в одноимённой неросети, разработанной компанией Google. Вы можете видеть архитектуру блока Xception на рисунке ПОПРАВИТЬ.



Архитектура блока BigResidual похожа, но всё же имеет существенные различия. Так, данный блок представляет из себя остаточный блок, имеющий следующую структуру:

1. На вход подаются значения;
2. Эти значения обрабатываются свёрткой 3\*3, не меняя размерность;
3. Значения из пункта 2 активируются функцией LeakyReLu и к ним снова применяется свёртка 3\*3;
4. Значения из пункта 3 активируются функцией LeakyReLu и к ним снова применяется свёртка 3\*3;
5. Значения из пункта 4 активируются функцией LeakyReLu и к ним снова применяется свёртка 3\*3;
6. Складываются и нормализуются значения из пунктов 2, 3, 4, 5, затем к ним применяется функция активации и Dropout с значением 0.4.

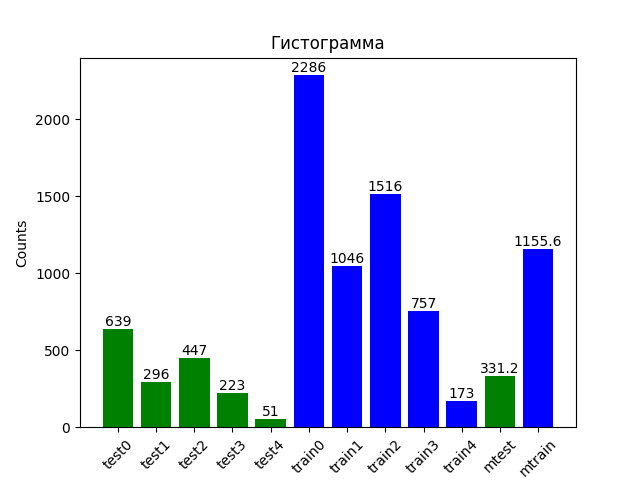
Вы могли заметить, что в блоке используется leakyrelu, а не обычный ReLu. Это было сделано сознательно для того, чтобы нейроны нейросети, случайно отключённые в процессе обучения, могли восстановиться через большое количество итераций. Также в блоке используется Dropout, случайно выключающий 40% нейронов в процессе обучения чтобы избежать переобучения. Схему данного блока вы можете увидеть ниже.



В нейросети первый такой блок выступает автоэнкодером. Перейдём к обзору архитектуры сети в целом.

Сеть реализована с помощью Sequential класса, а значит, представляет из себя последовательность блоков. Для уменьшения размерности в ней используются слои MaxPooling, а для увеличения – свёртки 1\*1. Мы можем видеть, что форма ввода для нейросети – 112\*112\*3, это значит, что нейросеть принимает изображение 112\*112 пикселей в трёх каналах. Далее они проходят через три остаточных блока, а после этого найденные features прогоняются через линейные слои и на выходе мы имеем 3 вероятности с активацией softmax, что соответствует вероятности принадлежности к одному из классов. Полную архитектуру модели можно увидеть на рисунке ПОПРАВИТЬ.

Вероятностей принадлежности к одному из классов всего 3 из-за того, что данных очень мало, на рисунке ниже – результат выполнения анализа по количеству классов в тренировочных и тестовых значениях.



Как мы можем видеть, в среднем в тренировочных данных примерно 1156, больше всего данных – в первом классе (здоровые), и меньше всего – в последнем (очень болные). Так что сначала было принято решение классифицировать первые 3 класса, однако в дальнейшем было обнаружено, что 2 класс не поддаётся обнаружению (скорее всего он почти не отличается от первого или отличается незначительно) и было принято решение классифицировать первый, третий и четвертый классы.

Модель компилируется с помощью критерия потерь loss\_sparse\_categorical\_crossentropy и оптимизатора Адам, по метрике accuracy. Выбор критерия потерь обусловлен тем, что это лучший из возможных критериев, так как он наиболее точно вычисляет потери для данной задачи классификации. Оптимизатор был выбран случайным образом, а метрика accuracy почти всегда используется при обучении.

**3.1.2. Нейросетевая структура**

Тренировка нейросети и выбор подходящей для неё структуры – самый сложный и

**3.2. Архитектура проекта**

На финальной стадии разработки проект имеет следующую структуру:

* Jupyter блокнот notebook709e7d8392\_3 – блокнот, в котором велась разработка и обучение нейронной сети. По сути, это – самый главный файл во всем проекте, но для работы нейросети он не нужен;
* Модуль backend – модуль, отвечающий за обработку запросов, представляет из себя веб-сервер, написанный на fastapi, kafka;
* Модуль analyze – модуль, нужный для исследования данных. Не обязателен для использования в работе;
* Модуль debug – модуль, нужный для подсветки патологий. Данный модуль принимает на вход модель и изображение, поданное на вход и возвращает его в обработанном виде.
* Модуль model – самый главный модуль. Он загружает веса предобученной нейронной сети для классификации и возвращает модель, пригодную для использования.
* Модуль preprocess – модуль, представляющий из себя предобработчик рентген снимков. Он использовался для обучения нейронной сети, так что для корректной работы его лучше оставить.
* Файл 65-relative-success-v3.weights.h5 – веса предобученной нейронной сети.

Рассмотрим каждый модуль подробнее.

**3.1.1. Jupyter блокнот**

Блокнот notebook709e7d8392\_3, как уже было упомянуто выше, представляет из себя файл, в котором велась разработка нейросети.

**3.1.2. Модуль app.py**

На каждое действие (маркировка, редактирование, добавление клиники) был создан роут (mark, post, create соответственно). Их описание нецелесообразно по причине низкой информативности, поэтому будет описана только главная часть модуля. Главной частью является функция main, в которой происходят следующие действия:

1. в переменную content загружаются геоданные всех клиник;
2. в переменную content2 загружаются геоданные всех маркеров;
3. далее данные фильтруются по специальности, если это необходимо;
4. данные переводятся в json и передаются в html шаблон.

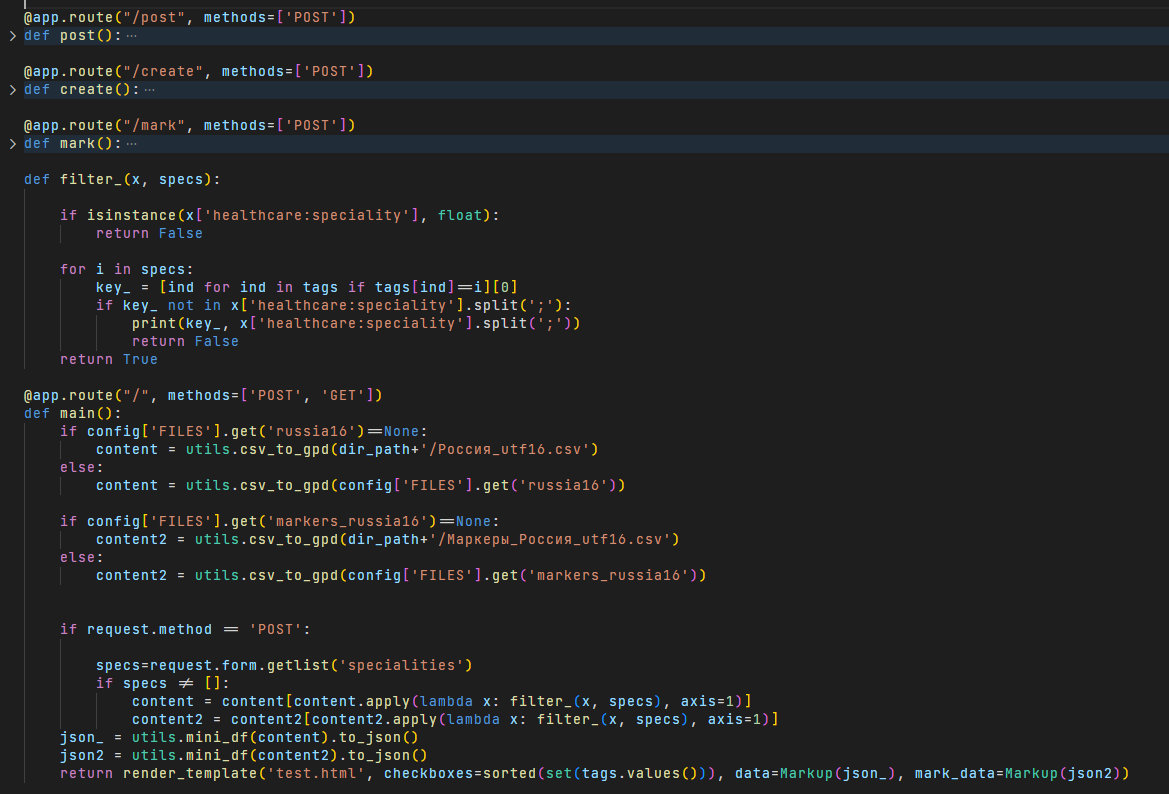


Рисунок 12 – модуль app

**3.1.3. Шаблон test.html**

Шаблон test.html имеет небольшую DOM- структуру, однако длинный JS-код, поэтому будут показаны лишь самые важные его участки, такие как:

1. Создание слоя из переданного JSON объекта
2. Создание самой карты с геоданными
3. Создание формы для редактирования элемента

**3.1.3.1. Создание слоя из переданного JSON объекта**

Как уже было вышеупомянуто, в шаблон test передаются геоданные из App.py. Как мы знаем, шаблонизатор jinja воспринимает переменную, заключенные в две фигурные скобки, как переданную переменную. В данном участке кода сначала создаётся простой рендерер для маркеров, затем в константу передаётся json, после этого он переводится в двоичный вид и после этого из него создаётся слой. Аналогично создаётся и слой для обычных объектов, с той лишь разницей, что в слое для обычных объектов более сложный рендерер.

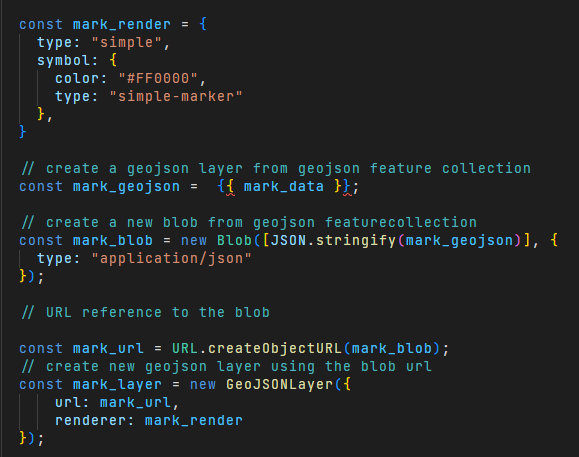


Рисунок 13 – создание слоя

**3.1.3.2. Создание слоя из переданного JSON объекта**

На данном участке кода мы видим, как сначала создаётся карта из двух слоёв, а затем она добавляется в вид. Добавление в вид – непосредственно размещение карты в HTML. Слои layer и mark\_layer – тепловая и маркерная карты клиник соответственно.

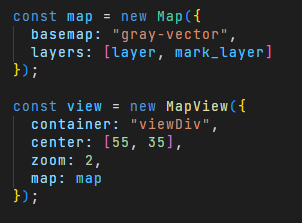


Рисунок 14 – создание карты

**3.1.3.3. Создание формы**

Для того, чтобы иметь возможность редактировать информацию о клиниках, сначала была предпринята попытка изменить под себя встроенное в фреймворк всплывающее окно, однако эта попытка окончилась провалом, по итогу которого было принято решение создать в уведомлении кнопку, которая бы просто создавала форму для изменения клиники. Код приведён ниже.

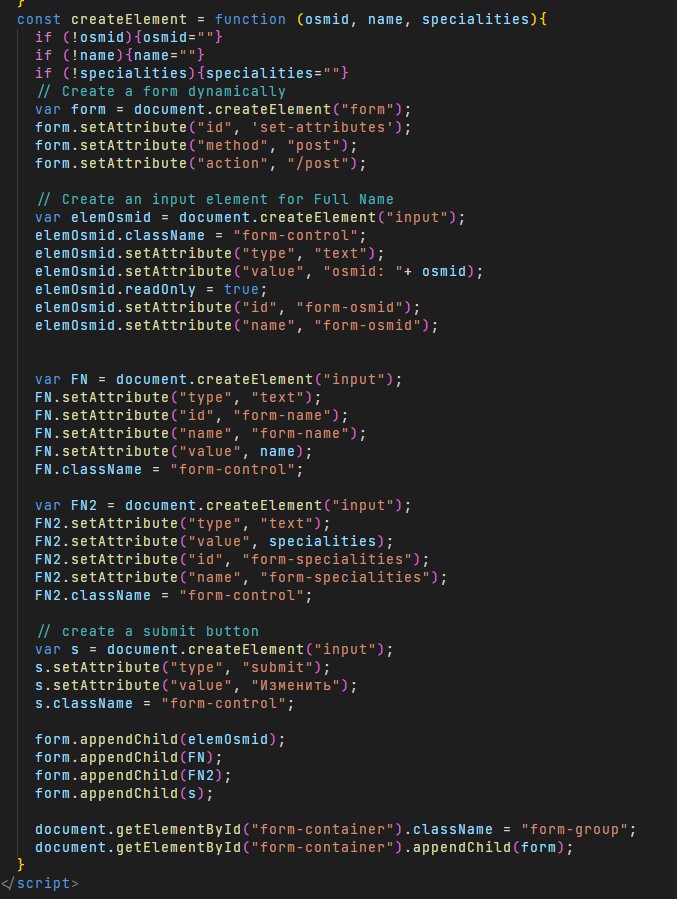


Рисунок 15 – создание формы

**3.2. Финальный вид приложения**

По итогу работы сервис принял следующий вид:

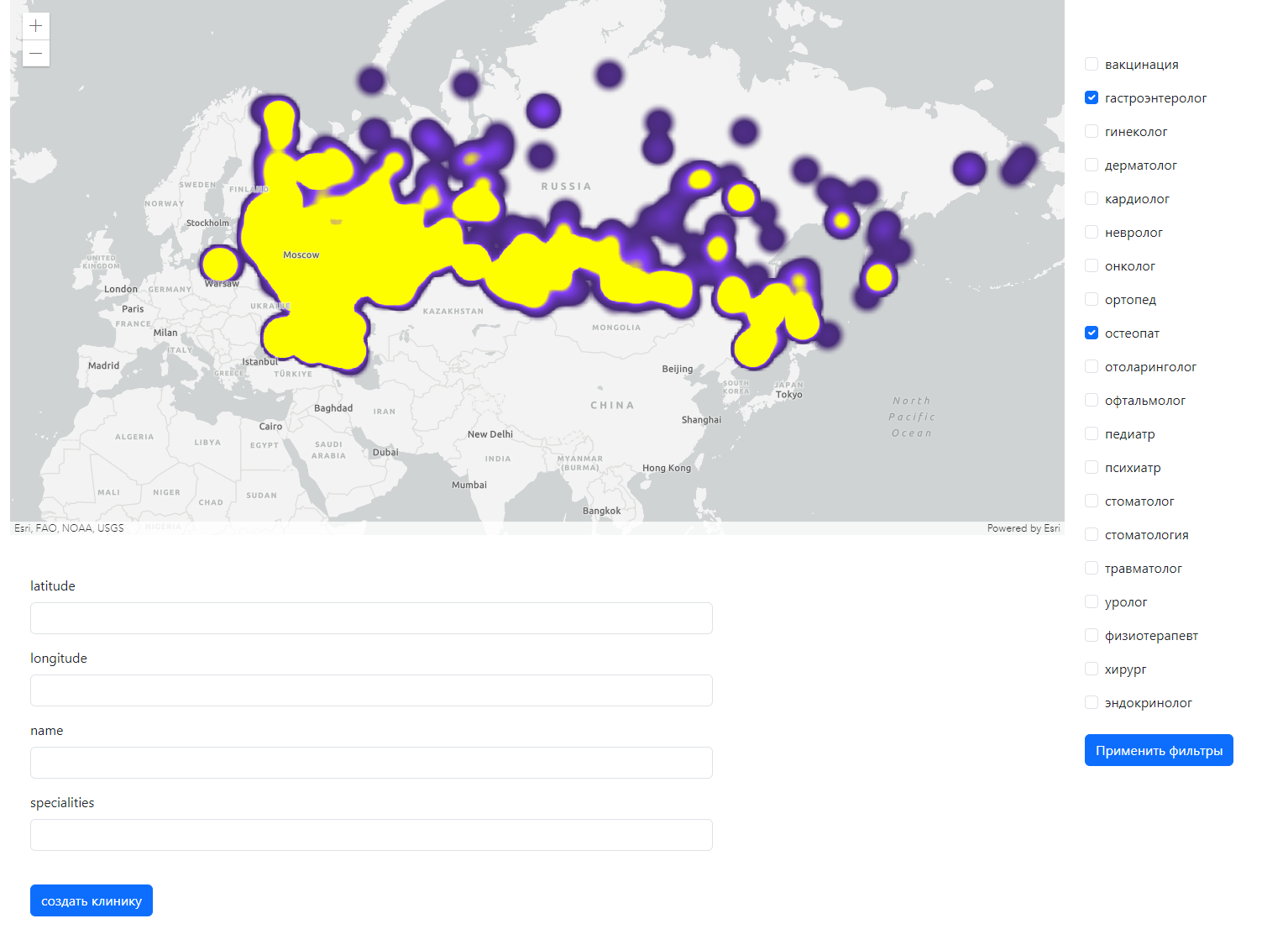


Рисунок 16 – итоговый вид

**Заключение**

В результате курсовой работы были получены навыки работы с геоданными и JS-фреймворками. Также были закреплены навыки работы с Bootstrap.

# Список литературы

1. Официальная документация к фреймворку ArcGis– [Электронный ресурс]. – URL: https://developers.arcgis.com/javascript/latest/api-reference/
2. Официальная документация к библиотеке osmnx– [Электронный ресурс]. – URL: https://osmnx.readthedocs.io/
3. Официальная документация к фреймворку Flask 2.2– [Электронный ресурс]. – URL: https://flask.palletsprojects.com/en/2.2.x/