МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра: математического обеспечения и суперкомпьютерных технологий**

Направление подготовки: «Фундаментальная информатика и информационные технологии»

**ОТЧЕТ**

по учебной практике

**Тема:**

**«Сегментация медицинских изображений.  
Разработка алгоритмов автоматической сегментации данных трехмерного сканирования сердца (КТ, МРТ, УЗИ) с помощью глубокого обучения на языке Python.»**

**Выполнил:** студент группы 381906-3

**\_\_\_\_\_\_\_\_\_**Олюнин Александр Владимирович

Подпись

**Научный руководитель:**

ст. преп. кафедры МОСТ ИИТММ ННГУ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Васильев Евгений Павлович

Подпись

Нижний Новгород

2022

**Оглавление**

[1. Введение 3](#_Toc105444255)

[2. Постановка задачи 5](#_Toc105444256)

[3. Описание фреймворков TensorFlow и Keras 6](#_Toc105444257)

[3.1 Возможности загрузки dataset 6](#_Toc105444258)

[3.2 Возможности Keras в построении моделей 8](#_Toc105444259)

[3.3 Популярные модели сегментации 12](#_Toc105444260)

[3.4 Расширенные возможности оптимизации процесса обучения 14](#_Toc105444261)

[4. Предварительная обработка изображений 19](#_Toc105444262)

[5. Медицинская часть U-Net 21](#_Toc105444263)

[6. Обзор внешних инструментов для обучения моделей 25](#_Toc105444264)

[6.1 Google Colaboratory 25](#_Toc105444265)

[6.2. Описание Pydicom 26](#_Toc105444266)

[7. Первое обучение модели U-Net на медицинских данных 27](#_Toc105444267)

[8. Обучение U-Net по распознаванию почек 34](#_Toc105444268)

[9. Заключение 38](#_Toc105444269)

[10. Список литературы и источников информации 39](#_Toc105444270)

[11. Приложение 41](#_Toc105444271)

[11.1 Листинг выполнения программного кода 41](#_Toc105444272)

[11.2 Код программы 42](#_Toc105444273)

# 1. Введение

Часто возникают задачи, в которых требуется разбить некоторое изображение на несколько частей с выделением групп, классов по набору отличительных признаков. В современной науке за этот процесс отвечает термин сегментация и находит свое применение в ряде задач, например, таких как поиск и распознавание объектов, нахождение границ и редактирование изображений. Подобные задачи стояли и в практической работе при изучении нейронных сетей. Тем не менее, сам процесс сегментации представляет собой разбиение изображения на области, которые должны обладать одинаковыми признаками. Такое решение находит свое применение, в том числе в медицине, где необходимо выделить участки изображения со здоровыми или пораженными органами, тканями и т.п.

С развитием технологий закономерно появилось несколько способов сегментировать изображения, причем как черно-белые, (особенно полутоновые изображения), так и цветные изображения. В медицине же изображения, как правило, подразумеваются в виде полутона с глубиной цвета от 16 бит, так как меньшего количества диапазона возможных значений яркости не всегда может быть достаточно [3].

Нужно отметить, что работа по сегментации может быть разделена на два следующих вида:

* Интерактивная
* Автоматическая

Отличие заключается, в том, что при интерактивной сегментации требуется ручной труд человека, в то время как при автоматической этот труд можно заменить алгоритмами, в чем и состоит главное ее преимущество. Основное внимание в данной работе будет уделено видам автоматической сегментации изображений. Среди всего разнообразия данных методов [4], особенно можно выделить:

* Пороговая сегментация
* Наращивание областей
* Выделение границ
* Текстурные методы

Все методы не далеко уходят от стандартных решений при обработке изображения. Самый простой из всех приведенных выше способов – пороговая сегментация. Суть работы заключается в том, чтобы выделить однородные по интенсивности участки с использованием некоторого порога, определяемого автоматически специальным алгоритмом. В случае применения метода наращивания областей, участки близкие по яркости будут присоединяться в один большой существенный. Выделение границ же подразумевает непосредственное выделение контуров изображения, подобные задачи решает множество фильтров, классическим примером может быть матричный фильтр оператор Собеля. Суть работы таких алгоритмов сводится, в грубом смысле, поиска производной, градиента изображения, выражаясь более мягко, поиск перепадов значений пикселей, что соответствует значениям максимума и минимума кривой. Текстурные методы по большей части применимы к цветным изображениям [2], как следует из названия, они ориентируются на диффузные свойства поверхности анализируемого объекта. Суть методов заключается в приведении изучаемых изображений к задаче различения уровней яркости, применяя специальные операторы, для этой цели.

*Оператор Собеля*

В дальнейшем речь пойдет о технологиях глубокого обучения в целях сегментации медицинских изображений, с полна реализующих преимущество заменить ручной труд человека на автоматическую работу нейронной сети.

# 2. Постановка задачи

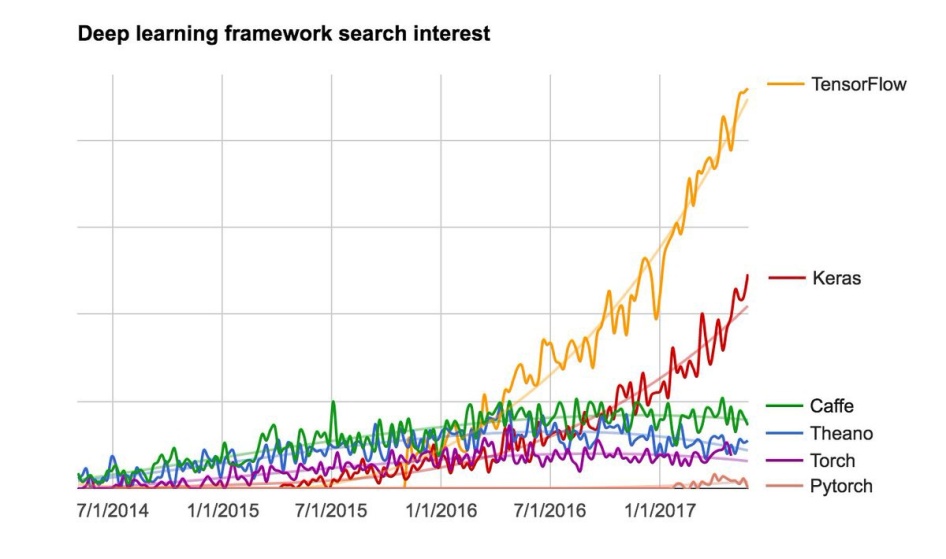
В данной практической работе стоит задача изучить язык программирования Python, основное внимание уделить широким возможностям библиотек, написанным для ускорения вычислений, таким как NumPy. Познакомиться с технологией глубокого обучения, исследовать возможности библиотек TensorFlow и Keras, разработанных на Python для удобного конструирования и обучения нейронных сетей. Создать и обучить свою собственную нейронную сеть. Изучить основные виды нейронных сетей, использующихся в области глубокого обучения, некоторые способы предобработки изображений при использовании сверточных нейронных сетей. Рассмотреть структуру и работу сети U-Net, созданной для сегментации медицинских изображений. Изучить способ хранения медицинских данных в виде Pydicom файлов. Создать первую сверточную нейронную сеть глубокого обучения на базе архитектуры U-Net, решив тривиальную задачу по обучению модели, после чего применить полученный опыт для более серьезного эксперимента в обучении моделей Keras. Проанализировать полученные результаты работы и сделать вывод.

# 3. Описание фреймворков TensorFlow и Keras

## **3.1 Возможности загрузки dataset**

Фреймворк Keras разработан для поддержки глубокого обучения в языке Python, при этом является более компактной и удобной надстройкой над библиотекой TensorFlow, набравшей высокую популярность за последнее время по сравнению с со своими аналогами (рис. 3.1), что благоприятствует быстрому созданию и анализу нейронных сетей [1]. Основными характеристиками можно считать:

* Поддержка переносимости или возможность запустить код на GPU и CPU без изменений
* Дружественный API, упрощающий разработку прототипов моделей глубокого обучения
* Встроенная поддержка основных видов нейронных сетей: прямая, сверточная, рекуррентная.



*Рис. 3.1 Рост количества поисковых запросов для разных фреймворков*

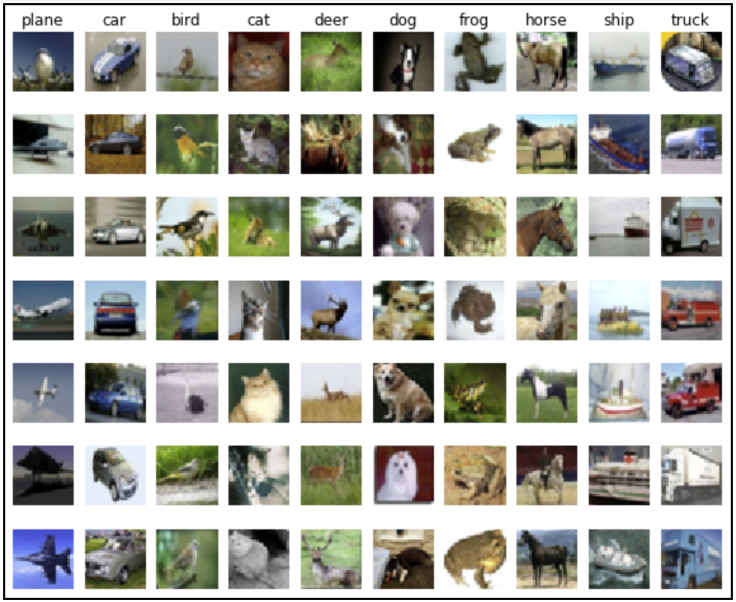
*глубокого обучения по данным Google*

Помимо вышеперечисленного, библиотека Keras обладает еще достаточно большим рядом преимуществ. В данном разделе речь пойдет о встроенных datasets прямо в Keras, и о том, как можно использовать для обучения нейронной сети сторонний dataset.

Итак, для работы с нейронными сетями в Keras предусмотрена поддержка некоторых полезных наборов данных, таких как:

* CIFAR10
* CIFAR100
* IMDB
* Reuters
* MNIST
* Boston\_housing

Наборы данных CIFAR содержат по 50 тыс. каждый учебных изображений размера 32 на 32 пикселя, помеченных в 10 и 100 категориях соответственно.



*Рис. 3.2 Набор данных CIFAR*

Dataset IMDB содержит 25 тыс. отзывов о фильмах с платформы IMDB, разделенными бинарными метками по их настроению, в виде прямой кодировки самых популярных в отзывах слов (каждое слово кодируется уникальным целым положительным числом) распределенных по убыванию их популярности.

Так же есть такой набор данных, как классифицированные темы новостных лент на 46 видов. Всего в базе содержится более 11 тыс. новостных лент журнала Reuters. Представление данных такое же, как и в IMDB – прямое кодирование.

Самый классический набор данных – это MNIST, содержит 60 тыс. полутоновых изображений 28 на 28 пикселей, помеченных по виду одной из десяти уникальных арабских цифр на изображении. В dataset в том числе включен набор тестовых изображений, не содержащий меток.

Дополнительный набор данных – boston\_housing, включающий в себя 13 атрибутов, описывающих некоторое состояние рынка цен на дома в пригороде Бостона в конце 70-ых годов прошлого века. Источником данных является библиотека StatLib, представленная в университете в Карнеги-Меллон.

Указанные стандартные datasets можно загрузить, используя небольшую конструкцию программы на языке Python

from keras.datasets import. <name of dataset>

(x\_train, y\_train),(x\_test, y\_test) = <name of dataset>.load\_data()

Кроме того, данные наборы можно загрузить сразу в более удобном виде, для этого есть инструменты, использующиеся как аргументы функции load\_data(). Например, наиболее применимыми из них являются возможности: загрузить данные в перемешанном виде, с усечением последовательности, с разделением для резервирования данных в качестве тестового набора.

В практической работе, были рассмотрены и использованы все виды встроенных в Keras datasets. Написанный в рамках работы код располагается на GitHub, см. приложение.

Помимо встроенных datasets, Keras располагает возможностью использовать независимые от него наборы данных. В практической работе было рассмотрено два способа:

1. Использование директории, в которой располагаются тренировочный набор изображений.

2. Таблица excel, заполненная тренировочными данными.

В обоих случаях данные были взяты из открытого источника, сайта для соревнований в области глубокого обучения Kaggle. Примеры кода, см. приложение.

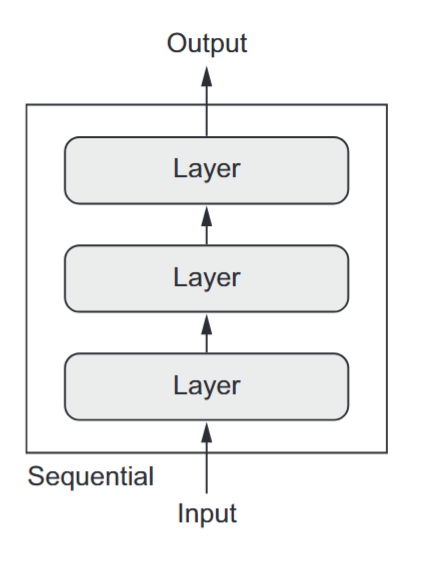
## **3.2 Возможности Keras в построении моделей**

Существует несколько основных семейств архитектур моделей нейронных сетей, поддерживаемых в Keras:

* Полносвязные,
* Сверточные,
* Рекуррентные,
* Энкодер-декодер,
* Трансформеры и др.

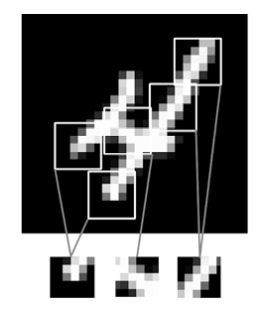
Данные типы вбирают в себя некоторые важные характеристики, позволяющие эффективно применять их в различных задачах, таких как бинарная классификация, множественная классификация, однозначная или многозначная, задача регрессии (предсказание значения по предыдущим показаниям). Решение каждой задачи есть построение пространства гипотез, в котором производится поиск наилучшей подходящей модели предсказателе. Выбор архитектуры полностью зависит от вида стоящей задачи.

Например, при работе с векторными данными наилучшим выбором будет использовать полносвязную нейронную сеть. Она представляет собой набор слоев Dense, которые строят свои выводы исходя не из структуры входных данных, а из их наличия. То есть, их работа заключается в поиске отношения между двумя любыми присутствующими во входных данных элементами. Они проводят качественный анализ данных и часто становятся неотъемлемой частью в построении более сложных моделей и используются в качестве их верхних слоев. На практике они применялись почти в каждом примере, и только их использование было рассмотрено для решения каждой из вышеперечисленных задач. Тут стоит выделить специальные настройки под ту или иную задачу. Для бинарной классификации нужно использовать функцию потерь binary\_crossentropy и функцию активации sigmoid. Вторая позволяет достичь на выходе распределение вероятности от 0 до 1, означающее принадлежность к определенному классу. Для однозначной классификации правильно использовать функцию потерь categorical\_crossentropy и функцию активации softmax. В случае многозначной классификации функция categorical\_crossentropy заменяется на binary\_crossentropy. Для задачи регрессии следует использовать функцию потерь средней квадратичной ошибки или средней абсолютной ошибки, mae и mse соответственно. Функция потерь при этом не требуется.



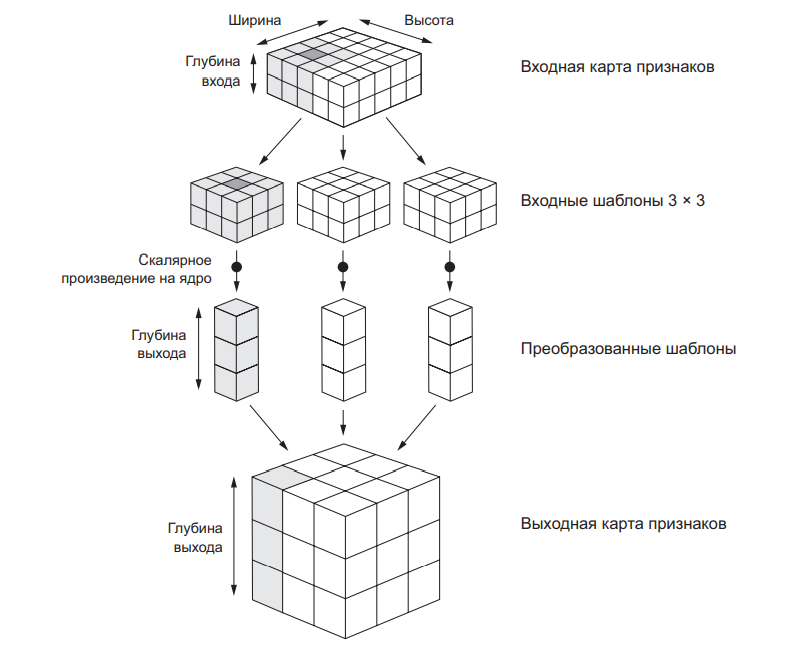
*Рис. 3.3. Схематический вид полносвязной нейронной сети*

Для работы с многомерными тензорами данных наилучшим выбором будет использование многомерных сверточных сетей. Они представляют собой несколько последовательно выстроенных слоев Conv (свертка) и MaxPooling любой из доступных размерностей (1D, 2D, 3D). В отличие от слоев полносвязных сетей, они проводят оценку локальных шаблонов, сочетаний значений элементов, что позволяет найти выделенный шаблон в любом месте данных. Такое свойство называется инвариантностью относительно переноса. Таким образом, они позволяют сделать эффективную пространственную выжимку информации из входных данных. Далее, как было подмечено ранее, полученные шаблоны часто передаются в слои полносвязной сети для дальнейшей классификации.



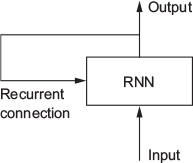
*Рис. 3.4 Разбиение изображения на локальные шаблоны*

Принцип работы свертки заключается в проходе многомерного вектора по тензору данных и применения операции скалярного произведения к соответствующим ей элементам этих данных. В случае двумерного тензора (часто просто изображения) эта операция соответствует матричному произведению. В компьютерной графике такой двумерный вектор назывался ядром, в глубоком же обучении носит название фильтра. Таких фильтров в слое сверточной сети может быть не мало, часто степени двойки. На выходе получается некоторое множество изображений с выделенными на них признаками изображения. Эти изображения называются картами признаков. Также, естественно, при использовании матрицы размера больше единицы в операции свертки, карта признаков уменьшается. Во избежание этого процесса есть специальные методы дополнения границ исходного изображения, но в целом, это используется не часто, так как суть сверточных сетей выделить главную информацию, для чего, и предназначена следующая операция – MaxPooling. Данный слой производит выбор максимальных значений по соседям, что снижает размерность данных, что и необходимо для сохранения размеров карты признаков в разумных пределах с ростом числа признаков. Это позволяет последующим сверточным слоям анализировать пространство входных признаков на большем протяжении.



*Рис. 3.5 Принцип работы свертки*

Последним из рассматриваемых является класс рекуррентных нейронных сетей. Наиболее эффективно их использование при обработке последовательности данных, основанной на существенной зависимости от параметра времени. В таком случае, использование сверточных сетей уже не является эффективным, так как данные не обладают свойством инвариантности относительно переноса. В Keras рекуррентные сети представлены слоями SimpleRNN, GRU, LSTM. На их выходе можно получить либо один вывод результирующей последовательности, либо несколько последовательностей от каждого из слоев. На практике они были применены в ходе предсказания температуры на следующие дни по метеорологическим данным.



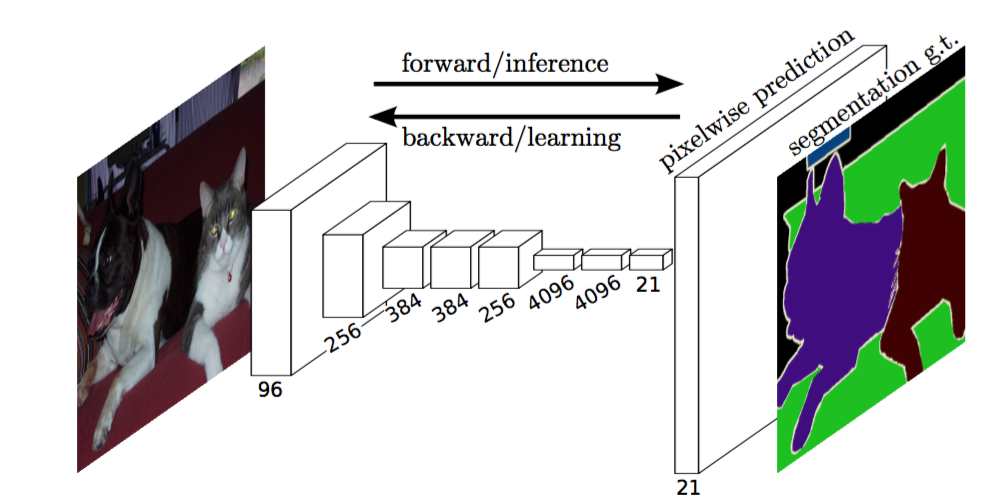
*Рис. 3.6 Принцип работы рекуррентной нейронной сети*

## **3.3 Популярные модели сегментации**

Модели сегментации в глубоком обучении обычно строятся на основе предварительно обученной на наборе ImageNet нейронной сети. В таком случае применяется заморозка сверточной основы, а верхние слои классификатора, сегментирования начинают свое обучение сначала – со случайных весов. Очень удобно, что в Keras есть возможность выбрать из уже готовых, предварительно обученных для этих целей сверточных нейронных сетей. Например, таких как ResNet – модель от Microsoft, она имеет большое количество слоев и оставшихся соединений, VGG-16, более легкая сеть, так как меньше количество слоев, использовалась на практике, MobileNet – модель от Google, наиболее оптимизирована для работы на устройствах с ограниченными ресурсами, по типу мобильных телефонов.

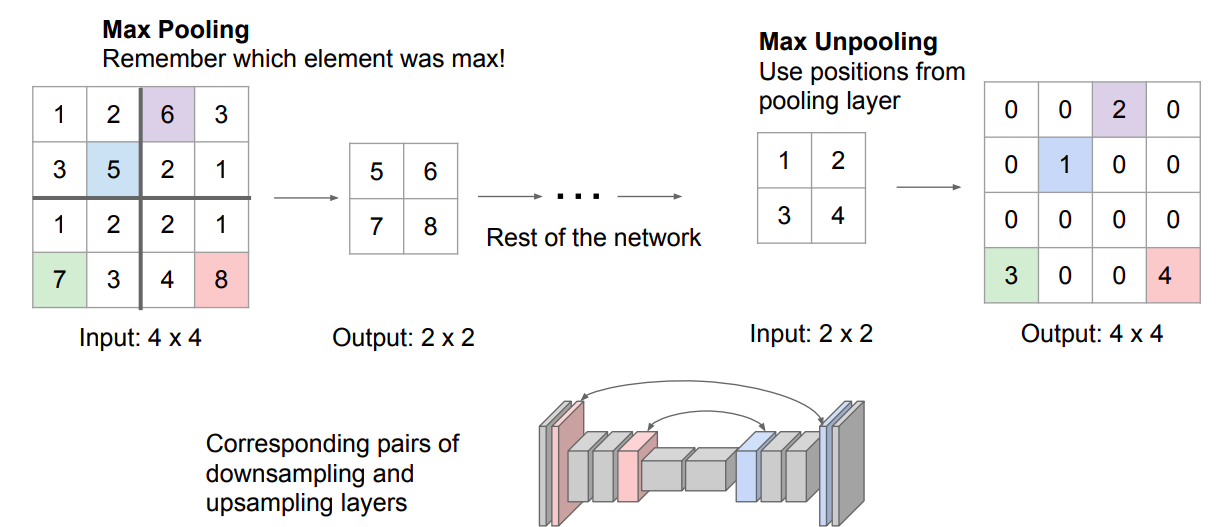
В Keras возможна реализация следующих моделей сегментации [5-8]:

1. FCN – полностью сверточная нейронная сеть, самая простая и популярная модель семантической сегментации. Суть работы заключается в уменьшении входного изображения до меньшего размера, тем самым получая увеличенный вектор пространства каналов, при помощи серии сверток. Такой набор сверток обычно именуется кодером. Затем, выход работы кодера подвергается дискретизации при помощи билинейной интерполяции или серии транспонированных сверток. Эта операция выполняется декодером. Существенным минусом в работе транспонированной свертки является появление артефактов шахматной доски из-за неравномерного перекрытия выходных данных. Еще одним недостатком служит разрешение на границах из-за потери информации в процессе кодирования.



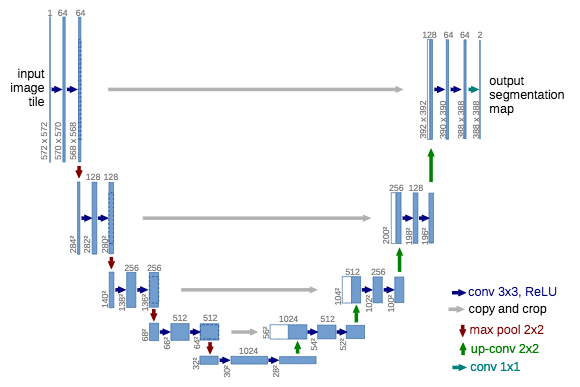
*Рис. 3.7 Архитектура FCN*

2. SegNet – основана на идее кодера и декодера, как и FCN, однако, оптимизация состоит в том, что их уровни симметричны друг другу. Операция дискретизации уровня декодера использует максимальный индекс пула соответствующего ему уровня кодера. В отличие от U-Net еще не использует пропуск соединений. Для повышения частоты дискретизации нет обучаемых параметров.



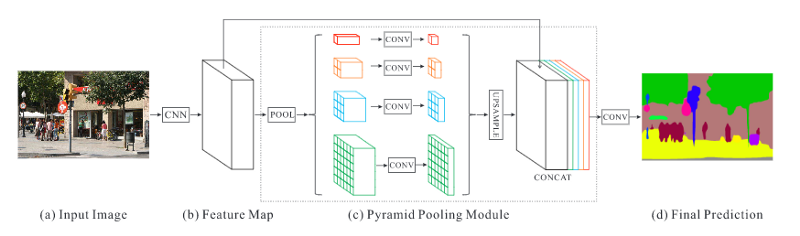
*Рис. 3.8 Архитектура SegNet*

3. U-Net – использует идею FCN, однако также предусматривает решение тех проблем декодера, которые были отмечены выше. Предполагает наличие симметричной обработки данных с использованием пропуска соединения с выхода блока свертки в соответствующий ему блок транспонированной свертки на том же уровне. Добавление пропуска соединения позволяет лучше представлять информацию из нескольких масштабов изображения. Информация с верхних уровней помогает модели лучше классифицировать участки изображения, в то время как с мелких уровней она помогает сегменту лучше локализироваться.



*Рис. 3.9 Архитектура U-Net*

4. PSPNet – сеть пирамидального распознавания сцены сочетает в себе 4 свертки разного масштаба пирамиды. Предварительно обученная модель базового слоя, в которой применение свертки извлекает карту признаков из входного изображения 1/8 его размера. Далее происходит согласованное объединение всех уровней свертки и повышенной выборки. В конце полученную структуру необходимо еще раз прогнать через сверточный слой.



*Рис. 3.10 Архитектура PSPNet*

## **3.4 Расширенные возможности оптимизации процесса обучения**

В Keras существует несколько инструментов, способных улучшить процесс обучения моделей [1]. Они позволяют существенно сократить время разработки хорошей нейронной сети и тем самым повысить ее конечное качество. Разберем несколько таких оптимизирующих механизмов, представленных внутри Keras.

1. Callback – объект класса, который обладает всей информацией о состоянии обучения нейронной сети и при этом может влиять на него. Данный механизм позволяет экономить время и ресурсы, в случае однозначно неудачной архитектуры модели, сохранять удачные попытки подбора весов и следить за обучением на более качественном уровне. Вот, некоторые возможности, которые он предоставляет:

* Зафиксировать состояние модели на определенных этапах обучения, т.е. сохранить веса непосредственно в процессе обучения
* Приостановить обучение модели в случае череды неудач – длительного роста значений функции потерь
* Динамически корректировать некоторые гиперпараметры обучения, такие как шаг обучения оптимизатора
* Журналирование оценок функции потерь на проверочных данных с целью дальнейшего изучения протекания процесса обучения

Причем уникальность механизма в том, что callback можно создавать самому с нужными характеристиками под конкретную задачу. Более того, можно создавать сразу несколько callback в виде списка объектов класса callback, например:

callbacks\_list = [keras.callbacks.EarlyStopping(

                    monitor = "val\_acc",

                    patience = 1),

                    keras.callbacks.ModelCheckpoint(

                    monitor = "val\_loss",

                    filepath = "my\_model.h",

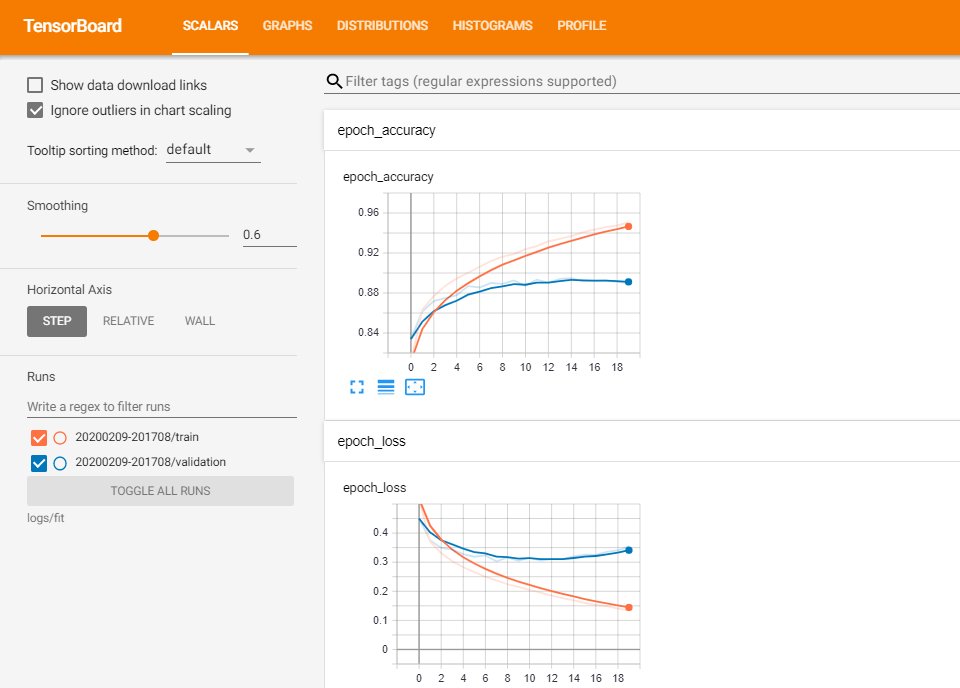
                    save\_best\_only = True)]

В данном коде EarlyStopping обратный вызов, который прервет обучение в случае, если метрики не улучшатся в течение 2 эпох (параметр patience). Второй обратный вызов в списке ModelCheckpoint сохраняет метрики в ходе обучения, причем только в том случае, если метрики улучшились (параметр save\_best\_only). В случае необходимости создать свой callback, достаточно просто создать свой класс, унаследованный от стандартного класса Callback.

2. Главная цель экспериментов с обучением нейронной сети – получение информации о качестве работы модели. С этой задачей может помочь справиться полезный инструмент TensorBoard. Его работа поддерживается внутренним браузером фреймворка TensorFlow. Данный механизм обеспечивает поддержку следующих возможностей:

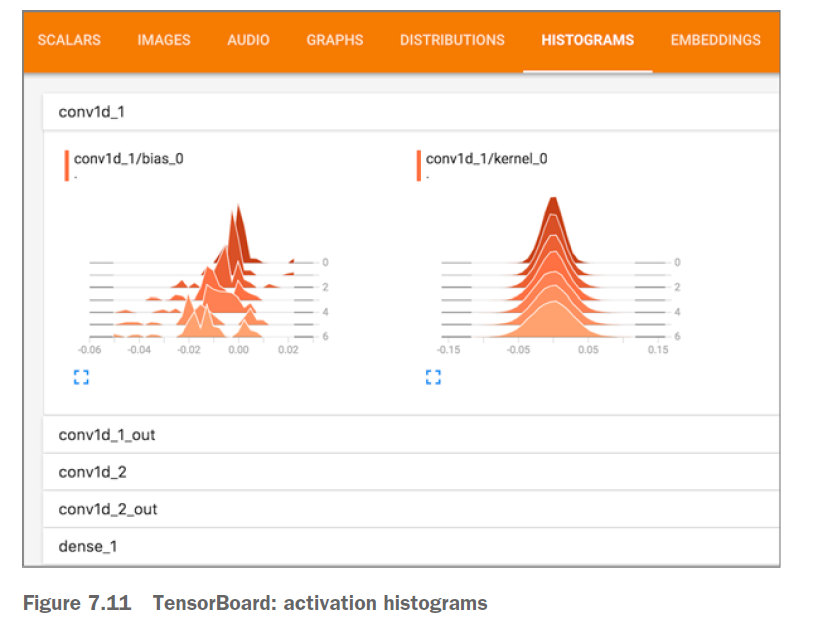
* Мониторинг метрик в ходе обучения
* Визуализация архитектуры модели
* Вывод гистограмм активаций и градиентов
* Исследование векторных представлений в трехмерном пространстве

Использовать данный механизм можно в виде обратного вызова. Далее, передать его в функцию обучения модели. Расположение вывода работы будет располагаться на локальном адресе, соответственно достаточно воспользоваться браузером. Главная страница TensorBoard демонстрирует динамически обновляющиеся графики метрик, получаемых в ходе обучения и проверки на проверочных данных.



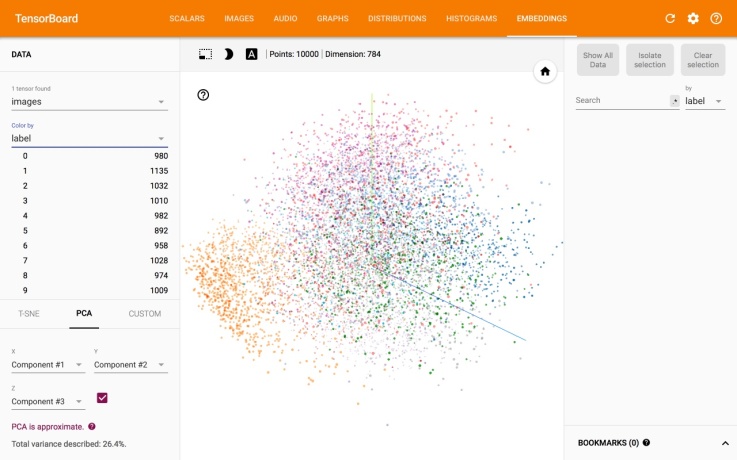
*Рис. 3.11 Мониторинг метрик*

Помимо этого, на вкладке Histograms располагаются гистограммы активаций.



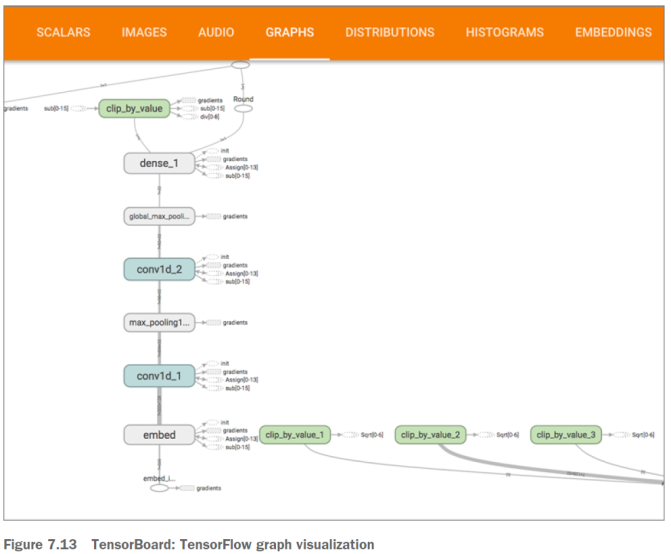
*Рис. 3.12 Гистограммы активаций*

В случае работы с векторным представлением слов, можно получить красивый график пространственного отношения всех слов во входном словаре. Удобство TensorBoard в том, что она способна автоматически понизить размерность пространства для удобства работы с помощью алгоритма снижения размерности по выбору программиста.



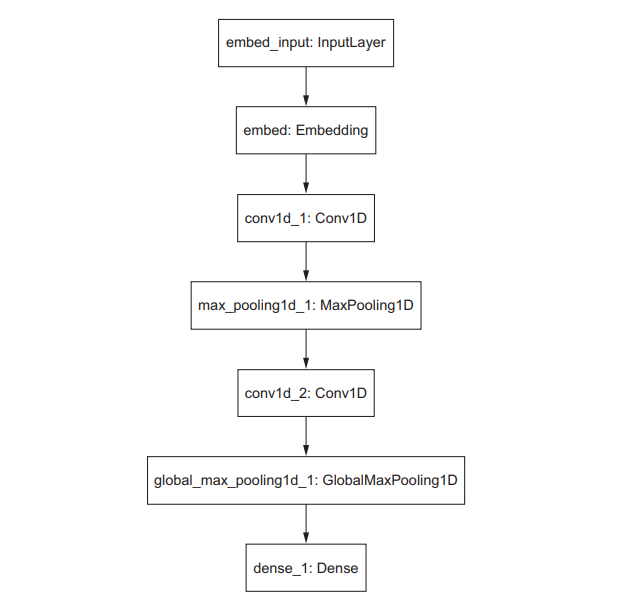
*Рис. 3.13 Интерактивная карта трехмерного пространства векторных представлений слов*

На вкладке Graphs можно наблюдать всю полноту низкоуровневого построения модели, которая происходит в TensorFlow при создании модели.



*Рис. 3.14 Граф операций TensorFlow*

3. Подобные представления о структуре нейронной сети можно получить другим механизмом, который поддерживает Keras – утилиты keras.utils.plot\_model, она позволяет получить изображение графа слоев модели. Пример того, как это будет выглядеть представлен на рис. 3.15.

**

*Рис. 3.15 Модель в виде графа слоев*

Таким образом, фреймворк Keras содержит несколько удобных инструментов для работы с обучением нейронных сетей, способных вывести процесс обучения и проверки на качественно новый уровень.

# 4. Предварительная обработка изображений

Процедура предварительной обработки данных используется практически всегда после получения информации с датчика и заключается в операции усреднения и выравнивания гистограмм, применении различного типа фильтров для исключения помех, возникающих в результате аппаратной дискретизации и квантования и для подавления внешних шумов. Изображения, подлежащие такой обработке, можно разделить на 3 типа:

* Многоуровневые и бинарные изображения
* Бинарные изображения или битовые карты изображений
* Изображения характерных линий контуров или скелетов объектов на изображении

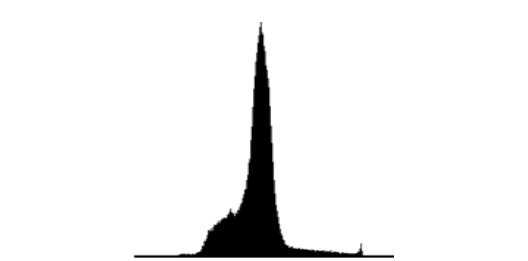
Самым распространенным и сложным для анализа является первый тип, в то время как два других идут на встречу упрощения этого процесса.

С одной стороны задача подавления помех входит в задачу улучшения изображения, а с другой – ее можно рассматривать как часть задачи сегментации [9]. Так же как необходимо отделить шум от изображения, так и в задаче сегментации необходимо отделить объекты от области фона. Данные операции являются трудоемкими, в связи с тем, что понятия «объект», «шум», «фон», «граница объекта» условны. Методы и алгоритмы сегментации можно рассматривать, как формализацию понятия выделяемости объекта от фона.

Влияние предварительной обработки может существенно зависеть от способа сегментации изображения. Например, часто для улучшения работы метода пороговой сегментации изображения применяют инвертирование изображения. Более того, данная предобработка полезна для использования в непосредственно в глобальной обработке, причиной тому способствует тот факт, что операция инвертирования помимо инвертирования цвета, в том числе инвертирует интенсивность, что делает светлые участки изображения темнее, а темные наоборот – светлее. Это в свою очередь в лучшую сторону влияет на работу алгоритма глобальной пороговой обработки. Из курса компьютерной графики известна формула, по которой можно осуществить процедуру инвертирования:

где - значение яркости пикселя изображения с координатами , а – новое значение яркости этого пикселя.

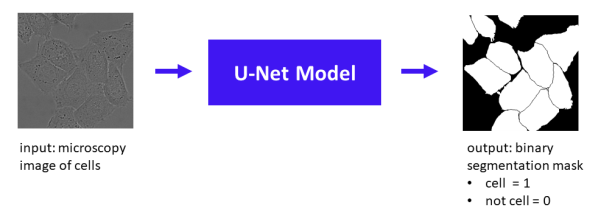
Однако, процедура инвертирования не всегда улучшает работу данного алгоритма, в том случае, если гистограмма изображения представляет собой очень сложную форму (рис. 4.1), то тогда инвертирование не имеет смысла, поскольку сложность гистограммы не изменится и корректность работы фильтра будет оставлять желать лучшего. Более того, как показывает практика, применение этой предобработки для других методов сегментации не дает сильного улучшения работы алгоритма.



*Рис. 4.1 Пример сложной гистограммы*

# 5. Архитектура модели глубокого обучения U-Net

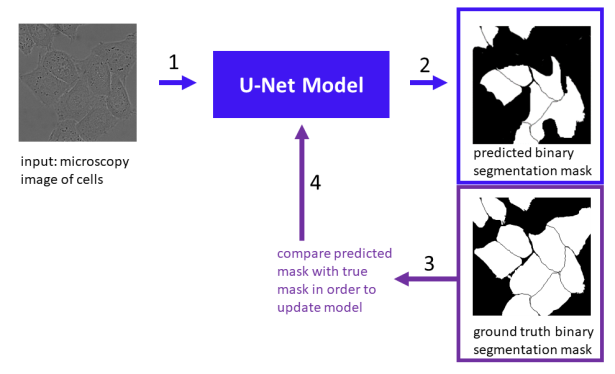
Оригинальная статья U-Net, набравшая огромную популярность за последние 6 лет, описывает архитектуру модели семантической сегментации [7]. Причем, изначально она была описана в контексте биомедицинских изображений, но впоследствии развития была также успешно применена и к другим видам изображений. Ключевая идея работы модели U-Net заключается в получении бинарной маски сегментации (рис. 5.1) [10, 11].



*Рис. 5.1 Принцип работы U-Net архитектуры*

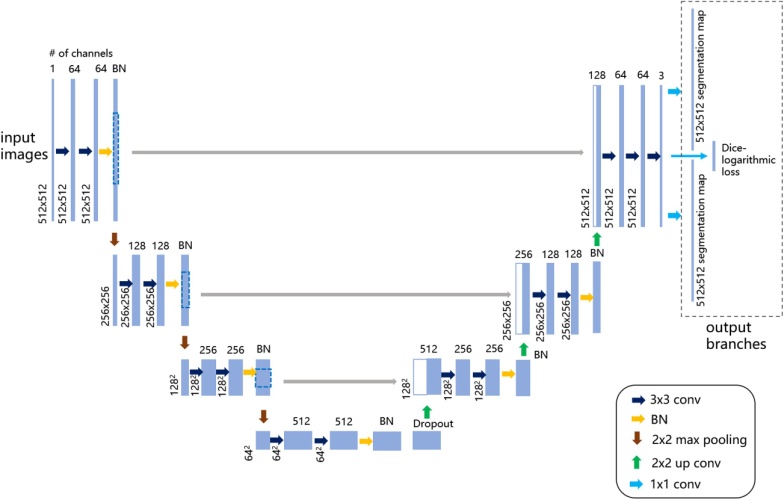
В данном примере входным изображением является полутоновое изображение клеток, сделанное с помощью микроскопа. Бинарная маска, которая получается на выходе, кодирует 0 как элементы фона изображения, в том числе и границы между клетками, а 1 как сами клетки. Отметим, что данный пример иллюстрирует семантическую сегментацию, так как выделенных участков всего два: передний план и задний. В типичной задаче сегментации есть необходимость разделить передний план на еще несколько классов.

Обучение модели заключается в том, что, сравнивая выходную бинарную маску предсказания и истинную бинарную маску необходимо добиться уменьшения значений функции потерь, за счет изменения весов механизмом обратного распространения ошибки (см. рис. 5.2).



*Рис. 5.2 Схема обучения U-Net модели*

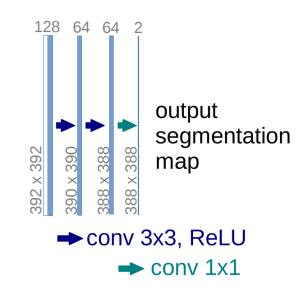
На рис. 5.3 можно наблюдать саму архитектуру U-Net, как демонстрирует название, она имеет форму в виде буквы U. Сначала U-Net получает низкое разрешение исходного изображения, сжимая его при помощи классической сверточной нейронной сети. Затем, увеличивает образец, чтобы получить на выходе сегментированное изображение. В правом нижнем углу рисунка 5.3 есть описание, какому типу операций соотносится каждая стрелка в архитектуре U-Net.

**

*Рис. 5.3 Архитектура U-Net*

Первая часть архитектуры или кодер, связан со второй частью или декодером серыми стрелками, которые обозначают операцию копирования. Ее работа заключается в том, что выходная карта признаков кодировщика каждого уровня копируется и соединяется (операция конкатенации) с картой признаков соответствующего ему уровня декодера, что помогает нейронной сети обучаться сегментации высокого разрешения.

Рассмотрим работу последнего уровня декодера. В самом конце сети данные проходят через слой свертки 1 на 1 пикселей с отображением 64 каналов вектора признаков в необходимое количество классов. В данном примере рассматривается два класса, поэтому последний слой будет выглядеть так:

**

*Рис. 5.4 Последний слой архитектуры U-Net при семантической сегментации*

Для получения семантической сегментации изображения используется функция активации pixel-wise softmax:

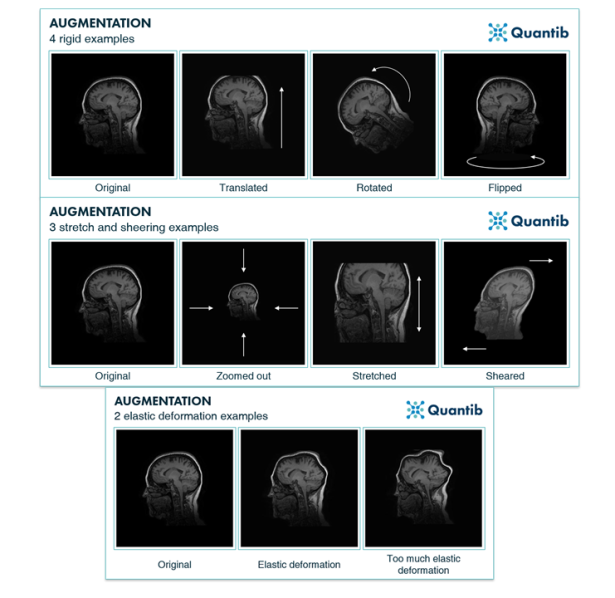
где – значения активаций в канале признаков *k* пикселя с координатойи *–* количество нужных классов.

В качестве функции потерь используется cross-entropy loss, которая имеет вид:

– весовые коэффициенты, – активации последнего слоя, т.е. предсказание модели.

Весовые коэффициенты вычисляются автоматически с использованием полученных ранее истинных сегментированных изображений при помощи стандартных методов компьютерного зрения. В данном примере дополнительно применяется математическая морфология для выявления тонких границ между клетками, которые сильно увеличивают весовые коэффициенты. Данное новшество позволяет изменять веса модели таким образом, чтобы эти границы действительно присутствовали и были нарисованы в нужных местах. В общем, весовые коэффициенты в функции потерь служат цели выявления границ между клетками.

К сожалению, построение и обучение такой модели сегментации может быть очень время затратным, так как присутствует необходимость практически вручную нарисовать истинное сегментированное изображение. К тому же, набор таких данных может быть ограниченным. Для того, чтобы справиться с этой проблемой можно воспользоваться методами расширения тренировочных данных. Один из таких состоит в том, чтобы применить случайные деформации к изображению, такие как сдвиги пикселей, вращения изображения, изменения оттенков серого или другие эластичные преобразования. Эластичные преобразования наиболее эффективны в случае медицинских изображений, потому что они устойчивы к такому виду деформации, и все еще выглядят реалистично после таких изменений.



*Рис. 5.5 Эластичные деформации*

Таким образом, модель U-Net достигла больших успехов не только в области медицинских изображений разного рода, но и в других видах сегментации изображений.

# 6. Обзор внешних инструментов для обучения моделей

## **6.1 Google Colaboratory**

Так как обучение сверточной нейронной сети трудоемкий процесс, требующий большого объема вычислений, то производить их на обычном процессе неприлично долго. С этой задачей в разы лучше могут справиться дискретные графические процессоры, при этом, за счет переносимости, даже не придется менять код. Да, на данный момент существует еще один способ обучать сложные модели – тензорный процессор, разработанный специально компанией Google для машинного обучения [12]. Он еще более производителен, чем GPU, но для его использования придется изменять код, так как он требует специальной предобработки тренировочных данных.

В наши дни позволить себе подходящий для машинного обучения процессор может не каждый. Поэтому на наш запрос о бесплатном использовании графических процессоров в некоммерческих целях полностью отвечает система Google Colaboratory. Она предоставляет возможность бесплатно и непрерывно использовать их с ограничением в 12 часов. Ключевое преимущество графических процессоров, используемых в системе, – это поддержка программно-аппаратной архитектуры параллельных вычислений от компании Nvidia [13]. CUDA позволяет организовывать доступ к набору инструкций графического ускорителя и управлять его памятью. Версия CUDA на момент написания главы.

nvcc: NVIDIA (R) Cuda compiler driver

Copyright (c) 2005-2020 NVIDIA Corporation

Built on Mon\_Oct\_12\_20:09:46\_PDT\_2020

Cuda compilation tools, release 11.1, V11.1.105

Build cuda\_11.1.TC455\_06.29190527\_0

Работа с Google Colaboratory происходит при помощи интерактивной облачной среды вида Jupyter Notebook для Python, которая позволяет совместить выполнение кода, написание mark down текста, рисование matplotlib графиков, работа с Tensor Board сразу в одном проекте. Хранилище используется также облачное – Google Drive. Кроме того, в Google Colaboratory изначально предустановлена библиотека TensorFlow и Keras.

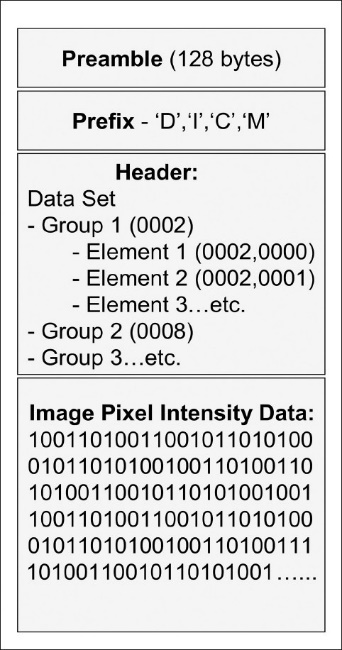
Одним словом, очень полезный инструмент для обучения сверточных нейронных сетей. Единственная проблема, с которой можно столкнуться – Google приостанавливает доступ к среде выполнения после 30 минут бездействия, в следствии чего нужно не забывать возвращаться в блокнот во время процесса обучения модели.

## **6.2. Описание Pydicom**

DICOM (Digital Imaging in Medicine) – это очень удобный, принятый за основу формат для хранения и обмена медицинскими изображениями. Поэтому очень важно уметь с ним работать. Этой цели и служит библиотека Pydicom.

Главное, с чем приходится работать инженеру нейронных сетей – dataset. В DICOM файлах dataset описывается встроенным словарем языка Python, где ключами являются теги DICOM, а значениями соответствующие им экземпляры класса DataElement [17, 18]. Это позволяет хранить полезные метаданные о dataset, доступ к которым легко получать напрямую с помощью тегов. Например, файл обычно содержит в таком виде данные о пациенте, сами изображения как множество значений пикселей, информацию о медицинском аппарате, произведшим снимок, и многое другое.

Типичная структура DICOM файла представлена на рис. 6.1.



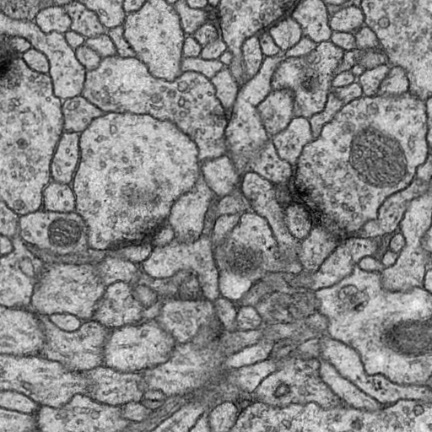
*Рис. 6.1 Структура файла DICOM*

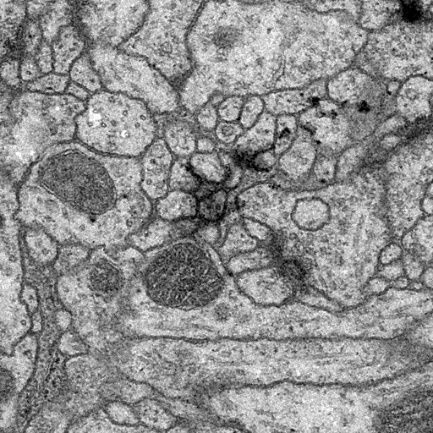
Стоит отметить, что формат предназначен для хранения 16 битовых изображений, диапазон значений пикселей варьируется от до . Поэтому по умолчанию Pydicom считывает данные пикселей в виде байтов. Однако, так как в одном файле DICOM могут содержаться пиксели разных типов данных (integer, unsigned integer, float), одно или несколько изображений, да еще и разных размерностей (2D, 3D), то Pydicom предоставляет метод pixel\_array, возвращающий удобную интерпретацию этих данных в виде массива NumPy.

В следующей главе нам предстоит работать с DICOM файлами, содержащими по одному двумерному изображению.

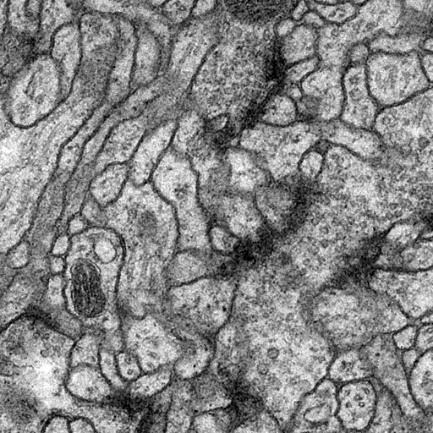
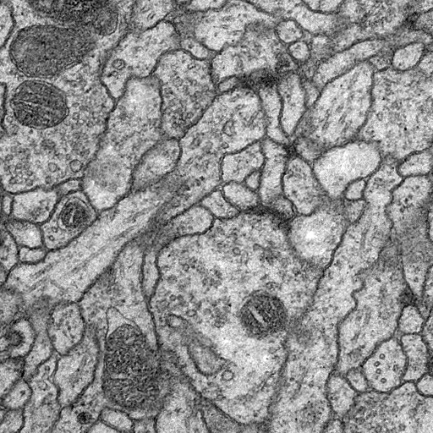
# 7. Первое обучение модели U-Net на медицинских данных

В качестве знакомства с работой нейронной сети на базе U-Net была выбрана одна из наиболее тривиальных проблем – сегментация изображений клеток мембраны, сделанных под электронным микроскопом. Цель задачи научить модель распознавать контуры, границы между клетками и строить свой прогноз в виде бинарной маски. Упрощению эксперимента благоприятствуют входные данные – они нормализованы, на них много ярких, броских контуров, что должно сильно помочь модели при обучении. Далее следует пара примеров изображений, содержащихся в этом наборе данных.

*Рис. 7.1 Тренировочные Image и Label из начального набора данных*

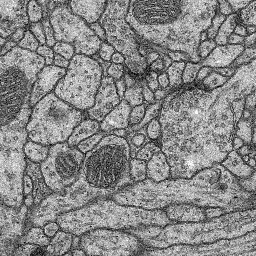
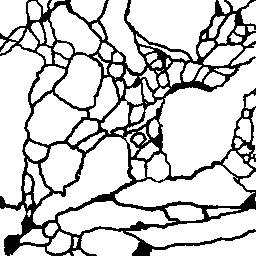
*Рис. 7.2 Изображения, отобранные для Test из начального набора данных*

Всего в тренировочном наборе данных содержится 30 изображений. Заметим, что этого может не хватить для хорошего результата. В связи с этим воспользуемся изученным в главе 5 методом расширения тренировочного набора данных – аугментацией или эластичными преобразованиями. Параметры деформации заданы следующим образом:

* Rotation range равен
* Width shift range равен
* Height shift range равен
* Zoom range равен
* Horizontal flip используется
* Fill mode в виде «nearest»

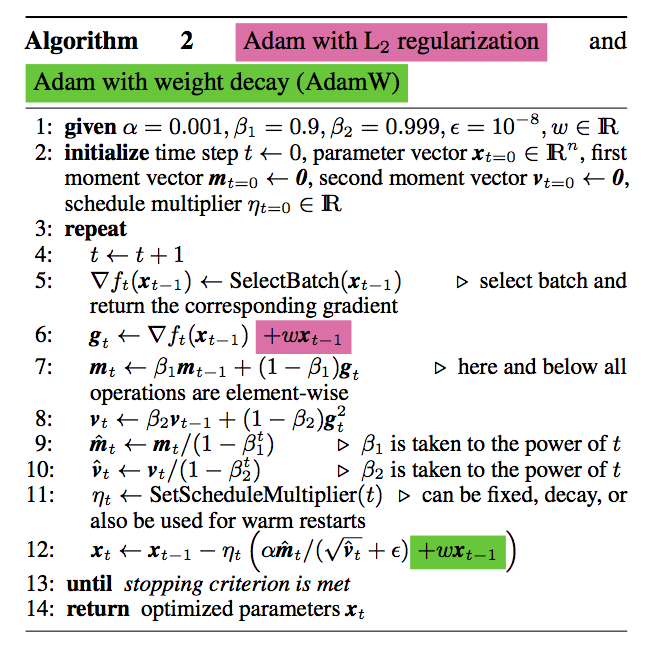
Rotation range – изменение изображения путем вращения его на случайный угол в диапазоне градусов. Width shift и height shift смещение по горизонтали и вертикали на случайный процент от размера изображения в пределах . Zoom range позволяет случайным образом пропорционально сжать или растянуть изображение в пропорции . Horizontal flip параметр отображения изображения по горизонтали. Fill mode указывает на то, каким образом заполнить недостающее пространство пикселей, образовавшееся при других трансформациях. Параметр nearest говорит о том, что пиксели будут заполняться в зависимости от ближайших оригинальных пикселей [16].

Визуализируем полученные таким методом изображения. Примеры полученных аугментированных изображений.

*Рис. 7.3 Аугментированные изображения для начального набора данных*

Теперь приступаем к настройке гиперпараметров обучения модели. Пусть генератор выдает тренировочные данные с параметром batch\_size = 20, количество эпох зададим равное 5, при этом параметр steps\_per\_epoch = 1000. Для сохранения весов (результата обучения) воспользуемся механизмом callback в Python, который сработает после каждой эпохи в том случае, если метрики на этом шаге не ухудшились. В качестве оптимизатора можно выбрать Adam – адаптивный оптимизатор, способный регулировать шаг в зависимости от градиента: большие шаги при малых изменениях градиента, малые шаги при быстрых, сильных изменениях градиента [14, 15]. Рассмотрим более подробно, как он устроен (см. рис. 7.4).



*Рис. 7.4 Алгоритм работы Adam (закрашенная часть – модификация AdamW)*

Оптимизатор Adam отлеживает средние значения градиента – первый момент и квадрата градиентов – второй момент . Эти показатели вычисляются на каждом шаге в зависимости от градиента по формулам.

Где – градиент в момент времени . Параметры и определяют затухание средних значений со временем, то есть чем дальше продвигается модель, тем меньше влияние старых средних значений. Затем шаг обучения корректируется согласно формуле.

Где – learning rate, – learning rate schedule multiplier. Тогда, если градиенты остаются приблизительно постоянными, то дисперсия градиентов приблизительно равна , а приблизительно равно . В таком случае и шаг обучения примерно равен . С другой стороны, если градиенты быстро меняются, тогда гораздо больше и шаг обучения гораздо меньше, чем .

В современном подходе к обучению модели принято использовать регуляризацию L2, которая основывается на идеи, что модели с меньшими весами в меньшей степени подвержены переобучению. Поэтому этот подход часто внедряют в оптимизатор или функцию потерь. Однако, для оптимизатора Adam регуляризация не эквивалентна уменьшению весов. В связи с чем инженеры машинного обучения разработали решение. Вновь обратимся к рисунку 7.4, но уже на закрашенные вставки от AdamW. Фиолетовым цветом выделен член регуляризации в оптимизаторе Adam, который участвует в вычислении новых градиентов. Проблема заключается в том, что он добавляется слишком рано, до вычисления средних значений , что приводит к тому, что модель начинает учитывать не только их, но и параметр регуляризации, в следствие чего не уменьшает веса, так как это предполагалось. Решение, добавленное в AdamW, вполне просто – перенести добавление члена регуляризации на момент после вычисления средних значений.

Итак, подобрать хороший адаптивный оптимизатор не самая простая задача. Кроме того, при обучении модели были встречены еще несколько проблем (см. рис. 7.5).

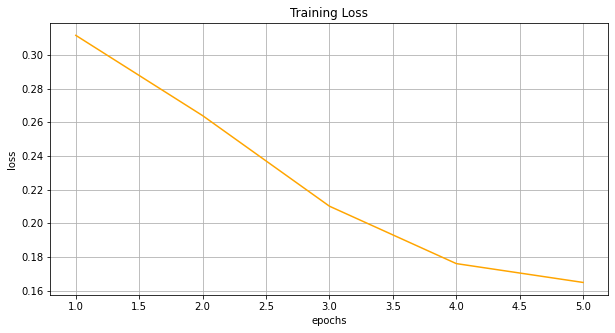
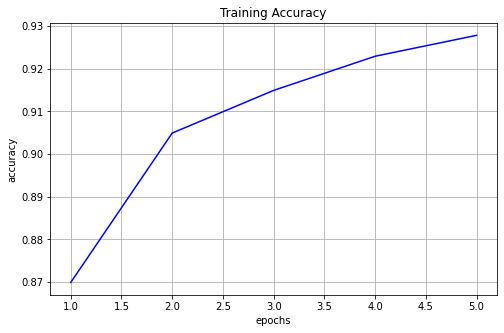
 

*Рис. 7.5 Примеры неудачных попыток обучения модели*

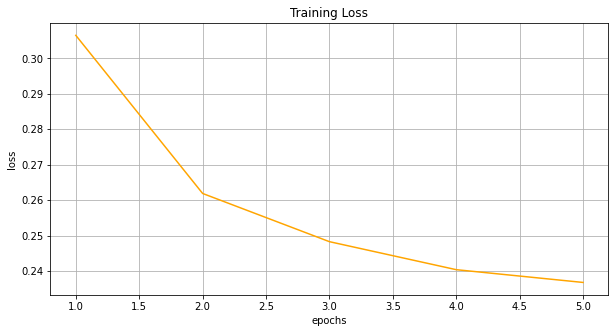
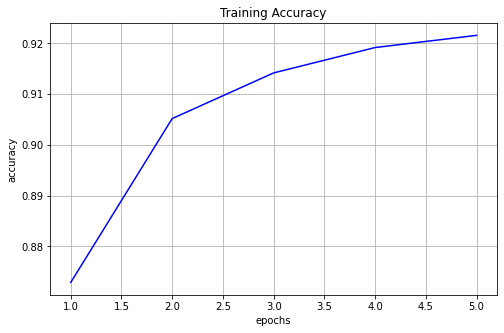
В первом случае обучение модели не происходит совсем, состояние остается ровно таким же, как при формировании случайных весов в самом начале, что является типичной ситуацией. Вторая ситуация сложнее, обучение проходит неверно, из-за чего веса просто концентрируются где-то сверху. Первая проблема была решена использованием параметра learning rate = , вместо learning rate = . Вторая проблема, вероятно, была вызвана ошибкой при составлении начальных данных, то есть маски не соответствовали исходным изображениям. Формирование нового тренировочного набора данных исправило ситуацию.

Для обучения использовался графический ускоритель, предоставляемый компанией Google в системе Google Colaboratory. На обучение модели ушло полтора часа. Результаты были сохранены в облачном хранилище Google Drive. В разделе листинг приведен листинг выполнения программного кода со значениями метрик по прохождению каждой эпохи. Код программы можно найти на GitHub (<https://github.com/PureEvil-UselessMan/SegmentMedImages>).

Приведем графики процесса обучения модели для разных оптимизаторов.

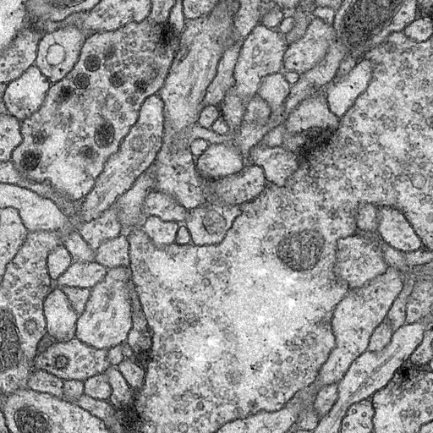
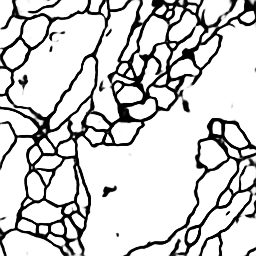
 

*Рис. 7.6 Потери и точность для Adam*

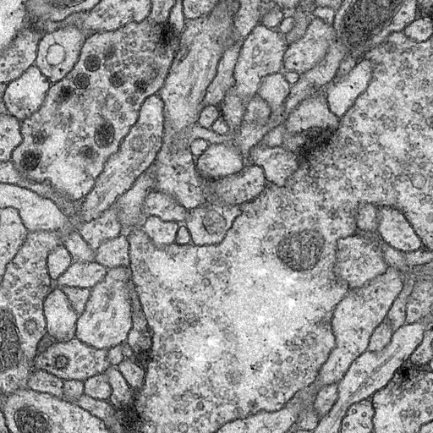
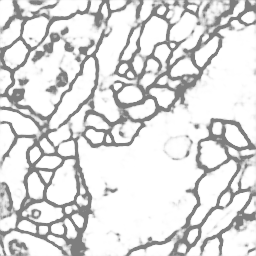
 

*Рис. 7.7 Потери и точность для AdamW*

После успешного обучения модели можно рассмотреть, на сколько точными получились ее прогнозы. Они представлены на рис 7.8 для Adam и на 7.9 для AdamW.

*Рис. 7.8 Предсказание модели для Adam, lr =*

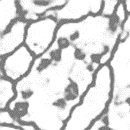
 

*Рис. 7.9 Предсказание модели для AdamW, lr = , wd =*

Как не трудно заметить (см. рис. 7.8, 7.9) при использовании стратегий оптимизатора Adam и AdamW бинарные маски получаются недостаточно корректными, включая в себя несколько лишних деталей (рис. 7.10).

Лишние детали

Лишние детали

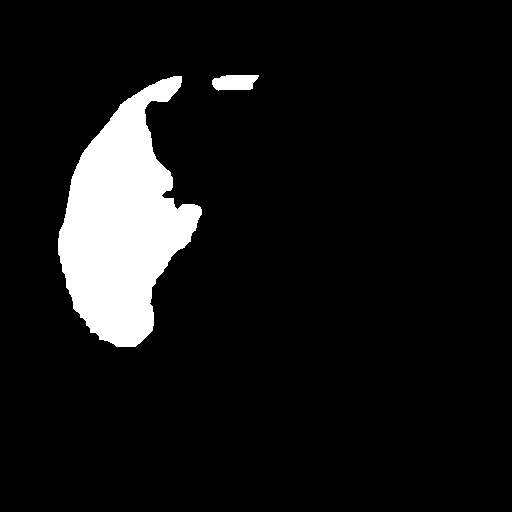
*Рис. 7.10 Попиксельное сравнение предсказаний Adam и AdamW*

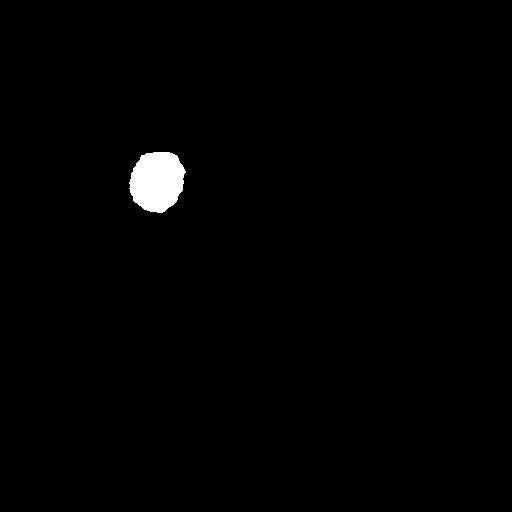
Но, тем не менее, разница на лицо. При пиксельном сравнении (рис. 7.10) видно, что Adam захватил меньше лишних деталей, чем его усовершенствованная версия AdamW.

Таким образом в данном конкретном случае оптимизатор Adam справился несколько лучше. Модель также зацепила не нужные фрагменты изображения, но сделала это меньше, чем при использовании AdamW. По результатам эксперимента у нас получилось успешно обучить модель глубокого обучения и сравнить две стратегии обратной связи. В дальнейшем обучении на других типах задач будем использовать стратегию оптимизатора Adam.

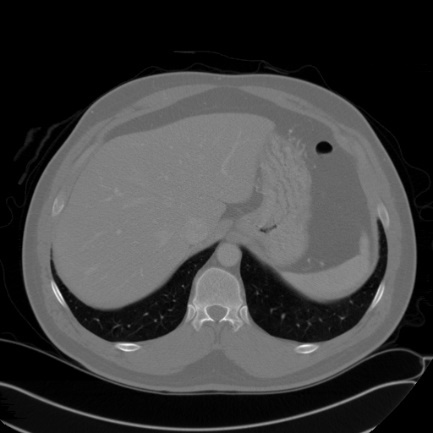
8. Обучение U-Net по распознаванию печени

Следующим шагом по обучению моделей на базе архитектуры U-Net становится переход к более сложным медицинским изображениям. Теперь, задача состоит в том, чтобы научить модель распознавать область печени на изображении и строить предсказание ее расположения в виде бинарной маски. Dataset, использующийся в этой главе (CT data batch) содержит несколько папок, соответствующим анонимным данным о пациентах – компьютерная томография печени. Каждая директория включает в себя множество изображений (от 50 до 95 штук) в виде файлов формата Pydicom. Сформируем тренировочные данные, аналогично главе 7. В таком случае, тренировочные данные представляют собой 30 чёрно-белых изображений. Примеры изображений см. на рис. 8.1, 8.2.

*Рис. 8.1 Тренировочные Image и Label из набора данных CT data batch*

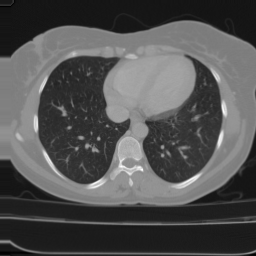
 

*Рис. 8.2 Изображения, отобранные для Test из набора данных CT data batch*

Повторяем алгоритм согласно главе 7 – расширяем наш текущий набор данных при помощи аугментации. Параметры аугментации ровно такие же.

* Rotation range равен
* Width shift range равен
* Height shift range равен
* Zoom range равен
* Horizontal flip используется
* Fill mode в виде «nearest»

Всего полученных таким образом изображений 1490 штук. Пример пары изображений приведен ниже на рис. 8.3. Так как изображения формируются случайным образом, то привести пример оригинала проблематично.

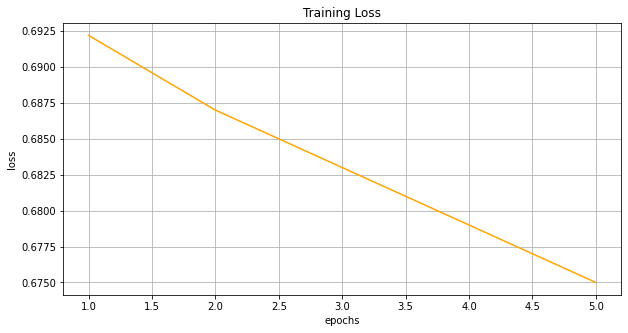
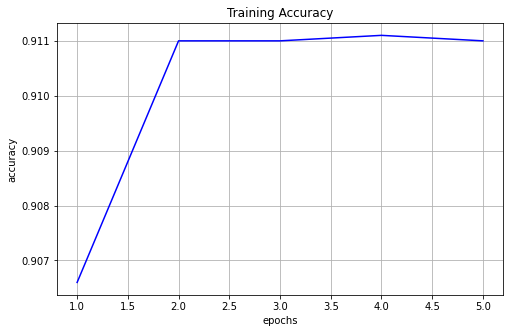
 

*Рис. 8.3 Аугментированные изображения для набора данных CT data batch*

Гиперпараметры настраиваем аналогично тем, что использовались в главе 7, так как опыта для их изменения пока что недостаточно. В будущем можно выбрать их более оптимально под данную конкретную задачу. То есть, параметр генератора batch\_size = 20, количество эпох равно 5, при этом параметр steps\_per\_epoch = 1000. Для сохранения весов (результата обучения) аналогично используем механизм callback в Python, который сработает после каждой эпохи в том случае, если метрики на этом шаге не ухудшились. Оптимизатор вычислений выберем Adam, так как в главе 7 он показал себя с лучшей стороны. Параметр learning rate = , так как при обучении модели для клеток мембраны это стало ключевым решением, приведшим к началу обучения модели.

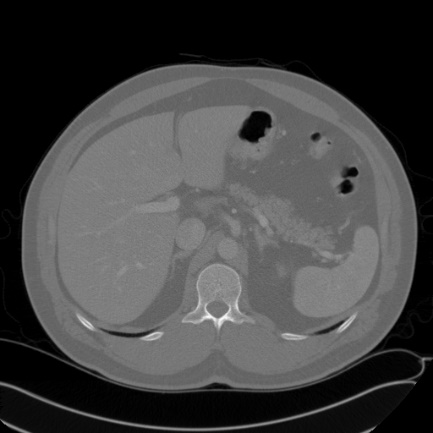
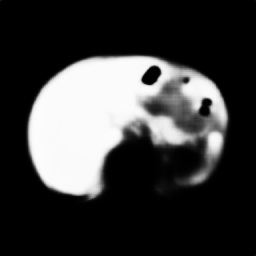
Для обучения использовался графический ускоритель, предоставляемый компанией Google в системе Google Colaboratory. На обучение модели ушло примерно полтора часа. Результаты были записаны в облачное хранилище Google Drive. Листинг выполнения программного кода со значениями метрик по прохождению каждой эпохи приведен в приложении листинг [3].

Однако, уже на этапе обучения стало понятно, что модель попросту не обучилась. Это можно легко увидеть на графиках (рис. 8.4). Значения потерь уменьшаются незначительно, а точность просто стагнирует.

** 

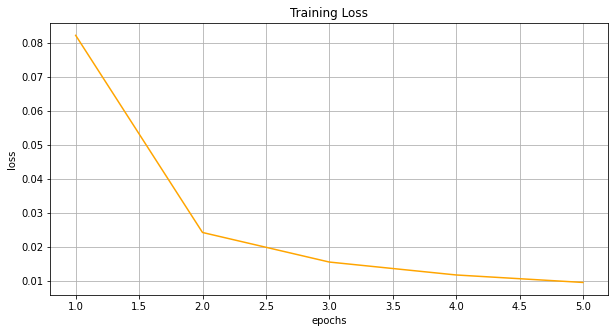
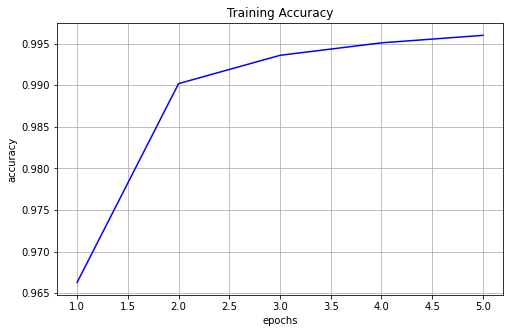
*Рис. 8.4 Потери и точность для первой попытки обучения*

Далее, попробуем заново сформировать тренировочные данные, составить новую аугментацию и заново обучить модель, так как не исключено, что оптимизатор Adam мог показать не очень удачный результат, попав в так называемый локальный минимум. Из гиперпараметров изменим только количество эпох, поставим 1. Смысл в том, что если модель начнет обучение, то это проявится сразу на первой же эпохе. И действительно, успех не заставил себя долго ждать. По итогу выполнения одной эпохи значения метрик составили loss: , accuracy: . Результат предсказания модели можно наблюдать на рис. 8.5.

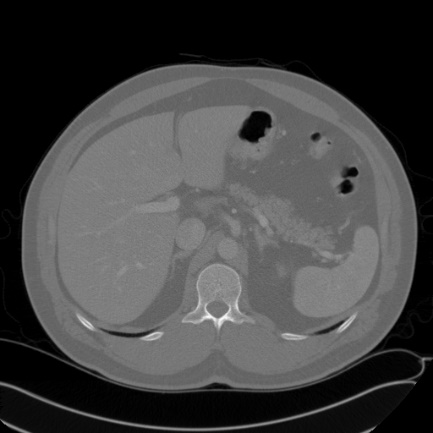
 

*Рис. 8.5 Предсказание модели после выполнения одной эпохи*

Теперь продолжим обучать данную модель. Метрики по результатам прохода еще 5 эпох на рис. 8.6. Далее, на рис. 8.7 приведены предсказания модели в виде бинарных масок.

*Рис. 8.6 Графики метрик для второй попытки обучения*

*Рис. 8.7 Бинарные маски после еще пяти эпох обучения*

Таким образом, в этой главе нам удалось успешно обучить сверточную нейронную сеть глубокого обучения на базе архитектуры U-Net. Судя по графику, обучение показало хорошую тенденцию. Хотя бинарные маски и цепляют лишнее, модель все же справляется с поставленной задачей.

# 9. Заключение

**Вывод:** в данной практической работе были изучены различные методы сегментирования изображения, как стандартных автоматических, так и с использованием технологий глубокого обучения. Был изучен язык Python, применены на практике библиотеки, расширяющие его возможности в вычислениях и построении нейронных сетей. Более того, были написаны реализации полносвязных, рекуррентных и сверточных нейронных сетей разной мерности для решения различного рода задач. Были рассмотрены специальные возможности фреймворков Keras и TensorFlow, повышающие качество и скорость экспериментов в целях построения наиболее удачных моделей нейронных сетей. Проанализированы наиболее распространенные решения для семантической и обычной сегментации изображений. В качестве такой сегментации медицинских изображений наиболее приемлемым вариантом может служить архитектура U-Net.

Во второй части практической работы была реализована и обучена модель на базе архитектуры U-Net. В качестве начального набора данных был выбран dataset, содержащий изображения клеток мембраны, сделанных под электронным микроскопом. Было изучен принцип работы и произведено сравнение оптимизаторов Adam и AdamW. По результатам эксперимента оптимизатор Adam показал несколько лучше результат, нежели его усовершенствованная версия, зацепив меньше лишних деталей. Далее, с помощью приобретенных навыков при обучении новой модели на задачи распознавания печени на изображениях компьютерной томографии анонимных пациентов удалось решить ряд проблем и в конечном счете получить удовлетворительный результат. Модель все-таки имеет некоторую неточность на тренировочных данных, но тем не менее, она способна верно определить область печени. Все манипуляции производились с применением формата DICOM, специального формата для файлов медицинских изображений, библиотекой Pydicom для интерпретации таких данных и облачного сервиса Google Colaboratory, бесплатно предоставляющего графические ускорители на некоторое ограниченное время.

# 10. Список литературы и источников информации

1. Шолле Ф. Глубокое обучение на Python. – СПб.: Питер, 2018 – 400 с.
2. Чичварин, Н. В. Текстурные методы (Сегментация) // Национальная библиотека им. Баумана. – Режим доступа: <https://ru.bmstu.wiki/Текстурные_методы_(Сегментация)>. – Загл. с экрана.
3. Малышев, В. Д. Разработка онлайн сервиса для сегментации медицинских изображений. – Минск: Ковчег, 2019 – 52 с.
4. Бойко, Д. О. Обзор методов сегментации медицинских изображений / А.Е. Филатова – Харьков: Национальный технический университет «Харьковский политехнический институт», 2015 – 70 c.
5. Gupta, D. A Beginner's guide to Deep Learning based Semantic Segmentation using Keras. – Режим доступа: <https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html>. – Загл. с экрана.
6. Odena, A., Dumoulin, V. and Olah, C. Deconvolution and Checkerboard Artifacts // Distill, 2016. – Режим доступа: <https://distill.pub/2016/deconv-checkerboard/>. – Загл. с экрана.
7. Ronneberger, O., Fischer, P. and Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation // arxiv.org. – Режим доступа: <https://arxiv.org/pdf/1505.04597.pdf>. – Загл. с экрана.
8. The most comprehensive semantic segmentation review in history (FCN, U-Net, SegNet, Deeplab, ASPP...) // ProgrammerSought. – Режим доступа: <https://programmersought.com/article/45131571697/>. – Загл. с экрана.
9. Тропченко, А. Ю. Методы вторичной обработки и распознавания изображений: Учебное пособие / A. A. Тропченко. – СПб: Университет ИТМО, 2015 – 215 с.

1. [Seo](https://medium.com/@SeoJaeDuk?source=post_page-----d07231eb29bf-----------------------------------), J. D. Medical Image Segmentation [Part 1] — U-Net: Convolutional Networks with Interactive Code, 2018. – Режим доступа: <https://medium.com/@SeoJaeDuk/medical-image-segmentation-part-1-unet-convolutional-networks-with-interactive-code-d07231eb29bf>. – Загл. с экрана.
2. Draelos, R. Segmentation: U-Net, Mask R-CNN, and Medical Applications. // Glass Box, 2020. – Режим доступа: <https://glassboxmedicine.com/2020/01/21/segmentation-u-net-mask-r-cnn-and-medical-applications/>. – Загл. с экрана.

1. [Banerjee](https://www.how2shout.com/author/sagnik17835377" \o "Posts by Sagnik Banerjee), S. Google Colaboratory, Python in Cloud: Quick Review, 2020. – Режим доступа: <https://www.how2shout.com/review/google-colaboratory-python-in-cloud-quick-review.html>. – Загл. с экрана.
2. Овсянникова, В. Что такое Google Colab и кому он нужен?, 2021. – Режим доступа: <https://blog.skillfactory.ru/chto-takoe-google-colaboratory-i-komu-on-nuzhen/>. – Загл. с экрана.
3. Fabio, M. G. Why AdamW matters, 2018. – Режим доступа: <https://towardsdatascience.com/why-adamw-matters-736223f31b5d>. – Загл. с экрана.
4. Loshchilov, I. and Hutter, F. Fixing Weight Decay Regularization in Adam // arxiv.org. – Режим доступа: <https://arxiv.org/pdf/1711.05101v2.pdf>. – Загл. с экрана.
5. Bharath, K. U-Net Architecture for Image Segmentation. – Режим доступа: <https://blog.paperspace.com/unet-architecture-image-segmentation/>. – Загл. с экрана.
6. Pydicom User Guide // Pydicom documentation. – Режим доступа: <https://pydicom.github.io/pydicom/stable/old/pydicom_user_guide.html>. – Загл. с экрана.
7. Virdee, A. Understanding DICOMs, 2020. – Режим доступа: <https://towardsdatascience.com/understanding-dicoms-835cd2e57d0b>. – Загл. с экрана.

# 11. Приложение

11.1 Листинг выполнения программного кода

1. Листинг выполнения обучения на начальном dataset с использованием Adam

Epoch 1/5

1000/1000 [==============================] - ETA: 0s - loss: 0.3116 - accuracy: 0.8699

Epoch 1: loss improved from inf to 0.31163, saving model to /content/drive/MyDrive/weights/weightsAdam\_cells.hdf5

1000/1000 [==============================] - 1649s 2s/step - loss: 0.3116 - accuracy: 0.8699

Epoch 2/5

1000/1000 [==============================] - ETA: 0s - loss: 0.2639 - accuracy: 0.9049

Epoch 2: loss improved from 0.31163 to 0.26391, saving model to /content/drive/MyDrive/weights/weightsAdam\_cells.hdf5

1000/1000 [==============================] - 1223s 1s/step - loss: 0.2639 - accuracy: 0.9049

Epoch 3/5

1000/1000 [==============================] - ETA: 0s - loss: 0.2101 - accuracy: 0.9149

Epoch 3: loss improved from 0.26391 to 0.21006, saving model to /content/drive/MyDrive/weights/weightsAdam\_cells.hdf5

1000/1000 [==============================] - 1218s 1s/step - loss: 0.2101 - accuracy: 0.9149

Epoch 4/5

1000/1000 [==============================] - ETA: 0s - loss: 0.1760 - accuracy: 0.9229

Epoch 4: loss improved from 0.21006 to 0.17597, saving model to /content/drive/MyDrive/weights/weightsAdam\_cells.hdf5

1000/1000 [==============================] - 1219s 1s/step - loss: 0.1760 - accuracy: 0.9229

Epoch 5/5

1000/1000 [==============================] - ETA: 0s - loss: 0.1648 - accuracy: 0.9278

Epoch 5: loss improved from 0.17597 to 0.16477, saving model to /content/drive/MyDrive/weights/weightsAdam\_cells.hdf5

1000/1000 [==============================] - 1217s 1s/step - loss: 0.1648 - accuracy: 0.9278

1. Листинг выполнения обучения на начальном dataset с использованием AdamW

Epoch 1/5

1000/1000 [==============================] - ETA: 0s - loss: 0.3065 - accuracy: 0.8729

Epoch 1: loss improved from inf to 0.30654, saving model to /content/drive/MyDrive/weights/weightsAdamW\_cells.hdf5

1000/1000 [==============================] - 1268s 1s/step - loss: 0.3065 - accuracy: 0.8729

Epoch 2/5

1000/1000 [==============================] - ETA: 0s - loss: 0.2619 - accuracy: 0.9052

Epoch 2: loss improved from 0.30654 to 0.26190, saving model to /content/drive/MyDrive/weights/weightsAdamW\_cells.hdf5

1000/1000 [==============================] - 1229s 1s/step - loss: 0.2619 - accuracy: 0.9052

Epoch 3/5

1000/1000 [==============================] - ETA: 0s - loss: 0.2483 - accuracy: 0.9142

Epoch 3: loss improved from 0.26190 to 0.24826, saving model to /content/drive/MyDrive/weights/weightsAdamW\_cells.hdf5

1000/1000 [==============================] - 1230s 1s/step - loss: 0.2483 - accuracy: 0.9142

Epoch 4/5

1000/1000 [==============================] - ETA: 0s - loss: 0.2404 - accuracy: 0.9192

Epoch 4: loss improved from 0.24826 to 0.24043, saving model to /content/drive/MyDrive/weights/weightsAdamW\_cells.hdf5

1000/1000 [==============================] - 1231s 1s/step - loss: 0.2404 - accuracy: 0.9192

Epoch 5/5

1000/1000 [==============================] - ETA: 0s - loss: 0.2368 - accuracy: 0.9216

Epoch 1: loss improved from 0.24043 to 0.2368, saving model to /content/drive/MyDrive/weights/weightsAdamW\_cells.hdf5

1000/1000 [==============================] - 1229s 1s/step - loss: 0.2368 - accuracy: 0.9216

1. Листинг выполнения обучения на dataset CT data batch попытка 1

Epoch 1/5

1000/1000 [==============================] - ETA: 0s - loss: 0.6922 - accuracy: 0.9066

Epoch 1: loss improved from inf to 0.69219, saving model to /content/drive/MyDrive/weights/weightsFirst\_liver.hdf5

1000/1000 [==============================] - 1158s 1s/step - loss: 0.6922 - accuracy: 0.9066

Epoch 2/5

1000/1000 [==============================] - ETA: 0s - loss: 0.6870 - accuracy: 0.9110

Epoch 2: loss improved from 0.69219 to 0.68705, saving model to /content/drive/MyDrive/weights/weightsFirst\_liver.hdf5

1000/1000 [==============================] - 1138s 1s/step - loss: 0.6870 - accuracy: 0.9110

Epoch 3/5

1000/1000 [==============================] - ETA: 0s - loss: 0.6830 - accuracy: 0.9110

Epoch 3: loss improved from 0.68705 to 0.68301, saving model to /content/drive/MyDrive/weights/weightsFirst\_liver.hdf5

1000/1000 [==============================] - 1138s 1s/step - loss: 0.6830 - accuracy: 0.9110

Epoch 4/5

1000/1000 [==============================] - ETA: 0s - loss: 0.6790 - accuracy: 0.9111

Epoch 4: loss improved from 0.68301 to 0.67899, saving model to /content/drive/MyDrive/weights/weightsFirst\_liver.hdf5

1000/1000 [==============================] - 1136s 1s/step - loss: 0.6790 - accuracy: 0.9111

Epoch 5/5

1000/1000 [==============================] - ETA: 0s - loss: 0.6750 - accuracy: 0.9110

Epoch 5: loss improved from 0.67899 to 0.67501, saving model to /content/drive/MyDrive/weights/weightsFirst\_liver.hdf5

1000/1000 [==============================] - 1135s 1s/step - loss: 0.6750 - accuracy: 0.9110

1. Листинг выполнения обучения на dataset CT data batch попытка 2

Epoch 1/5

1000/1000 [==============================] - ETA: 0s - loss: 0.0823 - accuracy: 0.9663

Epoch 1: loss improved from inf to 0.08230, saving model to /content/drive/MyDrive/weights/weightsThird\_liver.hdf5

1000/1000 [==============================] - 1975s 2s/step - loss: 0.0823 - accuracy: 0.9663

Epoch 2/5

1000/1000 [==============================] - ETA: 0s - loss: 0.0243 - accuracy: 0.9902

Epoch 2: loss improved from 0.08230 to 0.02429, saving model to /content/drive/MyDrive/weights/weightsThird\_liver.hdf5

1000/1000 [==============================] - 1156s 1s/step - loss: 0.0243 - accuracy: 0.9902

Epoch 3/5

1000/1000 [==============================] - ETA: 0s - loss: 0.0156 - accuracy: 0.9936

Epoch 3: loss improved from 0.02429 to 0.01562, saving model to /content/drive/MyDrive/weights/weightsThird\_liver.hdf5

1000/1000 [==============================] - 1151s 1s/step - loss: 0.0156 - accuracy: 0.9936

Epoch 4/5

1000/1000 [==============================] - ETA: 0s - loss: 0.0118 - accuracy: 0.9951

Epoch 4: loss improved from 0.01562 to 0.01183, saving model to /content/drive/MyDrive/weights/weightsThird\_liver.hdf5

1000/1000 [==============================] - 1157s 1s/step - loss: 0.0118 - accuracy: 0.9951

Epoch 5/5

1000/1000 [==============================] - ETA: 0s - loss: 0.0096 - accuracy: 0.9960

Epoch 5: loss improved from 0.01183 to 0.00962, saving model to /content/drive/MyDrive/weights/weightsThird\_liver.hdf5

1000/1000 [==============================] - 1167s 1s/step - loss: 0.0096 - accuracy: 0.9960

* 1. Код программы

1. Код программы, расположенный на GitHub

<https://github.com/PureEvil-UselessMan/SegmentMedImages>

1. Код программы, расположенный в Google Colaboratory

<https://colab.research.google.com/drive/1Epjn2pBWxn74ErBhClyeL9-6DRYiuchj?usp=sharing>