

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Пермский государственный национальный  
исследовательский университет»

Кафедра математического  
обеспечения вычислительных систем

УДК 004.93'14

**ИССЛЕДОВАНИЕ ИЗОБРАЖЕНИЙ ПЛЕВРАЛЬНЫХ  
ВЫПОТОВ ДЛЯ РАННЕЙ ДИАГНОСТИКИ ЗАБОЛЕВАНИЙ**

Выпускная квалификационная работа бакалавра

Работу выполнил студент  
ПМИ-1 группы 4-го курса  
механико-математического  
факультета  
\_\_\_\_\_Заманов М.Р.

Научный руководитель:  
к.ф.-м.н., доцент, доцент  
\_\_\_\_\_Замятина Е.Б.  
“ \_\_\_\_ ” \_\_\_\_\_ 2019 г.

Пермь 2019

## **Аннотация**

Тема научно-исследовательской работы «Исследование изображений плевральных выпотов для ранней диагностики заболеваний». В рамках данной работы были реализованы программные продукты для обучения нейронной сети и распознавания онкологических заболеваний и постановки соответствующих диагнозов по микроскопическому изображению плевральных выпотов больного человека.

Текст работы состоит из введения, заключения, 3 глав и 1 приложения.

Первая глава включает в себя знакомство с диагностикой заболеваний, анализ существующих систем, теоретический обзор сверточных нейронных сетей, библиотек Keras и OpenCV, знакомство с компьютерным зрением.

Во второй главе речь идёт о проектировании компонентов системы, выборе языков программирования и программных средств.

Третья же глава включает в себя проектирование архитектуры сверточной нейронной сети, предварительную обработку изображений, обучение сети и тестирование, реализацию пользовательского интерфейса и обзор результатов работы программы.

В приложении размещен исходный код и ссылка на репозиторий Git.

Работа изложена на 54 страницах, включает в себя 14 рисунков и 2 таблицы.

## Оглавление

Введение.....	5
Глава 1. Теоретический обзор.....	7
1.1    Диагностика заболеваний медицинскими работниками.....	7
1.2    Анализ существующих систем .....	8
1.3    Сверточные нейронные сети.....	10
1.3.1    Сверточный слой (convolutional).....	12
1.3.2    Слой субдискретизации (subsampling).....	12
1.3.3    Полносвязный слой (fully-connected).....	13
1.4    Библиотека Keras.....	14
1.5    Компьютерное зрение.....	17
1.6    Библиотека OpenCV.....	18
1.7    Обработка изображений .....	21
1.8    Выводы по главе.....	21
Глава 2. Проектирование системы распознавания .....	22
2.1    Проектирование требований к системе .....	22
2.2    Проектирование архитектуры системы .....	23
2.3    Выбор языка программирования .....	23
2.4    Выбор программных средств.....	24
2.5    Выводы по главе.....	25
Глава 3. Разработка системы и обзор результатов .....	26
3.1    Предварительная обработка изображений .....	26
3.1.1    Устранение шумовых помех.....	26
3.1.2    Бинаризация.....	28
3.1.3    Повороты изображений.....	31

3.2	Проектирование архитектуры сверточной нейронной сети .....	32
3.3	Обучение сети и сравнение результатов архитектур .....	37
3.4	Реализация пользовательского интерфейса .....	38
3.5	Тестирование программной системы.....	39
3.6	Выводы по главе.....	40
Заключение .....		41
Список сокращений и условных обозначений.....		42
Список использованных источников .....		43
Приложение. Код программы .....		46

## Введение

Диагностика разного рода заболеваний является одной из основных составляющих современной медицины, первой ступенью на пути к выздоровлению больного человека. Только после определения заболевания человека, врачи в силах принимать решения и назначать план лечения больного. Выздоровление почти всегда зависит от своевременной и, главное, правильной постановки диагноза. В данной работе перед нами ставится цель определить по снимку плеврального выпота пациента, болен ли человек онкологическим заболеванием.

Данное исследование является актуальным, так как очень важно как можно раньше выявлять онкологические заболевания для своевременного начала курса лечения, поскольку они намного лучше лечатся на начальном этапе, нежели на последующих. Если больному человеку вовремя не помогут обнаружить заболевание, связанное с онкологией, то исход может быть летальным, даже в наше время, когда достаточно серьёзно развиты технологии и медицина в целом, заболевания этой категории очень тяжело лечить, и далеко не всегда врачам это удаётся сделать.

Новизна работы заключается в том, что при её написании использовались сверточные нейронные сети. Реализация была выполнена при помощи возможностей библиотек Keras и OpenCV.

Объектом исследования является ранняя диагностика заболеваний, предметом – распознавание фотографических изображений плевральных выпотов, а инструментом исследования выступают сверточные нейронные сети (convolutional neural network, CNN).

Целью научно-исследовательской работы служит создание программного продукта, который ставит диагноз по микроскопическому изображению плеврального выпота пациента. Необходимо это для того, чтобы медицинские работники могли чуть более точно определять онкологические заболевания. Данная система разрабатывается исключительно в исследовательских интересах, цель её написания не носит коммерческий характер.

Также у медицинских работников появится возможность сравнивать результаты, полученные на основе программного средства, реализованного в рамках данной работы, с результатами программ других людей, возможно, даже студентов, которые занимались исследованием в этой же области, но применяли другой метод распознавания образов, либо в будущем будут заниматься исследованиями в этой области. В итоге будет получена максимальная точность при определении заболеваний людей на основе плевральных выпотов.

Для достижения поставленной цели было необходимо:

1. Реализовать фрагмент программы, который осуществляет предварительную обработку изображений.
2. Разработать архитектуру сверточной нейронной сети.
3. Обучить нейронную сеть с использованием возможностей библиотеки Keras.
4. Осуществить анализ результатов распознавания после обучения.

Для решения поставленных выше задач использованы следующие методы:

1. Методы предварительной обработки изображений (устранение шумовых помех, бинаризация и др.).
2. Методы распознавания образов, а более конкретно – сверточные нейронные сети.

Результатом работы выступает настольное приложение, которое на основе распознавания образов ставит диагноз по изображению плеврального выпота больного человека.

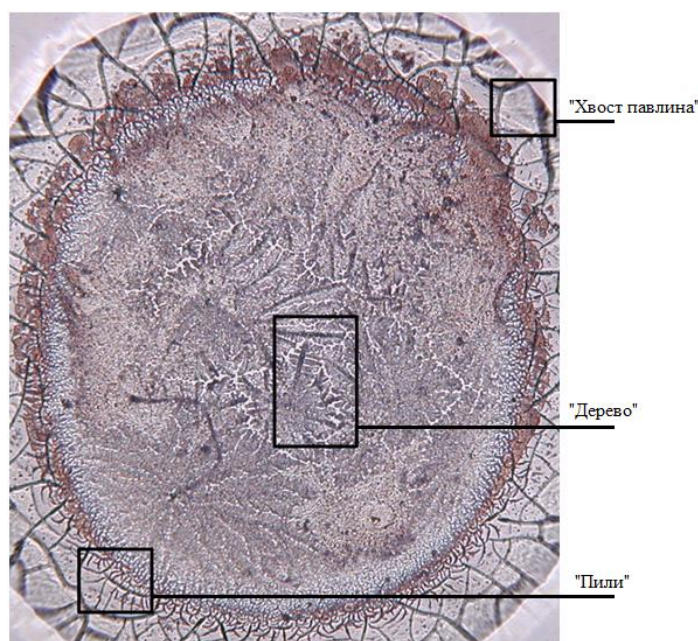
## **Глава 1. Теоретический обзор**

### **1.1 Диагностика заболеваний медицинскими работниками**

При лечении пациентов врачам, в первую очередь, необходимо составить заключение, понять, чем человек болен, насколько сильно развилась болезнь к настоящему моменту. В этих целях медицинские работники прибегают к разным подходам решения вопроса, но самым основным, конечно, является обследование больного с использованием его анализов. Специалисты в этой области берут у пациентов анализы крови, тканей, клеток, мочи и т.д. Данная процедура необходима не только для выявления заболевания, если оно у человека есть, но и для последующего контроля лечения. После оздоровительных процедур, назначенных лечащим врачом, пациент повторно сдаёт необходимые анализы, и только после этого врач может сказать, удалось ли вылечить от заболевания, или же требуется продолжить курс лечения, изменив некоторые его составляющие.

Врачи-лаборанты же, в свою очередь, рассматривают биологический материал под микроскопом или другой аппаратурой, оформляют результаты исследования и передают их терапевтам или узким специалистам (онколог, нефролог, гастроэнтеролог, кардиолог, пульмонолог и т.д.). Затем специалист по результатам анализов принимает решение, болен ли человек, какое лечение необходимо ему прописать, и составляет план будущих действий.

При работе же непосредственно с самим изображением анализа плеврального выпота [1] больного необходимо найти различные паттерны, по которым и диагностируется определённое заболевание. Для наглядности приведём пример изображения с выделенными паттернами. На Рисунке 1 были найдены такие паттерны, как «хвост павлина», «дерево» и «пили», следовательно, можно сделать вывод, что на изображении мы наблюдаем анализ плеврального выпота человека, больного онкологическим заболеванием.



**Рисунок 1 Образец плеврального выпота человека, больного онкологическим заболеванием**

Название паттернов, которые можно наблюдать на изображении – это названия, которые используют специалисты в своей области, названы же они так из-за внешнего сходства с предметами. Так, например, некоторые другие паттерны называются «череп», «звезда», «морщины», «соль», «пятно», «роза» и т.д. Также у одного и того же заболевания могут быть разные комбинации паттернов, что может в некотором роде усложнять диагностику.

Предварительно же необходимо ознакомиться с существующими системами в данной области, методами, используемыми для реализации таких систем.

## **1.2 Анализ существующих систем**

В настоящее время достаточно часто применяются методы распознавания образов и машинного обучения в медицине. С каждым годом в данной области совершаются новые открытия, и сама область изменяется и растёт. Конечно же, немаловажную роль в этом играет и машинное обучение. В медицине распознавание образов позволяет решать задачи установления диагнозов, предсказания и д.р.

Так, например, амбициозный проект IBM Medical Sieve [2] направлен на то, чтобы диагностировать как можно больше заболеваний благодаря умному



программному обеспечению. Чтобы оценить результаты МРТ, рентген-снимков, кардиограмм, врачу в среднем нужно потратить значительно больше времени на изучение картинки, чем системе машинного обучения. При этом точность компьютерного анализа в среднем выше, что позволит выявить дефекты и образования, которые мог бы пропустить человек. За счёт уменьшения количества времени на распознавание и обработку данных, врачи-радиологи смогут сконцентрироваться на наиболее важных и сложных случаях, вместо того, чтобы проверять сотни снимков ежедневно. Аппарат использует расширенную мультимодальную аналитику и клинические знания, способен анализировать и предлагать решения в области кардиологии и радиологии.

Вторым проектом, который хотелось бы выделить, является проект под названием Google DeepMind Health [2]. Данная система применяет технологии искусственного интеллекта в медицине. В данный момент DM Health сотрудничает с лондонской больницей «Мурфилдс Ай» (Moorfields Eye Hospital), и следствием этого сотрудничества будет анализ тысячи анонимных глазных снимков с целью обнаружения первичных симптомов слепоты. Также, в сотрудничестве с больницей Университетского колледжа Лондона (University College London Hospital), искусственный интеллект будет задействован в проекте по разработке алгоритма, который сможет автоматически различать здоровые и раковые ткани в области головы и шеи.

Также компания Enlitic [3] использует методы и технологии глубокого обучения, в частности, технологии распознавания образов, чтобы собирать полезную информацию из рентгеновских снимков и использовать ее для принятия клинических решений. Технология данной компании может очень быстро интерпретировать медицинские изображения. Эта технология может одновременно искать несколько возможных заболеваний, что исключительно полезно для раннего обнаружения, планирования лечения и мониторинга заболеваний. Система Enlitic на базе ИИ способна анализировать широкий ряд

неструктурированных источников данных, включая радиологические и паталогические изображения.

В данной работе, как и в вышеописанных системах, использовались возможности нейронных сетей, в частности – сверточных, так как именно сверточные нейронные сети лучше всего справляются с задачами распознавания изображений.

### **1.3 Сверточные нейронные сети**

Нейронная сеть [4] — это последовательность нейронов, соединенных между собой синапсами. Структура нейронной сети пришла в мир программирования прямоком из биологии. Благодаря такой структуре, машина обретает способность анализировать и даже запоминать различную информацию. Нейронные сети также способны не только анализировать входящую информацию, но и воспроизводить ее из своей памяти. Другими словами, нейронная сеть – это машинная интерпретация мозга человека, в котором находятся миллионы нейронов, передающих информацию в виде электрических импульсов.

Сверточные же нейронные сети (convolutional neural networks, CNN) [5] — это широкий класс архитектур, основная идея которых состоит в том, чтобы переиспользовать одни и те же части нейронной сети для работы с разными маленькими, локальными участками входов. Данный класс архитектур был предложен Яном Лекуном и входит в состав технологий глубокого обучения. Как и многие другие нейронные архитектуры, сверточные сети известны довольно давно, и в наши дни у них уже нашлось много самых разнообразных применений, но основным приложением, ради которого люди когда-то придумали сверточные сети, остается обработка изображений.

Сверточная нейронная сеть использует некоторые особенности зрительной коры, в которой были открыты так называемые простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённого набора простых клеток.

Таким образом, идея сверточных нейронных сетей заключается в чередовании сверточных слоев и субдискретизирующих слоев (слоёв подвыборки). Структура сети — однонаправленная (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки.

Сверточная нейронная сеть обладает рядом критических преимуществ:

1. При архитектуре сверточных нейронных сетей возможно распараллеливание вычислительных процессов за счет использования графических процессоров.
2. По сравнению с полносвязной нейронной сетью (перцептрон), сверточная нейронная сеть обладает гораздо меньшим количеством настраиваемых весов, что подталкивает сеть к обогащению демонстрируемой информации при обучении. Что делает возможным распознавание изображений с большим количеством признаков при наименьших требованиях к техническим средствам.
3. Сверточная нейронная сеть обладает достаточной устойчивостью к повороту и сдвигу распознаваемого изображения.

Среди недостатков сверточной нейронной сети обычно отмечают сложную настройку параметров (количество слоев, количества ядер каждого из слоев, варианты реализации каждого из слоев и т.д.). При разработке архитектуры сети стоит учитывать специфику области применения и требования, выдвигаемые к сети.

Традиционно сверточная нейронная сеть [6] содержит в себе следующие типы слоев: сверточные слои (convolution layers), субдискретизирующие слои (subsampling layers) и полносвязные слои (fully-connected layers) на выходе. Далее описаны все слои сети по отдельности.

### 1.3.1 Сверточный слой (convolutional)

Сверточный слой [7] используется для генерации «карт значений» при помощи фильтров (ядер свертки). Веса ядер свертки в начале обучения устанавливаются случайно, и корректировка происходит во время обучения.

Например, если применить свертку с матрицей весов  $W$  размера  $3 \times 3$  к матрице  $X$  размера  $5 \times 5$ . Умножение подматрицы исходной матрицы  $X$ , соответствующей окну, и матрицы весов  $W$  — это не умножение матриц, а просто скалярное произведение соответствующих векторов. Данный пример продемонстрирован на Рисунке 2.

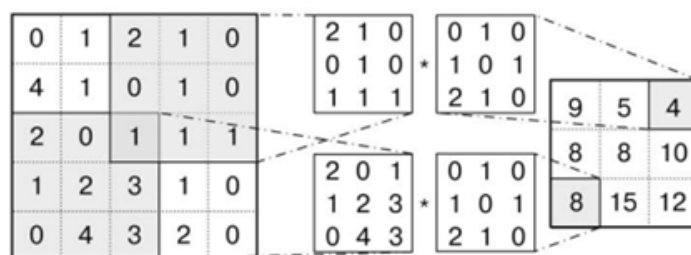
$$\begin{pmatrix} 0 & 1 & 2 & 1 & 0 \\ 4 & 1 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 \\ 1 & 2 & 3 & 1 & 0 \\ 0 & 4 & 3 & 2 & 0 \end{pmatrix} * \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 9 & 5 & 4 \\ 8 & 8 & 10 \\ 8 & 15 & 12 \end{pmatrix}$$


Рисунок 2 Сверточный слой

Почти всегда после свертки в нейронной сети следует нелинейность, вследствие чего требуется применить ту или иную нелинейную функцию  $h$ , которая будет применяться к каждому элементу полученного тензора по отдельности. Данный слой называется активационным. В качестве функции  $h$  часто используют ReLU, особенно в глубоких сетях, но и классические  $\sigma$  и  $\tanh$  тоже встречаются.

### 1.3.2 Слой субдискретизации (subsampling)

Субдискретизирующий слой [7] также называется слоем подвыборки (pooling). В сверточных сетях обычно исходят из предположения, что наличие или отсутствие того или иного признака гораздо важнее, чем его точные координаты. Субдискретизация же сокращает размерность, «обобщая» выделяемые признаки посредством потери части информации об их местоположении, т.е. занимается уплотнение карты признаков посредством

сжатия изображения (обычно группа пикселей 2x2 преобразуются в 1 пиксель).

В качестве операции подвыборки к каждой локальной группе нейронов чаще всего применяется операция взятия максимума (max-pooling), такая субдискретизация восходит еще к работам Хьюбела и Визеля. Иногда встречаются и другие операции подвыборки, например, первые группы ЛеКуна использовали для субдискретизации взятие среднего, а не максимума. Однако именно максимум встречается на практике чаще всего и для большинства практических задач дает хорошие результаты.

Как правило, рассматривается случай, когда шаг подвыборки и размер окна совпадают, то есть получаемая на вход матрица делится на непересекающиеся окна, в каждом из которых мы выбираем максимум.

Так, например, на Рисунке 3 слева продемонстрирована исходная матрица, затем идет матрица после субдискретизации с шагом 1, и справа находится матрица после субдискретизации с шагом 2.

0	1	2	1
4	1	0	1
2	0	1	1
1	2	3	1

4	2	2
4	1	1
2	3	3

4	2
2	3

**Рисунок 3 Слой подвыборки**

Хотя в результате подвыбоки действительно теряется часть информации, сеть становится более устойчивой к небольшим трансформациям изображения вроде сдвига или поворота.

Слой субдискретизации, как правило, идет сразу после сверточного слоя и выполняет функцию отсеивания ненужной информации.

### 1.3.3 Полносвязный слой (fully-connected)

После прохождения всех слоев свертки и пуллинга, остается большой набор каналов, хранящих абстрактные понятия, полученные из исходного изображения. Эти данные объединяются и передаются на полносвязную

нейронную сеть [7], состоящую из одного или более слоев. В отличие от сверточного слоя, в полносвязном слое каждый нейрон соединен со всеми нейронами на предыдущем слое, каждая связь имеет свой коэффициент.

Все возможности сверточных нейронных сетей реализованы в библиотеке Keras на языке программирования Python.

## **1.4 Библиотека Keras**

Keras – открытая высокоуровневая нейросетевая библиотека, написанная на языке Python, способная работать с такими фреймворками, как Theano, TensorFlow и CNTK. Разработана была данная библиотека с целью предоставления оперативной работы с сетями глубинного обучения.

Особенности библиотеки Keras [8]:

1. Позволяет легко и быстро создавать прототипы (благодаря удобству, модульности и расширяемости).

1.1 Удобство для пользователя. Keras ставит пользовательский опыт в основе всего. Данная библиотека следует лучшим рекомендациям по снижению когнитивной нагрузки: она предлагает согласованные и простые API, минимизирует количество действий пользователя, необходимых для общих случаев использования, и обеспечивает четкую и действенную обратную связь при ошибке пользователя.

1.2 Модульность. Под моделью понимается последовательность автономных, полностью настраиваемых модулей, которые могут быть подключены с минимальным количеством ограничений. В частности, нейронные слои, функции затрат, оптимизаторы, схемы инициализации, функции активации и схемы регуляризации – все это отдельные модули, которые пользователь может комбинировать для создания новых моделей.

1.3 Легкая расширяемость. Возможность легко создавать новые модули (как новые классы и функции) обеспечивает полную

выразительность, что делает библиотеку Keras подходящей для передовых исследований.

2. Поддерживает как сверточные сети, так и рекуррентные сети, а также их комбинации.
3. Может без проблем работать как на CPU, так и на GPU.

Далее будут кратко описаны самые часто используемые возможности библиотеки Keras [8]:

## 1. Модели:

1.1 Модель типа Sequential, последовательная модель, где слои идут друг за другом, у такой модели обязательно один вход и один выход.

1.2 Модель типа Model, которая позволяет создавать модели с несколькими входами и несколькими выходами, слои необязательно идут друг за другом, также возможно ветвление и параллельное использование нескольких слоёв.

### 1.3 Подходы для работы с моделями:

1.3.1 Sequential API. Создаётся модель типа Sequential, в неё добавляются слои, затем модель обучается.

1.3.2 Functional API. Слои не добавляются в модель, а вызываются в виде функций, на вход которым передаётся тензор входных данных, а на выходе функция слоя также возвращает тензор. Таким образом, на функцию слоя можно применить следующую функцию, на полученный тензор ещё одну функцию слоя и т.д.

## 2. Слои:

### 2.1 Основные слои:

2.1.1 Dense – полносвязный слой.

2.1.2 Activation – слой активации.

- 2.1.3 Dropout. Используется для регуляризации и предотвращения переобучения.
- 2.1.4 Flatten. Используется при необходимости преобразования двумерных и трехмерных данных в одномерные.
- 2.2 Сверточные слои:
  - 2.2.1 Conv1D, Conv2D, Conv3D – слои одномерной, двумерной и трехмерной свертка соответственно.
- 2.3 Слои подвыборки:
  - 2.3.1 MaxPooling1D, MaxPooling2D, MaxPooling3D –слой подвыборки с операцией выбора максимального значения.
  - 2.3.2 AveragePooling1D, AveragePooling2D, AveragePooling3D – классический слой подвыборки с операцией выбора среднего значения.
- 2.4 Рекуррентные слои – слои, выход которых соединен со входом.
- 2.5 Продвинутое слое активации:
- 2.6 Слои нормализации.
- 2.7 Слои для добавления шума в нейронную сеть.
- 3. Предварительная обработка.
  - 3.1 TimeseriesGenerator – предварительная обработка для последовательности временных рядов.
  - 3.2 Text Preprocessing – подготовка текстовых данных.
  - 3.3 Image Preprocessing – подготовка изображения.
  - 3.4 Losses – функция ошибки, используемая для обучения глубокой нейронной сети.
    - 3.4.1 mean\_squared\_error – средняя квадратичная ошибка.
    - 3.4.2 mean\_absolute\_error –средняя абсолютная ошибка.
    - 3.4.3 categorical\_crossentropy и binary\_crossentropy используются для задач классификации.
  - 3.5 Метрики – функции, которые используются для оценки производительности модели.



3.5.1 `categorical_accuracy` и `binary_accuracy` – метрики аккуратности.

3.5.2 `top_k_categorical_accuracy` – метрика k-средних.

3.6 Оптимизаторы – часть, реализующая алгоритм обратного распространения ошибки при помощи различных методов оптимизации. В оптимизаторе происходит обучение нейронной сети.

3.7 Adam – метод стохастической оптимизации.

3.7.1 RMSprop. Обычно является хорошим выбором для периодических нейронных сетей.

3.7.2 SGD – классический оптимизатор стохастического градиентного спуска. На практике используется реже, чем RMSprop и Adam.

3.8 Функции активации

3.8.1 Softmax – нормализованная экспоненциальная функция.

3.8.2 ReLU – полулинейная функция  $f(x) = \max(0, x)$ .

3.9 Утилиты.

Библиотека Keras работает с кодом, написанным на языке программирования Python. Совместима же данная библиотека с версиями 2.7–3.6 языка Python.

## 1.5 Компьютерное зрение

Компьютерное зрение [9] — это такая дисциплина (раздел искусственного интеллекта), которая занимается извлечением информации из изображений, причем изображения могут быть разного типа. Это могут быть фотографии, видео, наборы фотографий или медицинские снимки из магнитно-резонансного томографа.

Информация, которую разные алгоритмы извлекают из изображений в компьютерном зрении, тоже может иметь разную природу, разный тип.

Некоторые алгоритмы просто разбивают изображение на части, которые соответствуют отдельным объектам или разным частям объектов. Например, врач может взять медицинский снимок или часть медицинского снимка, разбить его на части и взять ту, которая соответствует интересующему его объекту, например, органу или опухоли, а дальше измерить объем или диаметр этого объекта. В таком случае не только экономится время врача, но и уменьшается вариативность, связанная с его личностью, что особенно полезно в случаях, когда измерения для одного и того же пациента проводятся несколькими врачами, или, когда измерения отделяют несколько месяцев.

Заниматься высокоуровневым компьютерным зрением сейчас очень интересно, потому что скорость прогресса очень высока, и происходит это по следующим причинам. Во-первых, компьютеры с каждым годом становятся мощнее, очевидно, это упрощает задачу. Во-вторых, появляется все больше и больше данных, и они становятся лучше за счет того, что улучшаются обычные фотокамеры и видеокамеры, медицинские сканеры, приборы у биологов и астрономов, и чем лучше данные, тем проще из них извлекать информацию. В-третьих, улучшаются алгоритмы, и тут главный прогресс связан прежде всего с глубинным обучением и сверточными нейронными сетями.

Так как большое количество существующих алгоритмов, работающих с изображениями, уже стали классическими, они были собраны в открытую библиотеку компьютерного зрения – OpenCV (Open Computer Vision).

## **1.6 Библиотека OpenCV**

OpenCV (Open Source Computer Vision Library ) [10] – самая популярная библиотека компьютерного зрения, написанная на языке C/C++, исходный код библиотеки открыт, сама же она включает более 1000 функций и алгоритмов. OpenCV разрабатывается с 1998 г., сначала компанией Intel, теперь же компанией Itseez при активном участии сообщества. Библиотека OpenCV [11] очень популярна, о чем свидетельствует огромное число загрузок данной

библиотеки (более 6 млн.). OpenCV – самая большая библиотека по широте тематики.

Библиотека распространяется по лицензии BSD, что означает, что ее можно свободно и бесплатно использовать как в открытых проектах с открытым кодом, так и в закрытых, коммерческих проектах.

Многомерная архитектура [12] проекта представлена на Рисунке 4. Библиотека состоит из 16 модулей. Функциональность доступна на разных языках: C, C++, C#, Python, CUDA, Java. Поддерживаются основные операционные системы: MS Windows, Linux, Mac, Android, iOS.

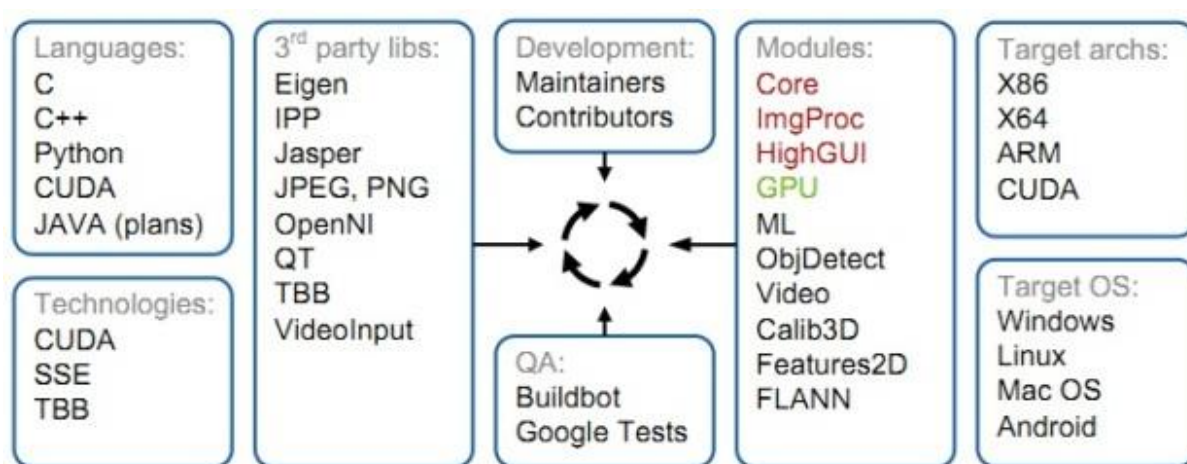


Рисунок 4 Многомерная архитектура проекта

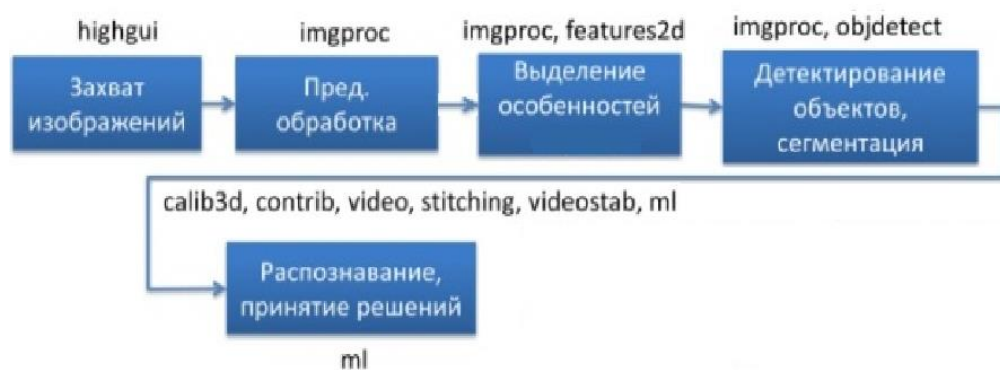
Основные модули библиотеки [12] можно отнести к 4 группам (разделам):

1. Модули core, highgui, реализующие базовую функциональность (базовые структуры, математические функции, генераторы случайных чисел, линейная алгебра, быстрое преобразование Фурье, ввод/вывод изображений и видео, ввод/вывод в форматах XML, YAML и др.).
2. Модули imgproc, features2d для обработки изображений (фильтрация, геометрические преобразования, преобразование цветовых пространств, сегментация, обнаружение особых точек и ребер, контурный анализ и др.).
3. Модули video, objdetect, calib3d (калибровка камеры, анализ движения и отслеживание объектов, вычисление положения в пространстве,

построение карты глубины, детектирование объектов, оптический поток).

4. Модуль ml, реализующий алгоритмы машинного обучения (метод ближайших соседей, наивный байесовский классификатор, деревья решений, случайный лес, машина опорных векторов, нейронные сети и др.).

Общая схема приложения, предназначенного для решения той или иной задачи компьютерного зрения, продемонстрирована на Рисунке 5. Также необходимо отметить, что не все приложения подпадают под эту схему.



**Рисунок 5 Общая схема приложения**

Все начинается с захвата изображений (модуль highgui). Выполняется считывание изображения из файла или видео с сетевой камеры через сетевой протокол. Далее осуществляется предварительная обработка (модуль imgproc), такая как устранение шума, выравнивание яркости, контраста, выделение и удаление бликов, теней.

Следующий этап – выделение особенностей (модули imgproc, features2d). Далее происходит детектирование интересующих нас объектов, выделение значимых частей, сегментация изображения (модули imgproc, objdetect).

После этого следует непосредственно решение основной задачи (модули calib3d, contrib, video, stitching, videostab, ml). В конце происходит распознавание и принятие конкретных решений (модуль ml).

Предварительная обработка изображений играет очень важную роль, были использованы возможности библиотеки OpenCV для предварительной

обработки образов. Ниже описаны все этапы преобразования изображений, реализованные при написании программного кода к данной работе.

### **1.7 Обработка изображений**

Для точного распознавания изображения необходимо сначала провести предварительную обработку распознаваемого образа, т.е. преобразовать исходный образ к более удобному для распознавания виду. Обработка будет осуществляться в 2 этапа:

1. Устранение шумовых помех
2. Бинаризация

Также было реализовано увеличение количества изображений, используемых для обучения нейронной сети, так как на момент написания работы медиками было предоставлено недостаточное количество образов. Увеличение количества изображений было реализовано посредством 40 поворотов на 9 градусов каждого изображения, которое обучало нейронную сеть. Так как изображения были прямоугольными, при повороте они могли претерпеть изменения, уменьшиться, ещё как-либо пострадать. Во избежание этого было принято решение первоначально привести все изображения к квадратному виду, не изменяя сами изображения, располагая их по центру. Каждый этап предварительной обработки изображений подробно разобран в главе 2.

### **1.8 Выводы по главе**

Таким образом, был проведен анализ предметной области и анализ существующих систем, рассмотрены возможности библиотек Keras и OpenCV, рассмотрены этапы предварительной обработки изображений и разобрана архитектура сверточных нейронных сетей и ее составляющие. В следующей же главе речь пойдет о проектировании компонентов системы, выборе языка программирования и программных средств.

## **Глава 2. Проектирование системы распознавания**

Проектирование системы исследования изображений плевральных выпотов с использованием ряда медицинских показателей включает в себя несколько этапов:

1. Проектирование требований к информационной системе в целом.
2. Проектирование архитектуры информационной системы.
3. Выбор языка программирования и необходимых технологий для реализации проекта.

### **2.1 Проектирование требований к системе**

Целью данной работы обозначено получение настольного приложения, которое распознавало бы и классифицировало наличие или отсутствие онкологических заболеваний у людей по микроскопическому изображению их плевральных выпотов. Также были выявлены некоторые требования к приложению.

Требования к удобствам использования:

1. Пользователю требуются базовые знания в использовании приложений, знания в своей предметной области для использования данной системы, глубокие познания технического плана же не требуются.
2. Система наглядно изображает ход своей работы.
3. Реализовать максимально возможную скорость хода работы программы.

Требования к надёжности:

1. Минимизировать количество аварийных завершений программы, свести их к нулю.

Требования к функциональным характеристикам:

1. Возможность загружать изображения формата jpg.
2. Классификация изображений по двум классам.
3. Вывод результата распознавания на экран.

Требования к производительности:

1. Скорость анализа одного изображения не должна превышать 3 секунд.

Минимальные системные требования:

1. Операционная система Windows 7 или более поздняя версия.
2. 32-разрядный или 64-разрядный процессор с тактовой частотой 1 ГГц или выше.
3. 1 Гб оперативной памяти.

## **2.2 Проектирование архитектуры системы**

Система состоит из 3 модулей:

1. Модуль обработки изображения

Первый модуль содержит алгоритмы подготовки загруженного образа к обучению/распознаванию. В первую очередь выполняется приведение изображения к квадратному виду, затем устранение шумов, после этого – бинаризация.

2. Модуль обучения – модуль, отвечающий за обучение сверточной нейронной сети.
3. Модуль распознавания и пользовательский интерфейс – модуль, который занимается загрузкой и классификацией изображения, вследствие чего получает ответ и ставит диагноз.

## **2.3 Выбор языка программирования**

Выбор подходящего инструментария является важным моментом при проектировании любой системы, так как это может, как значительно облегчить, так и сильно затруднить процесс разработки. Для реализации данного проекта был выбран такой язык программирования, как Python версии 3.6.6, хоть на данный момент и существуют более поздние версии данного языка программирования. Использование именно этой версии обусловлено тем, что она является последней версией данного языка, которая может работать с библиотекой Keras. Причиной выбора данного языка послужило желание воспользоваться вышеупомянутой библиотекой Keras при

реализации сверточных нейронных сетей. Также на Python'e удобно реализованы все возможности библиотеки OpenCV, которая была выбрана, как инструментальное средство для предварительной обработки изображения. Сам же язык является очень удобным языком для работы с изображениями.

Для реализации пользовательского интерфейса был выбран язык программирования C# в среде разработки Visual Studio 2017. Выбор пал на язык программирования C#, так как на нем очень удобно реализовывать пользовательский интерфейс, чем Python, к сожалению, похвастаться не может.

Также кроме непосредственно самих языков программирования, при реализации программного продукта использовались возможности некоторых программ, сервисов и библиотек языка Python. Всё используемое ПО и библиотеки описаны ниже.

## **2.4 Выбор программных средств**

В данном разделе будут описаны используемые во время реализации системы программные продукты, библиотеки и сервисы. Список используемых библиотек:

1. Pillow (import PIL) [13] – стандартная вспомогательная библиотека для работы с изображениями.
2. NumPy (import numpy) – библиотека с исходным кодом, используемая для высокоуровневых математических операций.
3. OpenCV (import cv2) – библиотека для компьютерного зрения, используемая для предварительной обработки изображений и др.
4. OS (import os) – библиотека, предназначенная для взаимодействия программы с ОС, для управления файлами.
5. Keras (import keras) – многофункциональная библиотека для машинного обучения, используемая для работы со сверточными нейронными сетями и др.



Также необходимо упомянуть вспомогательные программы и сервисы, которые использовались в процессе разработки:

1. Visual Studio Code – редактор исходного кода, разработанный компанией Microsoft. Выбор пал именно на VSC, так как он является удобным редактором, имеет встроенную командную строку (terminal).
2. GitHub Desktop – удобная утилита, помогающая работать с GitHub. Использовался для контроля версий и их управления.
3. Google Colaboratory [14] – это бесплатный облачный сервис компании Google, направленный на упрощение исследований в области машинного и глубокого обучения. Используя Colaboratory, можно получить удаленный доступ к машине с подключенной видеокартой, что сильно упрощает процесс обучения глубоких нейронных сетей.
4. Visual Studio 2017 – интегрированная среда разработки, позволяющая редактировать, отлаживать и создавать программный код, а затем публиковать приложение.

## **2.5 Выводы по главе**

В результате проведенного анализа были сформулированы требования к системе, архитектура системы. Также был обоснован выбор программных средств, языков программирования, которые представлены выше.

### **Глава 3. Разработка системы и обзор результатов**

После того, как язык программирования, методы обучения нейронных сетей и вспомогательные программные средства выбраны, архитектура системы и требования к ней сформированы, необходимо приступить непосредственно к разработке системы, тестированию и обзору результатов. В первую очередь необходимо реализовать программный код, который будет отвечать за предварительную обработку изображений.

#### **3.1 Предварительная обработка изображений**

Для точного распознавания изображения необходимо преобразовать исходный образ к более удобному для распознавания виду. В данной главе будут рассмотрены и разобраны все этапы предварительной обработки изображений.

##### **3.1.1 Устранение шумовых помех**

На практике часто встречаются изображения, искаженные шумом [19], появляющимся либо на этапе формирования изображения, либо на этапе передачи. Причинами же возникновения шума могут быть сбои в работе канала связи, дефект пленки, шум видеодатчика и другие. Главной целью выступает эффективное устранение шума, чтобы важные для последующего распознавания детали изображения не пострадали.

2D Convolution [20]. OpenCV предоставляет функцию `cv2.filter2D( )` для объединения ядра с образом. В качестве примера возьмём ядро усредняющего фильтра 5x5. Фильтрация с вышеупомянутым ядром приводит к следующему: для каждого пикселя окно 5x5 центрируется на этом пикселе, все пиксели, попадающие в это окно, суммируются, и результат затем делится на 25. Это и является вычислением среднего значения пикселей внутри этого окна. Эта операция выполняется для всех пикселей изображения, чтобы получить отфильтрованное изображение.

Размытие изображения достигается путем свертывания изображения с помощью фильтра нижних частот. Это полезно для удаления шума. Он фактически удаляет высокочастотный контент (например, шум, края) с изображения, в результате чего края становятся размытыми, когда применяется этот фильтр. OpenCV предоставляет в основном четыре типа техники размытия:

1. Averaging (усреднение) [20]

Это делается путем свертки изображения с помощью нормализованного прямоугольного фильтра. Он берет среднее значение всех пикселей в области ядра и заменяет центральный элемент этим средним значением. Делается это при помощи функции `cv2.blur( )` или `cv2.boxFilter( )`. Мы должны указать ширину и высоту ядра при использовании первой функции, либо аргумент `normalize = False` при использовании второй функции.

2. Gaussian Filtering (Гауссова фильтрация) [20]

В этом подходе вместо блочного фильтра, состоящего из равных коэффициентов фильтра, используется ядро Гаусса. Это делается с помощью функции `cv2.GaussianBlur( )`. Фильтр Гаусса меняет каждую точку текущего слоя или выделения, делая её значение равным среднему значению всех точек в определённом радиусе от рассматриваемой точки. Для того чтобы устранить шум, необходимо взять небольшой размер окна сглаживания.

3. Median Filtering (Медианная фильтрация) [20]

Здесь функция `cv2.medianBlur( )` вычисляет медиану всех пикселей под окном ядра, и центральный пиксель заменяется этим средним значением. Интересно отметить, что в гауссовых и блочных фильтрах отфильтрованное значение для центрального элемента может быть значением, которое может не существовать в исходном изображении. Однако это не относится к медианной фильтрации, поскольку центральный элемент всегда заменяется некоторым значением

пикселя в изображении, что эффективно снижает шум. Размер ядра же должен быть положительным нечетным целым числом.

#### 4. Bilateral Filtering (Двусторонняя фильтрация) [20]

Фильтры, представленные ранее, имеют тенденцию размывать края. Это не относится к двустороннему фильтру, `cv2.bilateralFilter()`, который очень эффективен для удаления шума при сохранении краев, но работает медленнее, чем другие фильтры.

В рамках данной работы для устранения шумовых помех выбор пал на двустороннюю фильтрацию Bilateral Filtering. Пример сглаживания шумов разными методами приведен на Рисунке 6.

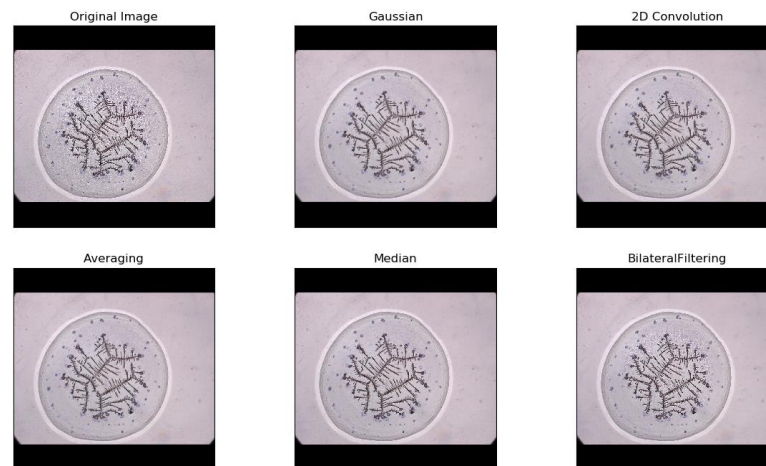


Рисунок 6 Фильтрации образа

#### 3.1.2 Бинаризация

Бинаризация изображений [21], т.е. перевод полноцветного или в градациях серого изображения в монохромное, где присутствуют только два типа пикселей (темные и светлые) имеет большое значение при распознавании образов. OpenCV предоставляет в основном три типа техники бинаризации изображения [22]: Simple Thresholding (с использованием простого порога), Adaptive Thresholding (с использованием адаптивного порога) и Otsu's Binarization (Бинаризация Оцу).

При бинаризации с использованием простого порога изображение делится на несколько каналов. Все пиксели, насыщенностью больше определённого значения, попадают в первый канал, который конвертируется в черный цвет. Во второй же канал попадают все пиксели, чья насыщенность меньше того же значения, данный канал конвертируется в белый цвет.

Бинаризация с нижним порогом – наиболее простая операция, в которой используется только одно значение порога, в соответствии со следующей формулой:

$$f'(m, n) = \begin{cases} 1, f(m, n) \geq t; \\ 0, f(m, n) < t, \end{cases} \quad (5)$$

где  $t$  – величина порога,  $f$  – исходное изображение,  $f'$  – преобразованное,  $m, n$  – координаты пикселя.

Используемая функция – `cv2.threshold()`. Первым аргументом является исходное изображение, которое должно быть изображением в градациях серого. Вторым аргумент – это пороговое значение, которое используется для классификации значений пикселей. Третий аргумент – это `maxVal`, который представляет значение, которое будет дано, если значение пикселя больше порогового значения. OpenCV предоставляет различные стили порогового значения, и это определяется четвертым параметром функции (в нашем случае был взят `cv2.THRESH_BINARY`).

При адаптивном пороге алгоритм вычисляет порог для небольших областей изображения. Таким образом, мы получаем разные пороговые значения для разных областей одного и того же изображения, и это дает нам лучшие результаты для изображений с разной освещенностью.

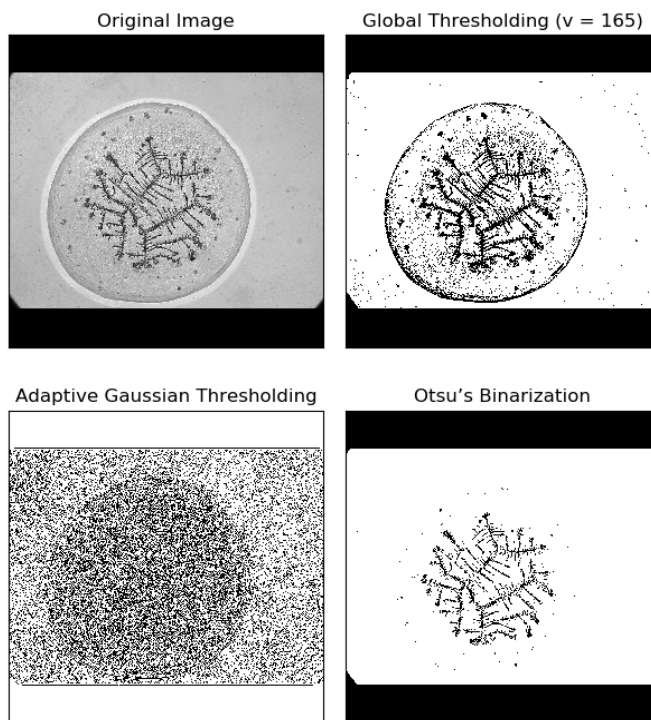
Используемая функция – `cv2.adaptiveThreshold()`. Данная функция имеет три «специальных» входных параметра и только один выходной аргумент. Входные параметры: `cv2.ADAPTIVE_THRESH_MEAN_C` (пороговое значение, которое является средним значением области соседства) или `cv2.ADAPTIVE_THRESH_GAUSSIAN_C` (пороговое значение, которое представляет собой взвешенную сумму значений окрестностей, где веса

представляют собой гауссово окно), block size (размер блока), который определяет размер окрестности и значение  $C$ , являющееся константой, которая вычитается из вычисленного среднего или взвешенного среднего.

Бинаризация Оцу автоматически вычисляет пороговое значение из гистограммы изображения для бимодального изображения. Для изображений, которые не являются бимодальными, бинаризация не будет точной.

Для бинаризации используется функция `cv2.threshold()`, но передается дополнительный флаг `cv2.THRESH_OTSU`. Для порогового значения вводится ноль, затем алгоритм находит оптимальное пороговое значение и возвращает в качестве второго выхода `retVal`, если пороговое значение Otsu не используется, `retVal` соответствует пороговому значению, которое вы использовали.

Для решения задачи бинаризации в программе применяется алгоритм с использованием простого порога, так как все изображения сделаны при помощи современного микроскопа, типизированы и не имеют явного освещения. Пример бинаризации образа разными методами приведён на Рисунке 7.

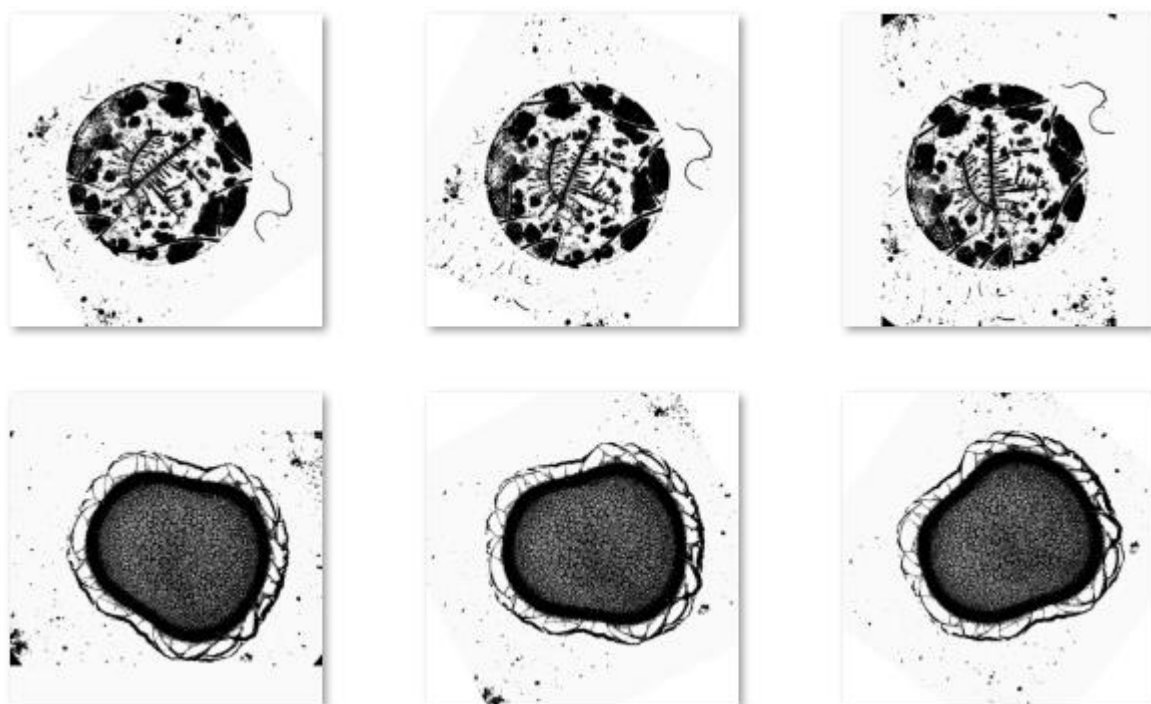


**Рисунок 7 Бинаризации образа**

### 3.1.3 Повороты изображений

Для увеличения количества элементов обучающего множества было решено каждое изображение разбить на 40 изображений посредством поворота первоначального образа на 9 градусов, и затем каждого последующего так же на 9 градусов. В итоге получаем изображение без поворота, повернутое на 9, 18, 27 и т.д. градусов.

Для поворота изображения использовались возможности библиотеки OpenCV, а именно, функция `getRotationMatrix2D()`, которая принимает, как параметры, координаты фрагмента изображения (в нашем случае берутся координаты середины изображения), вокруг которого нужно вращать, и значение количества градусов, на которое требуется повернуть изображение. Поворот, в свою очередь, осуществляется против часовой стрелки. Примеры поворотов изображений после устранения шумовых помех и после бинаризации представлены на Рисунке 8.



**Рисунок 8 Повороты образа**

После того, как все изображения прошли предварительную обработку, они готовы к обучению сверточной нейронной сети. Теперь необходимо спроектировать архитектуру нашей сверточной нейронной сети, весь процесс проектировки представлен в следующем разделе.

### 3.2 Проектирование архитектуры сверточной нейронной сети

Как было описано ранее в 1-ой главе, сверточная нейронная сеть представляет собой специальную архитектуру, состоящую из чередующихся сверточных и субдискретизирующих слоев, активационных и полносвязных слоев.

Учеными было проведено большое количество исследований на тему наилучшей архитектуры для CNN, но абсолютно верного решения для любой задачи найдено не было. Вследствие чего для каждой определенной задачи требуется своя архитектура, которая учитывала бы особенности решаемой задачи, наличие необходимых системных и временных ресурсов.

Наиболее популярными и результативными архитектурами выступают такие архитектуры, как VGG16, AlexNet, Inception и др. Данные архитектуры направлены на обучение с использованием больших вычислительных мощностей на основе больших наборов данных, предназначенных для обучения и тестирования сети. Далее будут представлены упрощенные варианты перечисленных архитектур, полученные экспериментальным путем.

Первой среди архитектур будет выступать архитектура на основе VGG16 [15], представленная на Рисунке 9:

1. На вход подается изображение  $128 \times 128$  пикселей.
2. Далее 2 раза повторяются: сверточный слой, состоящий из 64 фильтров с фильтром свертки размера  $3 \times 3$ , функция активации ReLU, слой субдискретизации с использованием метода выбора максимального значения (max pooling) с фильтром подвыборки  $2 \times 2$ .
3. Затем следует 2 повторения: сверточный слой, состоящий из 32 фильтров с теми же размерами фильтра свертки, функция активации ReLU, слой субдискретизации с тем же фильтром подвыборки и методом выбора значения.
4. После этого следует один плоский слой.
5. Далее 2 полносвязных слоя с 128 нейронами и функциями активации ReLU после каждого.



6. Далее идёт слой регуляризации.

Dropout – техника предотвращения переобучения, которая в процессе обучения, когда на вход нейронной сети подаётся новый объект, случайным образом выключает некоторое количество нейронов с заданной вероятностью. В нашем случае Dropout(0.5) означает, что нейрон будет отключаться с вероятностью 50%. Оставшимся в сети нейронам во время обучения приходится подбирать веса так, чтобы обнаруживать важные признаки самостоятельно, без участия соседних нейронов.

7. Полносвязный слой с 1 нейроном на выходе и функцией активации Sigmoid, так как у нас бинарная классификация, и в качестве результата архитектуры выступают 2 класса – онкология и не онкология.

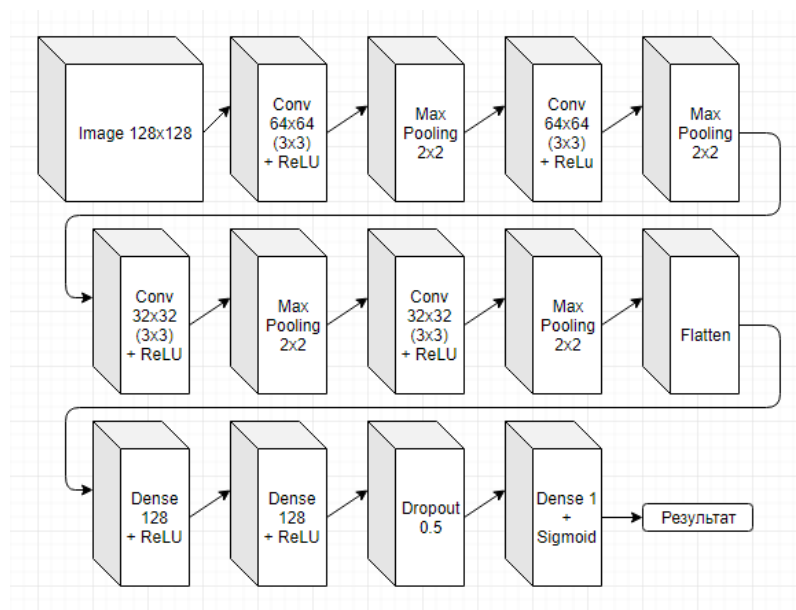


Рисунок 9 Архитектура 1

Вторая архитектура также была разработана на основе архитектуры VGG16 [15], отличается она от первой большим количеством сверточных слоев, где дополнительные сверточные слои после своих функций активации не имеют субдискретизирующих слоев, также отличается максимальное число фильтров в сверточных слоях и количество нейронов в полносвязных слоях. Архитектура под номером 2 продемонстрирована на Рисунке 10.

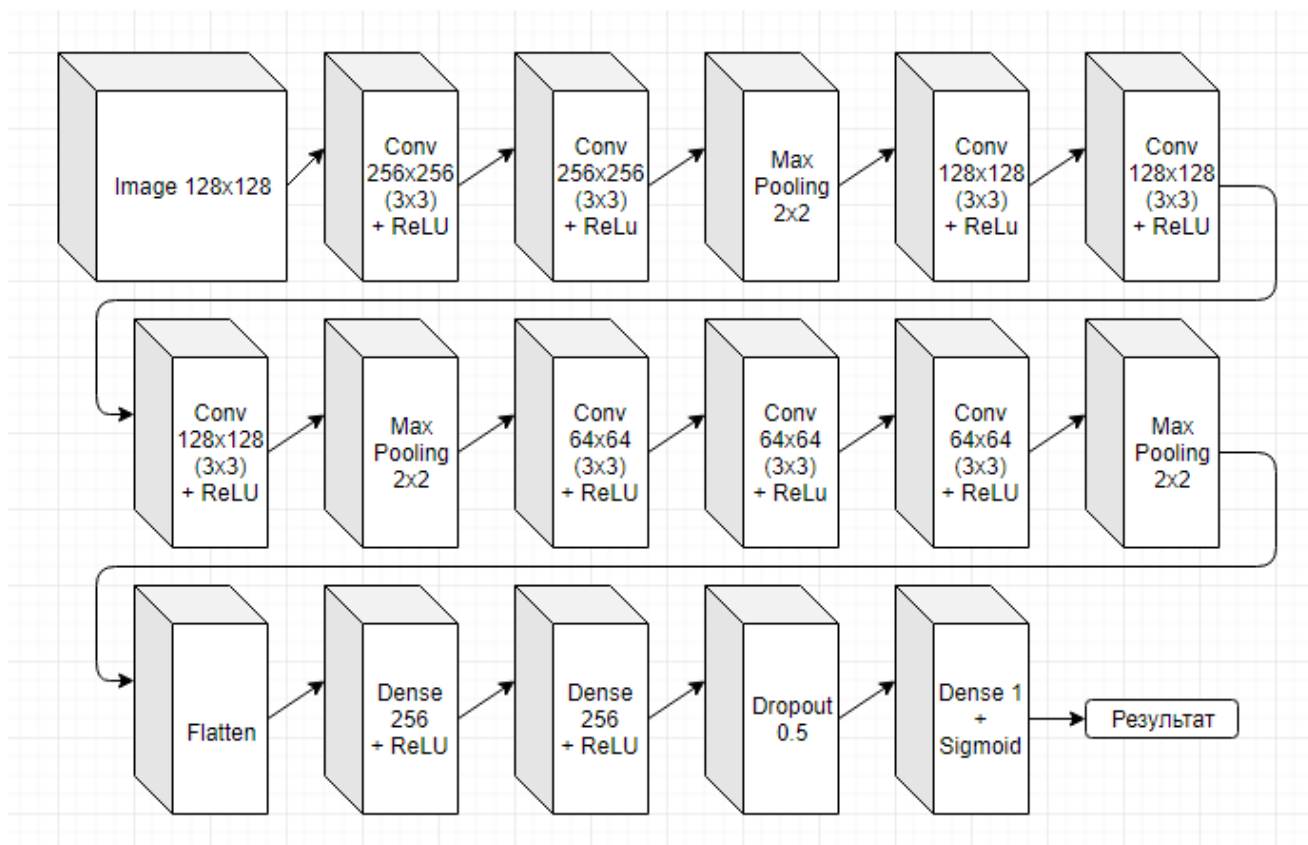
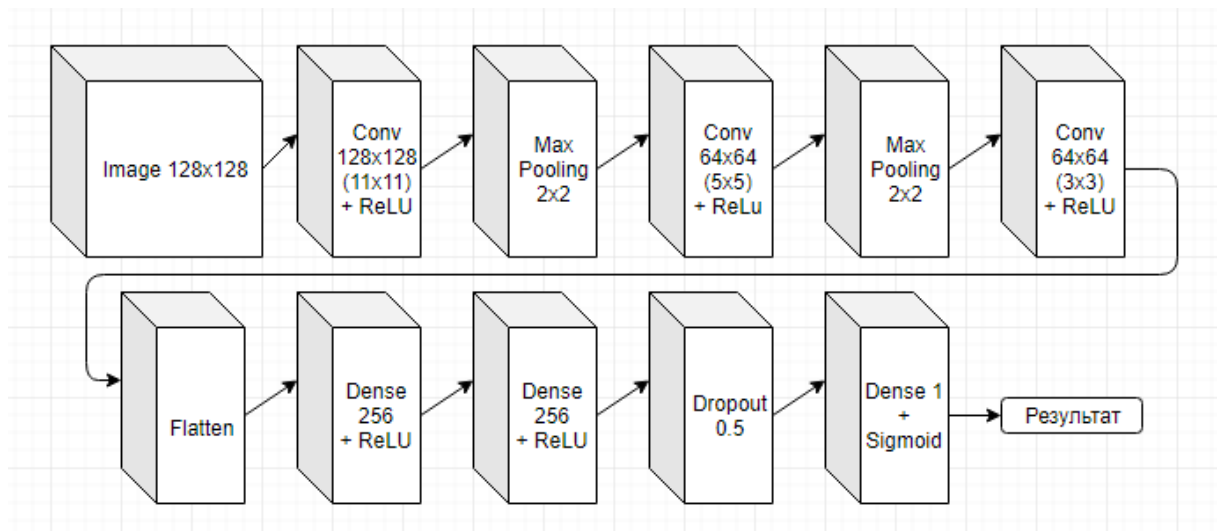


Рисунок 10 Архитектура 2

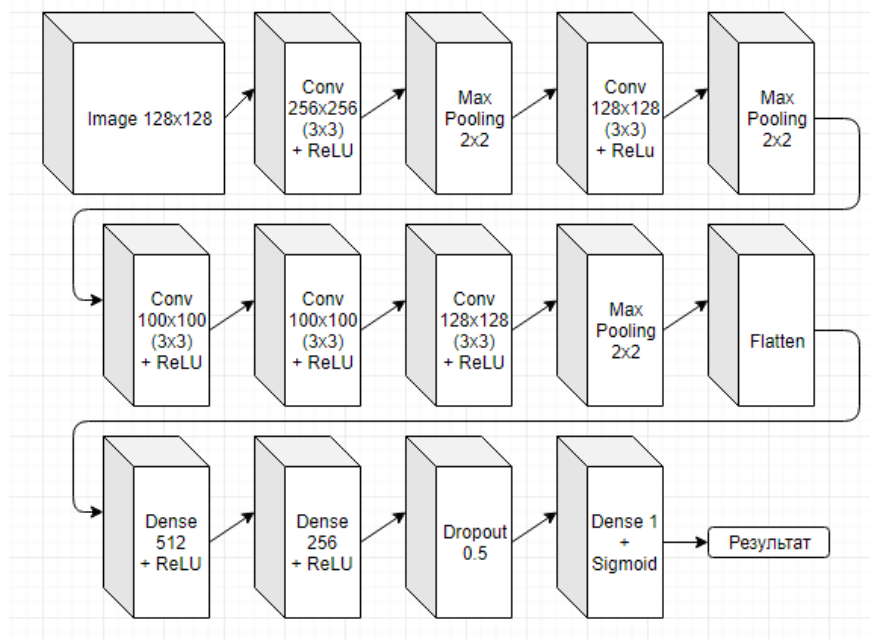
Третья архитектура, которая продемонстрирована на Рисунке 11, была разработана на основе указанной выше AlexNet [16] и представляет собой:

1. Сверточный слой, состоящий из 128 фильтров с фильтром свертки размера  $11 \times 11$ , функция активации ReLU, слой субдискретизации с использованием метода выбора максимального значения с фильтром подвыборки  $2 \times 2$ .
2. Затем следует 2 сверточных слоя, состоящих из 64 фильтров с размерами фильтров  $5 \times 5$  и  $3 \times 3$  соответственно, функциями активации ReLU и слоями субдискретизации с тем же фильтром подвыборки и методом выбора значения после каждого сверточного слоя.
3. После этого следует один плоский слой, 2 полносвязных слоя с 256 нейронами и функциями активации ReLU после каждого, слой регуляризации, полносвязный слой с 1 нейроном и функцией активации Sigmoid.



**Рисунок 11 Архитектура 3**

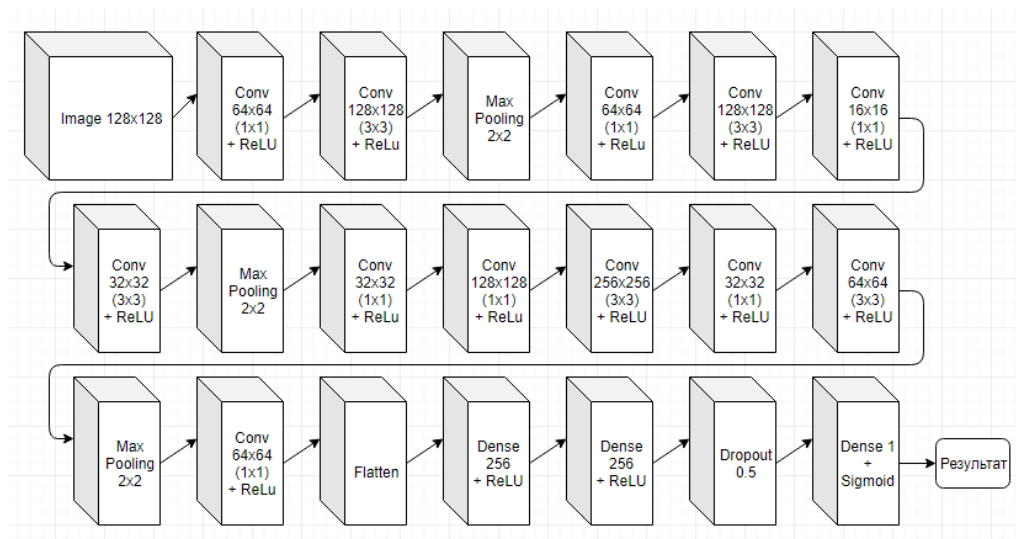
Четвертая архитектура так же была разработана на основе AlexNet [17], отличается от предыдущей она сверточными слоями, размерами ядер свертки, которые в данном случае все равны  $3 \times 3$ , также отличительной чертой является количество нейронов на первом полносвязном слое, которое составляет 512 нейронов. Данная архитектура продемонстрирована на Рисунке 12.



**Рисунок 12 Архитектура 4**

Последняя же нейронная сеть, представленная на Рисунке 13, была сформирована на основе архитектуры Inception [18] и представляет собой:

1. Сверточный слой, состоящий из 64 фильтров с фильтром свертки размера  $1 \times 1$ , функция активации ReLU, сверточный слой, состоящий из 32 фильтров с фильтром свертки  $3 \times 3$ , функция активации ReLU, слой субдискретизации с использованием метода выбора максимального значения с фильтром подвыборки  $2 \times 2$ .
2. Далее идут сверточные слои с функциями активации ReLU и следующим количеством фильтров: 64 фильтра с ядрами  $1 \times 1$ , 128 фильтров с ядрами  $3 \times 3$ , 16 фильтров с ядрами  $1 \times 1$ , 32 фильтра с ядрами  $3 \times 3$ . Затем следует слой субдискретизации с использованием метода выбора максимального значения с фильтром подвыборки  $2 \times 2$  и еще один сверточный слой с 32 фильтрами, ядром свертки  $1 \times 1$  и функцией активации ReLU.
3. Далее повторяется архитектура из 2-го пункта с отличными от вышеуказанных значений. Все числовые показатели также продемонстрированы на рисунке ниже.
4. После этого следует один плоский слой, полносвязный слой с 256 нейронами и функциям активации ReLU, слой регуляризации, полносвязный слой с 1 нейроном и функцией активации Sigmoid.



**Рисунок 13 Архитектура 5**

Для того, чтобы выбрать самую подходящую для нашей задачи архитектуру был проведен ряд тестов. Результаты тестирования всех архитектур продемонстрированы в следующей главе.

### 3.3 Обучение сети и сравнение результатов архитектур

Далее будет описан процесс обучения нейронной сети [23], который применялся ко всем архитектурам. Указанные ниже числовые значения получены экспериментальным путем и являются самыми результативными. После описания обучения будет приведена таблица, в которой между собой сравнены все архитектуры, спроектированные нами ранее. Исходный код обучения нейронной сети продемонстрирован в Приложении 1.

Обучение нейронной сети проводилось в течение 20 эпох, т.е. обучение нейронной сети производилось 20 раз на всём наборе данных, который составил 2800 изображений. Размер мини-выборки составил 20 элементов. В данном случае мы берем 20 изображений, рассчитываем по ним функцию ошибки, затем вычисляем градиент и изменяем веса в нейронной сети, после этого переходим к следующей мини-выборке, состоящей также из 20 изображений. Всего изображений в каждом классе было по 1400 (до осуществления поворотов их было по 35), 20% из которых передавалось в проверочное множество.

При компиляции вызывается функция `compile()`, в качестве функции ошибки указывается `binary_crossentropy`, так как имеется всего 2 класса (онкология и не онкология). Оптимизатор `adam`, метрика `accuracy`. Также создаем генератор изображений, который основывается на классе `ImageDataGenerator`. Генератор делит значения всех пикселей изображения на 255. Затем создаем генераторы данных для обучения и тестирования на основе изображений из каталога.

При помощи метода `fit_generator()` происходит непосредственно обучение нейронной сети. Данному методу сообщается генератор данных для обучения и генератор данных для проверки, также требуется указать необходимое число обращений к генераторам и количество эпох обучения. Сохранение модели осуществляется каждый раз, когда процент правильной работы программы растет, реализуется это при помощи метода `ModelCheckpoint()`.

Все 5 архитектур были задействованы в ряде тестов с целью выбора наилучшей из них. Все тесты проводились на одном и том же обучающем множестве с одинаковым количеством эпох, мини-выборкой и др. показателями. В Таблице 1 продемонстрированы результаты тестирования различных архитектур сети.

**Таблица 1 Результаты тестирования**

Имя архитектуры	Популярная архитектура, лежащая в основе	Точность
Архитектура 1	VGG16	95,71%
Архитектура 2	VGG16	63,92%
Архитектура 3	AlexNet	67,34%
Архитектура 4	AlexNet	81,71%
Архитектура 5	Inception	59,21%

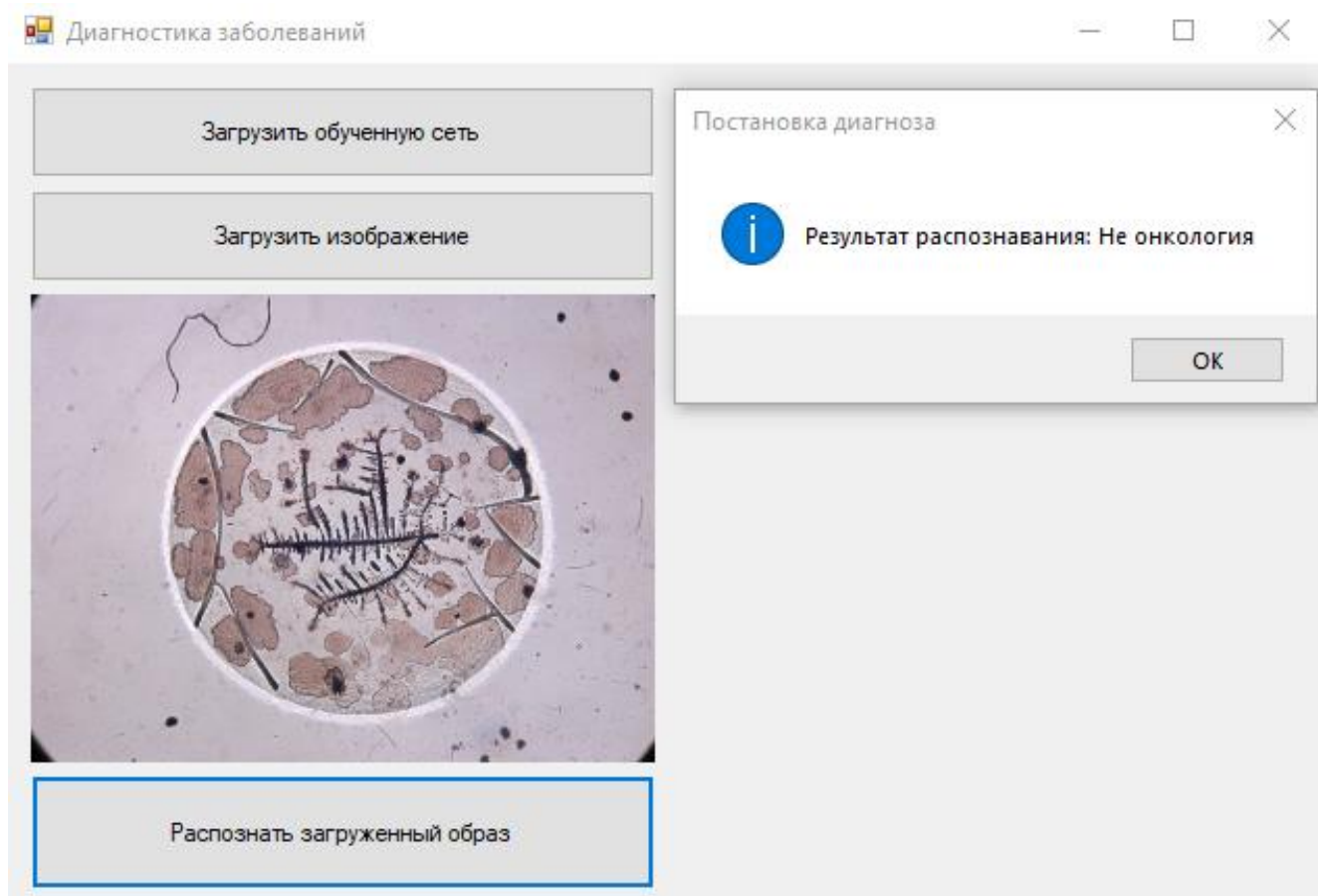
Наилучший результат показала архитектура под номером 1, в результате работы программы с данной архитектурой была получена обученная сверточная нейронная сеть, имеющая аккуратность 95,71% на тестовых данных.

Также был реализован программный код на языке программирования Python, отвечающий непосредственно за само распознавание входных изображений, данный код также приведен в Приложении 1. Последним этапом стала реализация пользовательского интерфейса и тестирование системы.

### **3.4 Реализация пользовательского интерфейса**

Программный продукт реализован на языке программирования C#, состоит из 1 формы, при помощи которой необходимо выбрать и загрузить обученную сеть и изображение для распознавания. При нажатии на кнопку «Распознать загруженный образ» вызывается программный код на языке Python, предназначенный для распознавания, в качестве аргументов при вызове выступают пути к таким файлам, как обученная сеть и распознаваемое изображение. Далее система распознает образ и возвращает результат непосредственно в приложение, реализованное на языке C#, которое, в свою очередь, выводит ответ на экран. Программный код пользовательского

интерфейса представлен в приложении 1. Работа приложения продемонстрирована на Рисунке 14.



**Рисунок 14** Форма для диагностики заболевания

### **3.5 Тестирование программной системы**

Необходимо протестировать основные функции программы методом черного ящика. Данный этап необходим для того, чтобы взглянуть на программу глазами пользователя, тем самым при необходимости избавиться от ошибок, с которыми мог бы столкнуться сам пользователь при эксплуатации программы. Результаты тестирования представлены в Таблице 2.

**Таблица 2 Функциональные требования**

	Функция	Ожидаемый результат	Результат
1	Загрузка изображения для последующей его обработки	Изображение добавлено в ячейку формы распознавания	Пройден
2	Попытка распознать образ без предварительной загрузки изображения	Сообщение о том, что для распознавания необходимо выбрать образ	Пройден
3	Попытка распознать загруженный образ без предварительной загрузки сети	Сообщение о том, что для распознавания необходимо выбрать обученную сеть	Пройден
4	Попытка распознать загруженный образ с предварительной загрузкой обученной сети	Вывод результата распознавания на экран	Пройден

### **3.6 Выводы по главе**

В данной главе был описан процесс обучения сети, предварительной обработки и распознавания изображений, тестирование работоспособности программной системы и тестирование разных архитектур с выбором наилучшей из них. В результате всех действий, описанных выше, был получен программный продукт, который распознаёт загружаемые фотографические изображения плевральных выпотов пациентов, основываясь на обученной нейронной сети.



## Заключение

В работе рассмотрено создание программного продукта для классификации изображений плевральных выпотов пациентов. Было выбрано 2 класса: онкологические заболевания пациентов и неонкологические (другие заболевания, например, туберкулез, сердечная недостаточность и т.д.). Был проведен анализ предметной области и существующих программных систем. При выполнении ВКР была спроектирована программная система и реализован прототип программного продукта для распознавания фотографических изображений плевральных выпотов.

На этапе проектирования были предложены несколько архитектур сверточных нейронных сетей с описанием всех входящих в них модулей. Среди них была выбрана архитектура, полученная экспериментальным путем, проектирование данной архитектуры описано в 3-ей главе. Эта архитектура оказалось самой продуктивной в реалиях поставленной задачи.

Также были описаны процессы реализации программного продукта для обучения нейронной сети и для распознавания входных изображений плевральных выпотов при помощи обученной сети. Была продемонстрирована реализация программного интерфейса и все этапы предварительной обработки изображений.

Решение данной задачи при помощи средств библиотеки Keras и с использованием сверточных нейронных сетей привело к результату, который оказался лучше, чем результат применения метода kNN (в рамках курсовой работы на 3-ем курсе). В итоге был получен следующий показатель точности классификации – 95,71%. Данный результат можно считать хорошим для использованного в ВКР объема входных данных.

Исследования могут быть продолжены, предполагается использовать результаты, которые могут быть получены при реализации нескольких архитектур сверточных нейронных сетей с применением метода SVM (метод опорных векторов).

### **Список сокращений и условных обозначений**

ВКР – выпускная квалификационная работа

МРТ – магнитно-резонансная томография

CNN – convolutional neural networks (сверточные нейронные сети)

API – application programming interface (программный интерфейс приложения)

CPU – central processing unit (центральный процессор)

GPU – graphics processing unit (графический процессор)

kNN – k-nearest neighbors (метод k-ближайших соседей)

SVM – support vector machine (метод опорных векторов)

### Список использованных источников

1. Клюкина, О.А., Гапанович, О.О., Козаовская, М.Н., Урбанович, Е.В., Толкач, Е.Е. Иммуноцитохимический метод в цитологическом исследовании плевральных жидкостей // Онкологический журнал. 2013. Т. 7, № 4. С. 45–49.
2. Искусственный интеллект в медицине // 22century.ru. 2017. URL: <https://22century.ru/popular-science-publications/artificial-intelligence-in-medicine> (дата обращения: 21.11.2018).
3. Системы искусственного интеллекта для обработки медицинских изображений // evercare.ru. 2017. URL: <https://evercare.ru/ai-medical-imaging> (дата обращения: 02.06.2019).
4. Нейронные сети // habr.com. 2016. URL: <https://habr.com/post/312450> (дата обращения: 13.12.2018).
5. Сверточная нейронная сеть// habr.com. 2016. URL: <https://habr.com/ru/post/309508> (дата обращения: 15.01.2019).
6. Сверточные нейронные сети // intellect.ml. 2016. URL: <https://intellect.ml/svertochnaya-nejronnaya-set-convolutional-neural-network-cnn-6013> (дата обращения: 19.01.2019).
7. Николаенко С., Кадуринов А., Архангельская Е. Сверточные нейронные сети // Глубокое обучение. 2018. С. 176–231.
8. Keras // keras.io. 2019. URL: <https://keras.io> (дата обращения: 14.02.2019).
9. Конушин А., Барина О., Конушин В., Якубенко А., Велижев А. Введение в компьютерное зрение // Graphics & Media Lab. Москва: МГУ ВМК, 2009. С. 1–54.
10. OpenCV library // opencv.org. 2013. URL: <https://opencv.org/> (дата обращения: 11.10.2018).
11. Конев А.А., Пальчикова И.Г. Библиотека OpenCV и её использование в задачах цитофотометрии // журнал Интерэкспо Гео-Сибирь. 2015. Т. 4, № 2. С. 71–76.

12. OpenCV // intuit.ru. 2015. URL: <https://www.intuit.ru/studies/curriculums/19469/courses/1105/lecture/17985?page=1> (дата обращения: 21.10.2018).
13. Pillow // pillow.readthedocs.io. 2014. URL: <https://pillow.readthedocs.io/en/5.1.x> (дата обращения: 05.02.2019).
14. Google Colab // medium.com. 2018. URL: <https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d> (дата обращения: 11.04.2019).
15. VGG16 — сверточная сеть для выделения признаков изображений // neurohive.io. 2018. URL: <https://neurohive.io/ru/vidy-nejrosetej/vgg16-model> (дата обращения: 17.02.2019).
16. AlexNet — свёрточная нейронная сеть для классификации изображений // neurohive.io. 2018. URL: <https://neurohive.io/ru/vidy-nejrosetej/alexnet-svjortochnaja-nejronnaja-set-dlja-raspoznavanija-izobrazhenij> (дата обращения: 12.05.2019).
17. AlexNet Implementation Using Keras // engmrk.com. 2018. URL: <https://engmrk.com/alexnet-implementation-using-keras> (дата обращения: 14.05.2019).
18. Review: GoogLeNet (Inception v1) // medium.com. 2018. URL: <https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7> (дата обращения: 12.05.2019).
19. Апальков И.В., Хрящев В.В. Удаление шума из изображений на основе нелинейных алгоритмов // GraphiCon'2007. Москва: Ярославский государственный университет имени П.Г. Демидова, 2007. С. 1–4.
20. Smoothing Images // opencv.org. 2014. URL: [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_imgproc/py\\_filtering/py\\_filtering.html#filtering](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_filtering/py_filtering.html#filtering) (дата обращения: 27.10.2018).
21. Бинаризация изображений // recog.ru. 2011. URL: <http://recog.ru/blog/applied/15.html> (дата обращения: 08.11.2018).

22. Image Thresholding // opencv.org. 2014. URL: [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_imgproc/py\\_thresholding/py\\_thresholding.html#thresholding](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_thresholding/py_thresholding.html#thresholding) (дата обращения: 08.11. 2018).
23. Онлайн курс "Программирование глубоких нейронных сетей на Python" // asozykin.ru. 2018. URL: <https://www.asozykin.ru/courses/nnpython> (дата обращения: 03.02.2019).

## Приложение. Код программы

Все необходимые данные и программный код размещены в репозитории Git по следующей ссылке: <https://github.com/misha1020/Disease-diagnosis>

Программный фрагмент на языке Python, отвечающий за сортировку и переименование файлов внутри указанной директории.

```
import os
import string
def Capitalize(myStr):
    return myStr.upper()
def LoadFilesNamesFromDir(dir):
    files = os.listdir(dir)
    for i, file in enumerate(files):
        splited = file.split(".")
        splited[1] = Capitalize(splited[1])
        file = splited[0] + "." + splited[1]
        files[i] = file
    imgNames = filter(lambda x: x.endswith(".jpg") or x.endswith(".JPG") or
x.endswith(".JPEG"), files)
    return imgNames
def Rename():
    direc = "../Make Not Onco/"
    imgs = LoadFilesNamesFromDir(direc)
    if (direc == "../Make Onco/"):
        currentClass = "Onco_"
    elif (direc == "../Make Not Onco/"):
        currentClass = "NotOnco_"
    else:
        currentClass = "NoInfo_"
    processName = "Process"
    i = 0
    for Image in imgs:
        os.rename(direc + Image, direc + processName + str(i) + ".JPG")
        i += 1
    imgs = LoadFilesNamesFromDir(direc)
    i = 0
    for Image in imgs:
        os.rename(direc + Image, direc + currentClass + str(i) + ".JPG")
        i += 1
def Sort():
    direc = "../All Images Mixed/"
```

```

imgs = LoadFilesNamesFromDir(direc)
count = 0
for Image in imgs:
    splited = Image.split("_")
    if (splited[0] == "NotOnco"):
        currentClass = "NotOnco_"
    elif (splited[0] == "Onco"):
        currentClass = "Onco_"
    if (count%3 == 2):
        os.rename(direc + Image, direc + currentClass + "2_" + str(count) + ".JPG")
    elif (count%3 == 1):
        os.rename(direc + Image, direc + currentClass + "1_" + str(count) + ".JPG")
    else:
        os.rename(direc + Image, direc + currentClass + "0_" + str(count) + ".JPG")
    count += 1
imgs = LoadFilesNamesFromDir(direc)
# Количество элементов данных в одном классе
nb_images = 35
i = 0
for Image in imgs:
    currentName = Image.split("_")
    os.rename(direc + Image, direc + currentName[0] + "_" + str(i%nb_images) +
".JPG")
    i += 1

#Sort()
Rename()

```

Программный фрагмент на языке Python, отвечающий за копирование и распределение файлов по разным директориям для дальнейшего обучения нейронной сети.

```

import shutil
import os
import numpy as np
from PIL import Image
import cv2
data_dir = "../All Images Final" # Каталог с набором данных
train_dir = "../Images Train Val/train" # Каталог с данными для обучения
val_dir = "../Images Train Val/val" # Каталог с данными для проверки
val_data_portion = 0.2 # Часть набора данных для тестирования (в %)
nb_images = 35 # Количество элементов данных в одном классе
# Создание директории

```

```

def create_directory(dir_name):
    if os.path.exists(dir_name):
        shutil.rmtree(dir_name)
    os.makedirs(dir_name)
    os.makedirs(os.path.join(dir_name, "Onco"))
    os.makedirs(os.path.join(dir_name, "NotOnco"))
# Копирование изображений из одной директории в другую
def copy_images(start_index, end_index, source_dir, dest_dir):
    for i in range(start_index, end_index):
        shutil.copy(os.path.join(source_dir, "Onco_" + str(i) + ".jpg"),
                    os.path.join(dest_dir, "Onco"))
        shutil.copy(os.path.join(source_dir, "NotOnco_" + str(i) + ".jpg"),
                    os.path.join(dest_dir, "NotOnco"))

create_directory(train_dir)
create_directory(val_dir)
start_val_data_idx = int(nb_images * (1 - val_data_portion))
print(start_val_data_idx)
print("Изображений скопировано в train directory.")
copy_images(0, start_val_data_idx, data_dir, train_dir)
copy_images(start_val_data_idx, nb_images, data_dir, val_dir)

```

Программный фрагмент на языке Python, отвечающий за предварительную обработку и повороты изображений.

```

import os
import cv2
import glob
import numpy as np
#import scipy.misc as sm
from PIL import Image
# Устранение шумовых помех
def Blur(img):
    blurBilFiltr = cv2.bilateralFilter(img, 9, 75, 75)
    return blurBilFiltr
# Бинаризация
def Binary(img):
    imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    retG1, thGlobal = cv2.threshold(imgGray, 175, 250, cv2.THRESH_BINARY)
    return thGlobal
#Поворот изображения на указанное число градусов
def Turn(img, val):
    (h, w) = img.shape[:2]

```



```

        center = (w / 2, h / 2)
        M = cv2.getRotationMatrix2D(center, val, 1.0)
        rotated = cv2.warpAffine(img, M, (w, h), borderValue=(255,255,255))
        return rotated

# Приведение изображения к квадратному формату
def Square(im, min_size = 256, fill_color = (255, 255, 255)):
    x, y = im.size
    size = max(min_size, x, y)
    new_im = Image.new('RGBA', (size, size), fill_color)
    new_im.paste(im, ((size - x) // 2, (size - y) // 2))
    return new_im

# Изменение размеров изображения
def Squeeze(img, size):
    original_image = img
    max_size = (size, size)
    original_image.thumbnail(max_size, Image.ANTIALIAS)
    return img

# Конвертация изображения для работы с библиотекой PIL к формату OpenCV изображения
def ConverPILtoOpenCV(pil_image):
    numpy_image = np.array(pil_image)
    opencv_image = cv2.cvtColor(numpy_image, cv2.COLOR_RGB2BGR)
    return opencv_image

# Предварительная обработка всех изображений в директориях и повороты всех изображений
def main():
    directory = "../Images Train Val"
    for dir in ([x[0] for x in os.walk(directory)]):
        for filename in glob.glob(dir + "/*.jpg"):
            print(filename)
            imageInput = Image.open(filename)
            splitted = filename.split(".jpg")
            imageInput = Square(imageInput)
            imageInput = ConverPILtoOpenCV(imageInput)
            imageInput = Blur(imageInput)
            imageInput = Binary(imageInput)
            k = 0
            while (k < 360):
                image = imageInput.copy()
                image = Turn(image, k)
                cv2.imwrite(splitted[0] + "_" + str(k) + ".jpg", image)
                k += 9
            os.remove(filename)

main()

```

Программный фрагмент на языке Python, отвечающий за обучение сверточной нейронной сети.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.models import model_from_json
from keras.callbacks import ModelCheckpoint
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

glob_dir = 'drive/My Drive/Images Train Val/'
train_dir = glob_dir + '/train'
val_dir = glob_dir + '/val'
img_size = 128
input_shape = (img_size, img_size, 3)
epochs = 20 # Количество эпох
batch_size = 20 # Размер мини-выборки
nb_images = 1400 # Количество элементов данных в одном классе
val_data_portion = 0.2 # Часть набора данных для проверки
nb_train_samples = int(nb_images * (1 - val_data_portion))
nb_val_samples = int(nb_images * val_data_portion)

model = Sequential()
model.add(Conv2D(64, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
```

```

model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
datagen = ImageDataGenerator(rescale=1. / 255)

train_generator = datagen.flow_from_directory(
    train_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='binary')
val_generator = datagen.flow_from_directory(
    val_dir,
    target_size=(img_size, img_size),
    batch_size=batch_size,
    class_mode='binary')

filepath='drive/My Drive/Model.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
                             save_best_only=True, mode='max')
callbacks_list = [checkpoint]

model.fit_generator(
    train_generator,
    steps_per_epoch = nb_train_samples // batch_size,
    epochs = epochs,
    validation_data = val_generator,
    validation_steps = nb_val_samples // batch_size,
    callbacks = callbacks_list)

```

Программный фрагмент на языке Python, отвечающий за распознавание и классификацию образов.

```

import numpy as np
import sys
import os
import cv2
from keras.preprocessing import image
from keras.models import load_model
from PIL import Image
from keras.models import model_from_json
from keras import optimizers

```

```

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# Загрузка, предварительная обработка изображения, загрузка модели CNN и классификация
изображения

def main():
    modelName = str(sys.argv[1])
    loaded_model = load_model(modelName)
    filename = str(sys.argv[2])
    splited = filename.split(".jpg")
    imageInput = Image.open(filename)
    imageInput = Square(imageInput)
    imageInput = ConvertPILtoOpenCV(imageInput)
    imageInput = Blur(imageInput)
    imageInput = Binary(imageInput)
    filename = splited[0] + "Bin.jpg"
    cv2.imwrite(filename, imageInput)
    img = image.load_img(filename, target_size=(128, 128))
    x = image.img_to_array(img)
    x /= 255
    x = np.expand_dims(x, axis=0)
    prediction = loaded_model.predict(x)
    os.remove(filename)
    classes=['No', 'Yes']
    print(classes[round(float(prediction[[0]]))])

main()

```

Программный фрагмент на языке C#, являющийся реализацией пользовательского интерфейса

```

public partial class FormRecognizer : Form
{
    Bitmap image;
    string imageName;
    string modelName = "none";
    public FormRecognizer()
    {
        InitializeComponent();
    }
    private string Recognize()
    {
        string result;
        Process p = new Process();
        p.StartInfo.FileName = "python.exe";
        p.StartInfo.RedirectStandardOutput = true;

```

```

        p.StartInfo.UseShellExecute = false;
        p.StartInfo.Arguments = "MyData.py" + " \"" + modelName + "\"" + " \"" +
imageName + "\"";
        p.StartInfo.CreateNoWindow = true;
        p.StartInfo.RedirectStandardError = true;
        p.Start();
        StreamReader s = p.StandardOutput;
        string output = s.ReadToEnd();
        string[] r = output.Split(new char[] { '\r' });
        p.WaitForExit();
        if (r[0] == "Yes")
            result = "Онкология";
        else if (r[0] == "No")
            result = "Не онкология";
        else
            result = "Не удалось распознать";
        return result;
    }
    private void btLoadPhoto_Click(object sender, EventArgs e)
    {
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Filter = "Image Files(*.JPG)|*.JPG|All files (*.*)|*.*";
        if (open_dialog.ShowDialog() == DialogResult.OK)
        {
            try
            {
                imageName = open_dialog.FileName;
                image = new Bitmap(imageName);
                pbImage.Image = image;
                pbImage.Invalidate();
            }
            catch
            {
                DialogResult rezult = MessageBox.Show("Невозможно открыть
выбранный файл", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
    private void btLoadModel_Click(object sender, EventArgs e)
    {
        OpenFileDialog open_dialog = new OpenFileDialog();
        open_dialog.Filter = "Models(*.h5)|*.h5";
        if (open_dialog.ShowDialog() == DialogResult.OK)

```

```

        modelName = open_dialog.FileName;
    }
    private void btRecognize_Click(object sender, EventArgs e)
    {
        if (pbImage.Image == null)
            MessageBox.Show("Для распознавания необходимо выбрать образ!",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else if (modelName == "none")
            MessageBox.Show("Для распознавания необходимо выбрать обученную
сеть!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
        {
            string result = Recognize();
            MessageBox.Show("Результат распознавания: " + result, "Постановка
диагноза", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }
}

```