**МИНОБРНАУКИ РОССИИ**

# ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

# УЧРЕЖДЕНИЕ

**ВЫСШЕГО ОБРАЗОВАНИЯ**

# «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет компьютерных наук

Кафедра цифровых технологий

Использование вейвлет-преобразования и сверточных нейронных

сетей для классификации записей кашля

ВКР Бакалаврская работа

02.03.01 Математика и компьютерные науки

Распределенные системы и искусственный интеллект

Допущено к защите в ГЭК 2021

Зав. кафедрой С. Д. Кургалин, д. ф.-м. н., профессор

Обучающийся А.В. Стрельников, 4 курс, д/о

Руководитель А.В. Максимов, старший преподаватель

Воронеж 2021

**МИНОБРНАУКИ РОССИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Факультет компьютерных наук

Кафедра цифровых технологий

**УТВЕРЖДАЮ**

заведующий кафедрой

*подпись, расшифровка подписи*

. .2021

**ЗАДАНИЕ**

**НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ ОБУЧАЮЩЕГОСЯ** Стрельников Антон Викторович

*фамилия, имя, отчество*

1. Тема работы: Использование вейвлет-преобразования и сверточных нейронных сетей для классификации записей кашля, утверждена решением ученого совета факультета компьютерных наук от …………2021

2. Направление подготовки: 02.03.01 Математика и компьютерные науки

3. Срок сдачи студентом законченной работы …….2021

4. Календарный план:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Структура ВКР | Сроки выполнения | Примечания |
| 1 | Введение | 07.03.2021 - 15.03.2021 |  |
| 2 | Глава 1. ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ | 05.04.2021 - 29.04.2021 |  |
| 3 | Глава 2. СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ. | 05.05.2021 - 12.05.2021 |  |
| 4 | Глава 3. ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ | 15.05.2021 - 24.05.2021 |  |
| 5 | Заключение | 25.05.2021 - 30.05.2021 |  |

|  |  |  |
| --- | --- | --- |
| Обучающийся: |  | Стрельников А.В. |
|  | *подпись* | *расшифровка подписи* |
| Руководитель: |  | Максимов А.В. |
|  | *подпись* | *расшифровка подписи* |

**Реферат**

Бакалаврская работа 62 с., 18 рис.

ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ, НЕЙРОННЫЕ СЕТИ, СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, КЛАССИФИКАЦИЯ ИЗОБРАЖЕНИЙ, МАСШТАБОГРАММЫ, СКАЛОГРАММЫ.

Объектом исследования являются вейвлет-преобразование, сверточные нейронные сети, эффективность работы сверточных нейросетей для задач классификации

Цель работы – создать приложение для анализа респираторной активности, которое позволяет конвертировать имеющиеся записи кашлей в .wav формат, производить вейвлет - преобразование записей, отделять записи кашля от речи и посторонних шумов, классифицировать записи по заболеваниям (с разделением на обычные, продуктивные, свистящие, а также записи Covid больных), делать предсказания по тестовым записям на принадлежность к какому-либо классу

В процессе выполнения работы проводились исследования вейвлет-преобразований, сверточных нейронных сетей, их взаимодействия и точности предсказаний нейронных сетей.

В результате исследования было выяснено, что сверточные нейросети справляются с задачами классификации изображений с большой точностью, что позволяет им делать достоверные предсказания по записям кашля.

Степень внедрения: предлагается использовать данный алгоритм для эффективного тестирования Covid больных

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ 5**](#_Toc74560297)

[**ГЛАВА 1. ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ 7**](#_Toc74560298)

[**1.1 Частотно-временная локализация 7**](#_Toc74560299)

[**1.2 Сходства и отличия вейвлет преобразования в сравнении с оконным преобразованием Фурье 8**](#_Toc74560300)

[**1.3 Типы вейвлет-преобразований 13**](#_Toc74560301)

[**1.3.1 Непрерывные вейвлет-преобразования 13**](#_Toc74560302)

[**1.3.2 Дискретное избыточное вейвлет-преобразование(фрейм) 13**](#_Toc74560303)

[**ГЛАВА 2. СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ. 16**](#_Toc74560304)

[**2.1 Понятие и признаки 16**](#_Toc74560305)

[**2.2 Дискретная свертка 17**](#_Toc74560306)

[**2.2.1 Определение размера выхода свертки 21**](#_Toc74560307)

[**2.2.2 Выполнение дискретной свертки в двух измерениях 22**](#_Toc74560308)

[**2.3 Слои подвыборки 25**](#_Toc74560309)

[**2.4 Работа с множественными входными или цветовыми каналами 26**](#_Toc74560310)

[**2.5 Регуляризация нейронной сети с помощью отключения 29**](#_Toc74560311)

[**2.6 Функции потерь для классификации 31**](#_Toc74560312)

[**ГЛАВА 3. ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 35**](#_Toc74560313)

[**3.1 Использованные средства 35**](#_Toc74560314)

[**3.2 Структура программного обеспечения 35**](#_Toc74560315)

[**3.3 Функционал программного обеспечения 36**](#_Toc74560316)

[**3.3 Интерфейс программного обеспечения 55**](#_Toc74560317)

[**3.4 Примеры работы программы и обсуждение результатов 56**](#_Toc74560318)

[**ЗАКЛЮЧЕНИЕ 60**](#_Toc74560319)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 61**](#_Toc74560320)

# 

# 

# ВВЕДЕНИЕ

Кашель является самым распространенным симптомом заболеваний органов дыхания и встречается у 20% лиц в популяции. На данный момент времени диагностика кашля с помощью визуальных аналоговых шкал и специальных опросников отличается выраженной субъективностью. Задача данной работы создать систему для анализа и диагностики записей кашля. Актуальность диагностики кашля многократно возрастает в связи с пандемией Covid-19, одним из симптомов которой является кашель.

Для реализации данной системы воспользуемся вейвлет-преобразованием и сверточными нейронными сетями. Вейвлет - преобразование поможет с построением масштабограммы или скейлограммы. Сверточные нейронные сети будут использоваться для классификации изображений скейлограммы, так как они показывают высокую эффективность в данной области.

К целям данной работы относится реализация приложения анализа респираторной активности, которое позволяет конвертировать имеющиеся записи кашлей в .wav формат, производить вейвлет преобразование записей, отделять записи кашля от речи и посторонних шумов, классифицировать записи по заболеваниям (с разделением на обычные, продуктивные и свистящие, а также записи Covid больных), делать предсказания по тестовым записям на принадлежность к какому-либо классу.

Для выполнения поставленной цели необходимо решить следующие задачи:

1. Подготовить датасет аудиозаписей кашлей и посторонних звуков

2. Выполнить вейвлет преобразование аудиозаписей и построить скейлограмму

3. Подготовить датасет из скейлограмм имеющихся записей

4. Создать нейронную сеть, отделяющую кашель от речи и посторонних звуков

5. Создать нейронную сеть, способную классифицировать записи по скейлограмме и позволяющую делать предсказания по тестовым записям.

Схожая система анализа была разработана «Сбербанком». Она способна выявлять заражение COVID-19 с помощью анализа записей дыхания, кашля и голоса. Также записываются и анализируются результаты опроса больного о симптомах заболевания. Обученная нейросеть анализирует спектрограммы звуков, которые показывают их мощность в зависимости от частотных диапазонов. Данная система может диагностировать ковид за 1 минуту с большой точностью. В данной работе будет использоваться схожая система программного анализа, включающее вейвлет-преобразование и машинное обучение.

# 

# 

# 

# ГЛАВА 1. ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ

1.1 Частотно-временная локализация

Предположим, что t - непрерывная переменная и у нас имеется заданный заранее сигнал f(t). Давайте получим его локальную во времени интересно знать его частотную характеристику. Данную операцию можно сравнить с музыкальной тематикой, когда музыкант получает информацию о том какую тональность и ноту необходимо взять. То есть музыкант получает частотную информацию. Сначала рассмотрим самое обычное преобразование Фурье:

Преобразование Фурье дает представление о частотной характеристике заданного сигнала f. Однако извлечение информации о пиках с высокой

частотой, связанной с временной локализации, представляется довольно проблематичным. Мы можем получить временную локализация, например, с помощью окон, когда берется некоторый кусок f, который хорошо локализован, и потом выполняется его преобразование Фурье:

(1.1)

Использование оконного преобразования Фурье является довольно распространенной техникой для пояснения частотно-временной локализации некоторого сигнала. Эксперты, которые связаны и работают с анализом сигнала, встречаются с ним чаще в дискретном варианте, когда t и ω принимают значения t = nt0, ω = mω0, где m, n пробегают ℤ, а ω0, t0 > 0 —

фиксированные. Тогда (1) преобразуется в

(1.2)

1.2 Сходства и отличия вейвлет преобразования в сравнении с оконным преобразованием Фурье

При использовании вейвлет-преобразование получается похожее частотно-временное описание. Отличие составляют некоторые значительные особенности. Формулы, аналогичные (1.1), (1.2), выглядят следующим образом:

(2.1)

(2.2)

В обоих случаях мы предполагаем, что удовлетворяет условию

(2.3)

Из формулы (2.1) получаем формулу (2.2), основываясь на том, что а, b принимают только определенные дискретные значения: а = а0m, b = nb0a0m, m, n пробегают значения из Z, величины a0 > 1, b0 > 0 — фиксированные. Также сразу заметно одно очевидное сходство преобразования Фурье и вейвлет преобразования. Описанные выше формулы (1.1) и (2.1) являются обычным скалярным произведением f и семейства функций, у которых имеются два индекса,

для (1.1)

для (2.1).

Функцию имеет особое название - «материнский вейвлет». В свою очередь функции называются «вейвлетами». (Заметим, что и g неявно подразумеваются вещественными, хотя это не оказывает никакого значительного влияния). Если же они не вещественные, то необходимо воспользоваться комплексным сопряжением в (1.1), (2.1). Обычно в качестве используют (t) = (1- t2)exp(- t2/2).

Данная конструкция имеет название мексиканская шляпа из-за того, что если построить график, то его форма похожа на мексиканскую шляпу. Данная функция локализована во времени и в пространстве хорошо и удовлетворяет (2.3).

меняет свою частоту, когда параметр а изменяет свои значения. Происходит это следующим образом: малые параметры |а| соответствуют высоким частотам или очень мелкому масштабу большие значения масштабирующего параметра |a| соответствуют малым частотам или большому масштабу . Cмещение центра временной локализации возможно с помощью изменения параметра b: каждая s) локализована около s = b. Из этого следует что,

(2.1)

дают частотно-временное описание f. Также различие оконного преобразованием Фурье и вейвлет-преобразования заключается в форме анализирующих функций и , которые отображены на рисунке ниже.

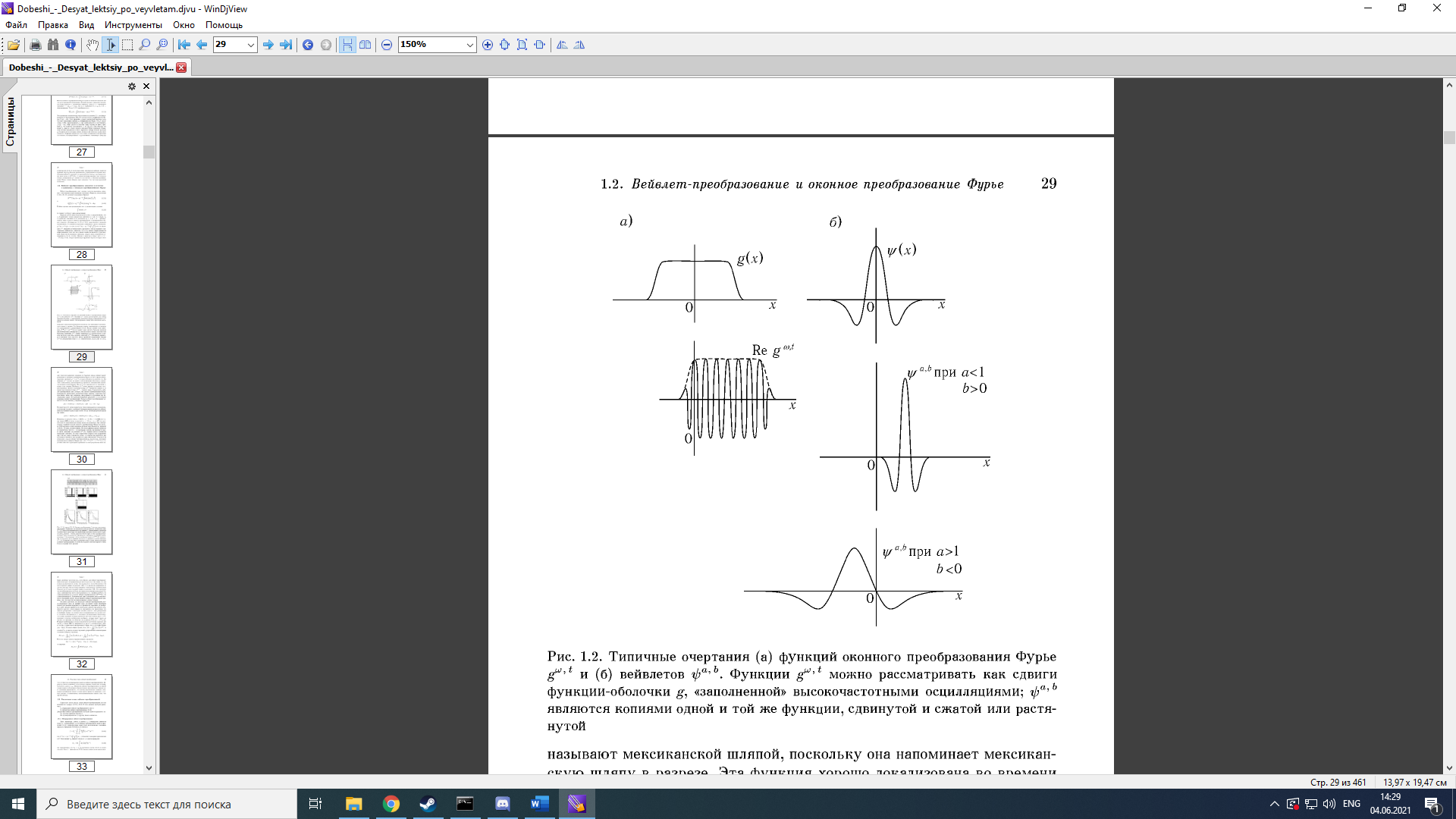


Рисунок 1.1 - Обычные очертания (а) функций оконного преобразования Фурье

и (б) вейвлетов . Функции можно рассматривать как сдвиги

функции-оболочки g, «заполненной» высокочастотными осцилляциями;

представляют собой копии одной и той же функции, сдвинутой и сжатой или растянутой

Функции включают в себя одинаковые для всех функции-оболочки g. Они наполнены высокочастотными осцилляциями и сдвинуты к подходящему расположению по времени. Все , обладают одинаковой шириной вне зависимости от значения ,. Наоборот, имеет ширину во времени, соответствующую частоте: высокочастотные являются узкими, в то время как низкочастотные — намного шире. В результате, вейв-

лет-преобразование дает лучшую, чем оконное преобразование Фурье,

возможность рассмотреть высокочастотные явления с коротким сроком жизни такие, как, например, сингулярности в функциях или интегральных ядрах. Это проиллюстрировано рисунком 1.2, на котором

показаны оконное преобразование Фурье и вейвлет-преобразование одного и того же сигнала f, заданного формулой

На практике этот сигнал задается не таким непрерывным выражением,

а отсчетами (samples), и добавка -функции аппроксимируется добавлением постоянной только в один отсчет. Тогда в этой дискретной версии

мы имеем

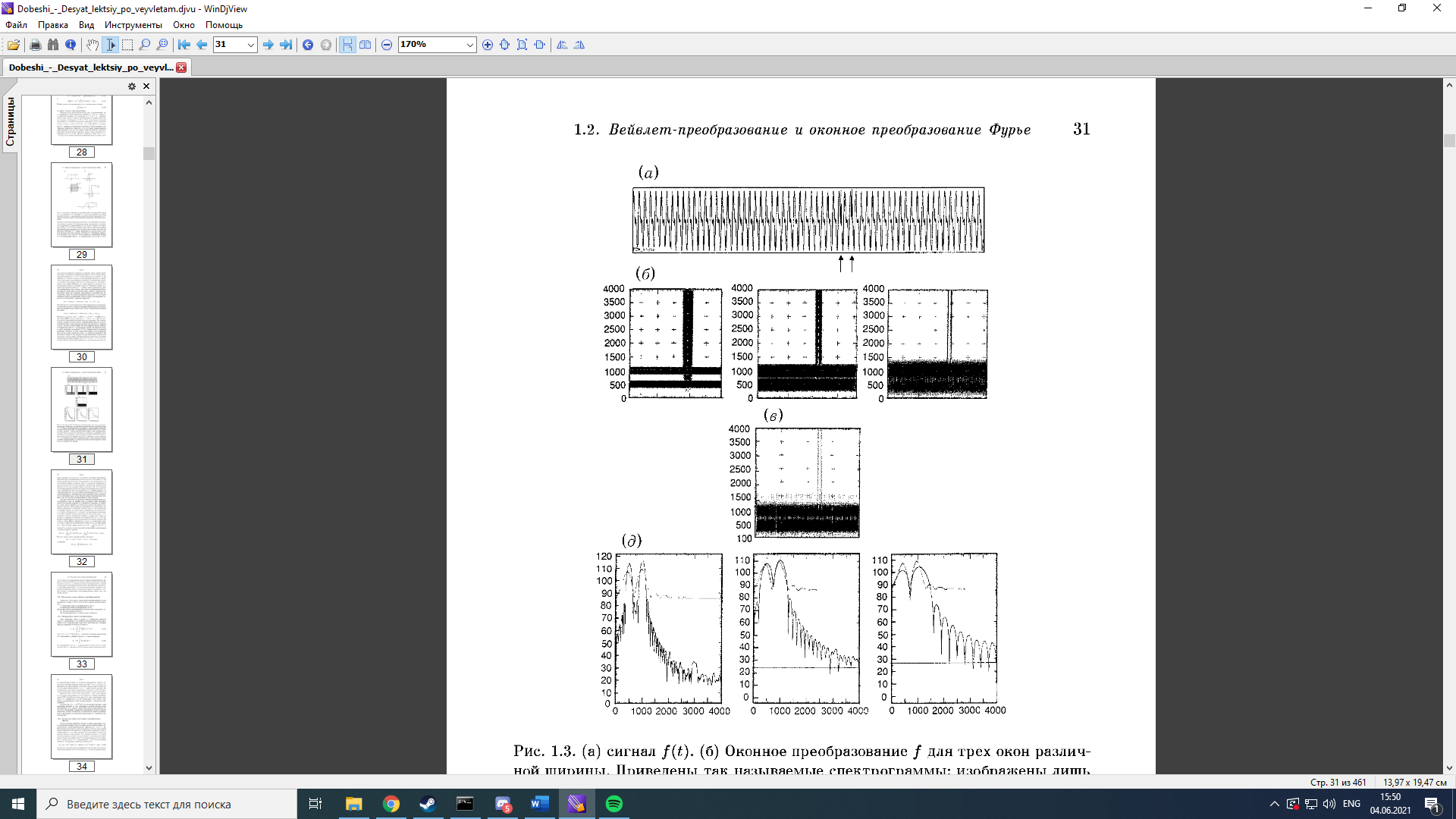
Рисунок 1.2в показывает модули вейвлет-преобразований f, полученных с помощью вейвлета Морле ,= 4. 

Рисунок. 1.2 - (а) сигнал f(t), (б) Оконное преобразование f для трех окон различной ширины. Приведены так называемые спектрограммы: изображены лишь с использованием различных оттенков серого цвета (при этом наибольшие значения соответствуют черному цвету, нулевые — белому, промежуточные серые оттенки пропорциональны log ) на плоскости t (абсцисса) и (ордината), (в) Вейвлет-преобразование f. Для сравнения с (б) мы изобразили также и использовав тот же метод, и ось линейной частоты (т. е. ординату, соответствующую a-1), (г) Сравнение частотного разрешения между тремя спектрограммами

и вейвлет-преобразованием.

Эксперименты и численное моделирование показывают, что волны давления, которые являются чистыми тонами,

вызывают ответное возбуждение мембраны, которая имеет такую же

частоту во времени, но оболочку по у равную .

В первом приближении, которое оказывается достаточно хорошим для

частот свыше 500 Гц, зависимость от соответствует сдвигу на log: существует одна функция такая, что очень близко

. В самом общем случае, если , то

отклик F(t, у) дается соответствующей суперпозицией «элементарных

откликов» (response functions)

Если мы введем замену параметризации, определив

, G(a, t) = F(t, log a),

то получим

что с точностью до нормировки является вейвлет-преобразованием. Параметр сжатия возникает естественным образом вследствие логарифмического сдвига в . Появление вейвлет-преобразования на первой стадии нашего собственного биологического акустического анализа дает основание предполагать, что методы акустического анализа, основанные на вейвлетах, имеют лучшие шансы привести, например, к схемам сжатия, с искажениями, невоспринимаемыми нашим ухом, чем другие методы.

1.3 Типы вейвлет-преобразований

1.3.1 Непрерывные вейвлет-преобразования

Здесь параметры сдвига и сжатия а, b непрерывно меняются

вдоль R, с ограничением а 0. Вейвлет-преобразование задается формулой

(2.1),

любая функция может быть восстановлена с помощью формулы обращения

(3.1)

где , а < , > обозначает скалярное произведение

в L2. Постоянная С зависит только от и дается формулой

(3.2)

мы предполагаем, что < (в противном случае (3.1) не имеет

смысла). Если — функция из L1(R) (как раз такие случаи представляют практический интерес), то является непрерывным, тогда мо-

жет иметь конечное значение только если , т.е. .

Формула (3.1) может быть рассмотрена с двух точек зрения:

A) как способ восстановления f, если известно ее вейвлет-преобразование Tвейв f или Б) как способ записи f в виде суперпозиции вейвлетов , коэффициенты в этой суперпозиции точно заданы через

вейвлет-преобразование f.

1.3.2 Дискретное избыточное вейвлет-преобразование(фрейм)

В этом случае оба параметра, сжатия а и сдвига, принимают только дискретные значения. Для а мы берем целые (отрицательные и положительные) степени фиксированного параметра а0 > 1, т. е. а = a0m.

Как показано на рисунке 1.2, разные значения m соответствуют разной

ширине вейвлетов. Следовательно, дискретизация параметра сдвига b

должна зависеть от m: узкие (высокие частоты) вейвлеты сдвигаются

малыми шагами, чтобы покрыть весь временной спектр, в то время

как более широкие (низкие частоты) вейвлеты сдвигаются большими

шагами. Поскольку ширина пропорциональна , мы выбира-

ем b = nb0a0m, где b0 > 0 — фиксированное, n. Соответствующие

вейвлеты с дискретными индексами выглядят как

На рисунке 1.4а схематически изображена сетка центров частотно-временной локализации, соответствующая .

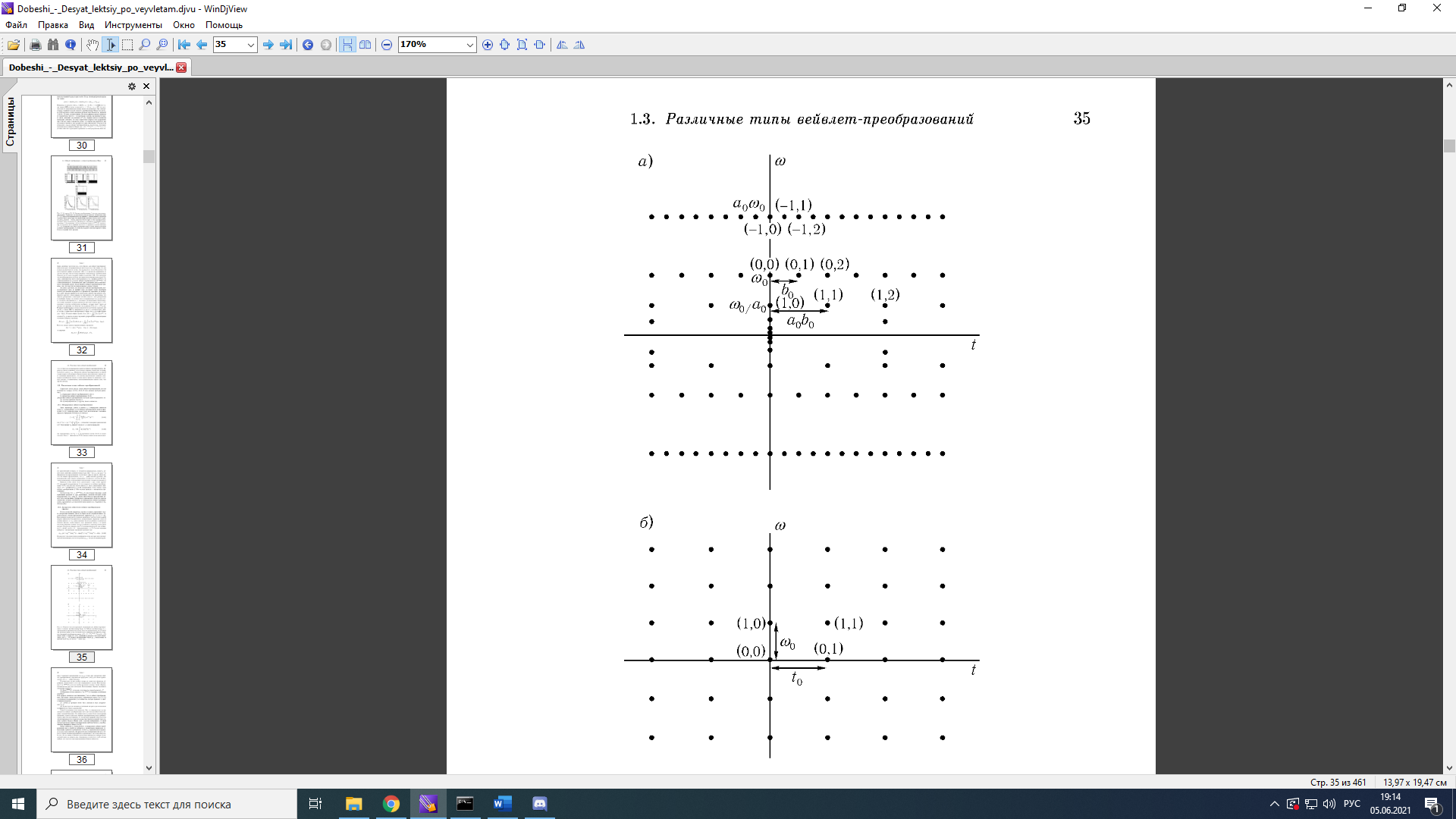


Рисунок. 1.4 - Решетки частотно-временной локализации для вейвлет-преобразования и оконного преобразования Фурье, (а) Вейвлет-преобразование: локализованы по времени около a0mnbo- Здесь мы предположили, что имеет два частотных пика в (это имеет место, например, для вейвлета, заданного функцией «мексиканская шляпа»,

Тогда имеют пики в точках , являющихся центрами частотной локализации для . (б) Оконное преобразование Фурье: gm,n локализованы по

времени около nto, по частоте — около

Тогда для заданной функции f скалярное произведение точно дает дискретное вейвлет-преобразование, определенное формулой (предполагается, что — вещественная).

(2.2)

Дискретное вейвлет-преобразование в большом количестве случаев дает хорошее и весьма избыточное описание исходной функции, также как и в непрерывном случае. Данному факту можно найти весьма эффективное применение. Допустим, необходимо выполнить вейвлет-преобразование некоторого имеющего сигнала. В данном случае, используя избыточность, можно выполнить это преобразование приблизительно, при этом восстановленный f будет обладать довольно высокой точностью. Помимо этого, избыточность можно ликвидировать. Для ликвидации необходимо удалить его несущественные значения.

, определенная по формуле

должна быть конечной. Данное условие сильно ограничивает возможный диапазон выбора вейвлета , который будет использоваться в непрерывном вейвлет-преобразовании. Для упрощения реализации зачастую выбирается такой, который хорошо сгруппирован во временной и частотной областях.

# ГЛАВА 2. СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ.

2.1 Понятие и признаки

Для эффективной реализации алгоритмов необходимо успешное извлечение заметных (значимых) признаков. В случае алгоритмов МО и традиционных модели МО – это ключевой аспект. Ведь они опираются на входные признаки, которые поступают от ученого или эксперта данной предметной области. Также большую роль играют вычислительные методики выделения признаков. Сверточные нейронные сети (CNN) и другие определенные нейросети могут в автоматическом режиме выделять в сырых данных признаки, имеющие наибольшую важность в данном случае. Из-за такого строения слои CNN выступают в роли маркеров выделения признаков: первые слои, которые находятся сразу же после входного инициализируют низкоуровневые признаки из входных данных, последующие полносвязные слои (также имеются в многослойном персептроне) используют данные признаки чтобы организовать возможность прогнозирования непрерывного целевого значения или метки класса.

Глубокие сверточные нейронные сети и некоторые другие типы многослойных нейронных сетей способны создавать определенную иерархию признаков. При этом происходит объединение низкоуровневых признаков в послойной манере. Данное объединение происходит для формирования высокоуровневых признаков. Допустим мы имеем нейронную сеть, входом которой являются изображения. Тогда начальные слои извлекали бы низкоуровневые признаки наподобие граней и пятен, которые объединялись бы вместе с целью образования высокоуровневых признаков. Возможно создание и более сложных форм с использованием высокоуровневых признаков (например, общие контуры объектов вроде зданий, дорог, автомобилей). Сети CNN показывают хорошие результаты при решении задач, которые относятся к изображениям.

Это, в значительной степени, описывается следующими заявлениями.

1. Разреженная связность. С небольшим участком пикселей связывается одиночный элемент в карте признаков. Отличие от персептронов заключается в том, что при использовании персептронов происходит связывания с целым входным изображением.
2. Совместное использование параметров. Одни и те же веса используются по всей площади входного изображения.

Из данных заявлений можно сделать такой вывод: использование сверточного слоя, вместо традиционного полносвязного многослойного персептрона приводит к значительному уменьшению количества параметров в сети. При этом произойдет улучшение способности захвата заметных признаков. При рассмотре со стороны данных изображения возможно сделать предположение, что близлежащие пиксели имеют большее отношение друг к другу, чем пиксели, находящиеся далеко друг от друга.

Сверточные нейронные сети обычно включаются в себя несколько слоев подвыборки и несколько сверточных слоев. После них в конце находится один или большее количество полносвязных слоев.

Полносвязные слои являются самым обычным многослойным персептроном, в котором каждый входной элемент i связан с каждым выходным элементом j с весом wij.

При этом сверточные и полносвязные слои имеют свои веса и смещения, которые оптимизируются во время обучения. Объединяющие слои или слои подвыборки описываются отсутствием каких-либо параметров обучения.

2.2 Дискретная свертка

Основополагающей операцией в сверточных нейронных сетях выступает дискретная свертка или просто свертка. Из-за ее важности необходимо понимать устройство ее работы.

Дискретная свертка для двух одномерных векторов (х - наш вход, который также является сигналом) и w обозначается как у = х \* w, а w – ядро или фильтр. Запишем математическую формулу дискретной свертка:

Квадратные скобки [ ] используются для обозначения индексации элементов в векторе. Индекс i проходит по всем элементам выходного вектора у.

Факт суммирования по индексам от - до + выглядит странным в так как мы всегда имеем дело с конечными векторами признаков в приложениях МО. Допустим в х имеется 10 признаков с индексами 0, 1, 2, .. , 8, 9, тогда индексы -: -1 и 10 : + превышают максимальные и минимальные границы вектора х. Из этого следует, что для корректного вычисления суммы в предыдущем уравнении было предположено следующее: векторы х и w заполнены нулями. Таким образом выходной вектор у также будет бесконечного размера и состоять из большого количества нулей. На самом деле это оказывается очень неудобно и в результате вектор х заполняется лишь конечным числом нулей. Данный процесс имеет название дополнением нулями и просто дополнением. Количество нулей, дополняемых с каждой стороны, обозначается как р.

Допустим, что изначально имеем вектор фильтра w и входной вектор х. Они включают в себя n и m элементов соответственно, где m n. Получаем что вектор хР, дополненный таким образом имеет размер n + 2р.

Уравнение для вычисления дискретной свертки после данного изменения приобретет следующий вид:

Теперь перейдем к индексации вектора х посредством i + m - k. Здесь

важно отметить, что при суммировании векторы х и w индексируются в разных направлениях. Для вычисления суммы, которая имеет определенный индекс, идущим в обратном направлении, необходимо вычислить суммы с обоими индексами в прямом направлении после зеркального обращения одного из векторов, х или w, сразу после того, как они будут дополнены. Далее просто происходит вычисление их скалярного произведения. Допустим, что у нас имеется зеркально повернутый фильтр w. Он нужен чтобы получить повернутый фильтр wr. Далее мы вычисляем скалярное произведение x[i: i + m]·wr для получения одного элемента y[i], где x[i : i + m] - участок х размера m. Данное вычисление повторяется таким же способом, который используется при работе со скользящим окном, чтобы иметь все элементы выхода. При этом фильтр wr(повернутый) каждый раз смещается на две ячейки. Это приводит к появлению такого гиперпараметром свертки как страйд s.

Дополнение входов для контроля размера выходных карт признаков

Формально дополнение может применяться с любым р 0. В зависимости от выбранного значения р граничные ячейки могут трактоваться иначе, чем ячейки, расположенные в середине х.

Рассмотрим пример n = 5 и m = 3. Тогда при р = 0 элемент х[0] используется только в вычислении одного выходного элемента (скажем, у[0]), в то время как х [1] применяется в вычислении двух выходных элементов (например, у[0] и у[1]). Таким образом, можно заметить, что это отличающееся обращение с элементами х способно искусственно придать больший вес среднему элементу, х[2], поскольку он появляется в большинстве вычислений. Выбрав р = 2, можно устранить проблему, и в таком случае каждый элемент х будет вовлечен в вычисление трех элементов у.

Размер выхода у также зависит от выбора используемой стратегии

дополнения. Существуют три режима дополнения, которые широко

применяются на практике: полный (full), одинаковый (same) и допустимый

(valid).

1. В полном режиме параметр дополнения р устанавливается в р = m - 1.

Полное дополнение увеличивает количество измерений вывода и потому редко используется в архитектурах сверточных нейронных сетей.

1. Одинаковое дополнение обычно применяется, если нужно гарантировать, что выходной вектор имеет такой же размер, как входной вектор х. В этом случае параметр дополнения р вычисляется в соответствии с размером фильтра вместе с требованием, чтобы размер входа и размер выхода совпадали.
2. Вычисление свертки в допустимом режиме относится к случаю,

когда р = 0 (дополнение отсутствует).

Самое распространенное дополнение в сверточных нейронных сетях-одинаковое(same). Его преимуществом заключается в том, что одинаковое дополнение предохраняет размер вектора (или высоту и ширину входных изображений при обработке задач компьютерного зрения), повышая удобство проектирования архитектуры сети.

Недостаток допустимого дополнения состоит в том, что в нейронных сетях с множеством слоев объем тензоров значительно уменьшается, и это может пагубно сказаться на эффективности сети. На практике рекомендуется предохранять пространственный размер, используя одинаковое дополнение для сверточных слоев, и взамен уменьшать пространственный размер посредством объединяющих слоев. Размер полного дополнения приводит к тому, что выход оказывается больше входа. Полное дополнение обычно применяется в приложениях обработки сигналов, где важно свести к минимуму граничные эффекты.

2.2.1 Определение размера выхода свертки

Размер выхода свертки определяется общим количеством сдвигов фильтра w вдоль входного вектора. Пусть размер входного вектора n, а

фильтр - размер m. Тогда размер выхода из у = х \* w с дополнением р и

страйдом s определяется следующим образом:

[]- скобки, в данном случае обозначают операцию округления в меньшую сторону.

Для демонстрации вычисления сверток в одном измерении

ниже показана наивная реализация, результаты которой сравниваются с

функцией numpy. convolve:

**import** numpy **as** np

**def** cvld **(**х**,** w**,** р**=**О**,** s**=**l**):**

rot\_w **=** np**.**array**(**w**[:** **:-**1**])**

pad\_x **=** np**.**array**(**x**)**

**if** p**>**O**:**

pad\_z **=** np**.**zeros**(**shape**=**p**)**

х\_padded **=** np**.**concatenate**([**pad\_z**,**

pad\_x**,**

pad\_z**]** **)**

res **=** **[]**

**for** i **in** **range(**O**,** **int(len(**x**)/**s**),**s**):**

res**.**append**(**np**.sum(**pad\_x **[**i**:**i**+**rot\_w**.**shape**[**O**]]** **\***rot\_w**))**

**return** np**.**array**(**res**)**

**## Выполним проверку**

х **=** **[**1**,** 3**,** 2**,** 4**,** 5**,** 6**,** 1**,** 3**]**

w **=** **[**1**,** О**,** 3**,** 1**,** 2**]**

**print** **(**'Реализация Convld: '**,**

cvld **(**х**,** w**,** р**=**2**,** s**=**l**))**

Реализация Convld**:** **[** 5. 14. 16. 26. 24. 34. 19. 22.**]**

**print** **(**'Результаты NumPy: '**,**

np**.**convolve**(**x**,** w**,** mode**=**'same'**))**

Результаты NumPy**:** **[** 5 14 16 26 24 34 19 22**]**

2.2.2 Выполнение дискретной свертки в двух измерениях

Предшествующая информация также распространяется на 2 измерения.

Пусть мы имеем двумерные входы:

матрица Xn1\*n2, и матрицей фильтра Wm1\*m2, где m1 n1 и m2 n2 ,

Получает что, результатом такой двумерной свертки между Х и W будет матрица Y= Х \* W:

Повороты матрицы фильтра, выбора страйдов и дополнения нулями, также расширяются на двумерные свертки при условии, что они работают независимо на оба измерения. На рис. 2.1 изображена двумерная свертка входной матрицы размером 8 \* 8 с использованием ядра размером 3 \* 3. Матрица входа дополняется нулями с р = 1. В итоге получаем, что выход такой двумерной свертки будет иметь размер 8 \* 8.

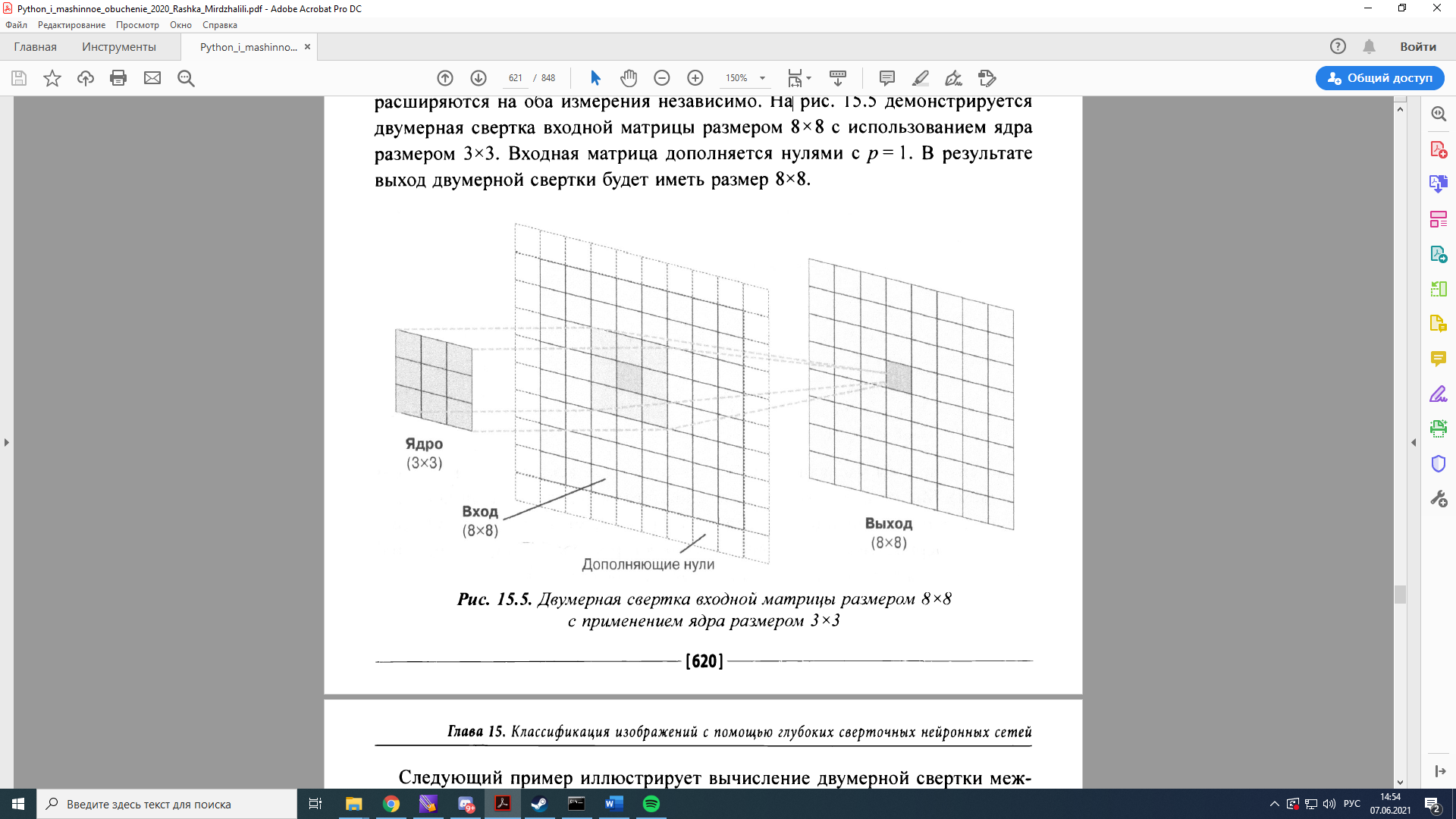


Рисунок 2.1 - Двумерная свертка 8 х 8 с применением ядра размером 3 x 3

Рассмотрим пример вычисления двумерной свертки между входной матрицей Х3\*3 и матрицей ядра W3\*3 с дополнением р = (1, 1) и страйдом s = (2, 2). Исходя из имеющегося дополнения входной матрицы, чтобы получить в итоге дополненную матрицу X5\*5 ДОП, к каждой стороне добавляется один слой нулей

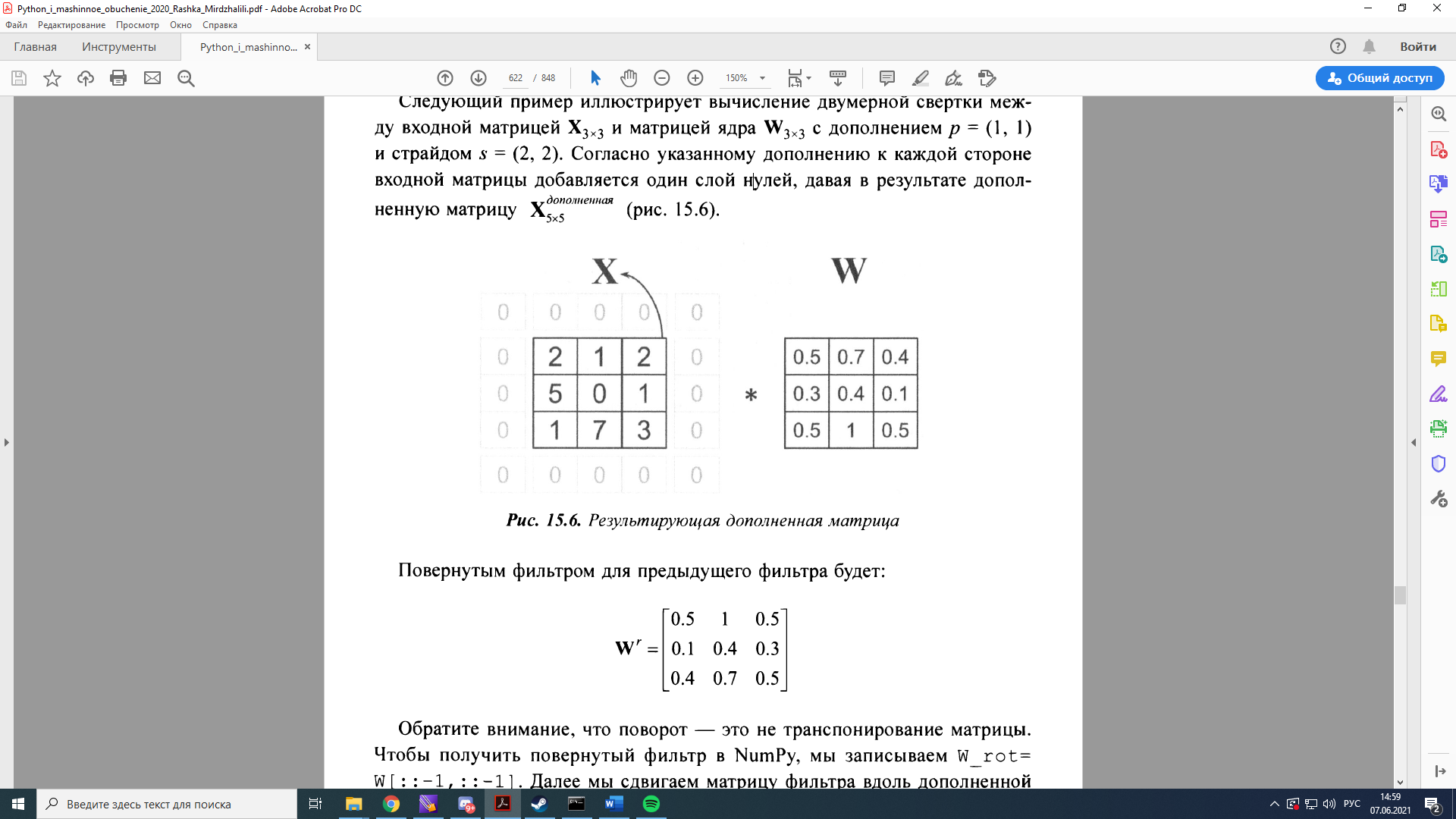


Рисунок 2.2 - Финальная матрица

Реализация двумерной свертки в соответствии с описанным алгоритмом.

Пакет scipy.signal предлагает способ вычисления двумерной свертки

через функцию scipy.signal.convolve2d:

**import** numpy **as** np

**import** scipy**.**signal

**def** cvl2d**(**X**,** W**,** р**=(**О**,** 0**),** s**=(**l**,** 1**)):**

W\_rot **=** np**.**array**(**W**)** **[:** **:-**1**,:** **:-**1**)**

X\_orig **=** np**.**array**(**X**)**

nl **=** X\_orig**.**shape**[**O**]** **+** 2**\***р**[**О**]**

n2 **=** x\_orig**.**shape**[**l**]** **+** 2**\***p**[**l**]**

Х padded **=** np**.** zeros **(** shape**=** **(** nl**,** n2**)** **)**

X\_padded**[**p**[**O**]:**p**[**O**]+**X\_orig**.**shape**[**O**],**

p**[**l**]** **:**p**[**l**]+**X\_orig**.**shape**[**l**]]** **=** X\_orig

res **=** **[]**

**for** i **in** **range(**O**,** **int(** **(**X\_padded**.**shape**[**O**]** **-** \

W \_ rot **.** shape **[**О**]** **)** **/** s **[**О**]** **)** **+** 1**,** s **[**О**]** **)** **:**

res**.**append**(** **[])**

**for** j **in** **range(**O**,** **int(** **(**X\_padded**.**shape**[**l**]** **-** \

W \_ rot**.** shape **[** 1**]** **)** **/** s **[** 1**]** **)** **+** 1**,** s **[** 1**]** **)** **:**

X\_sub **=** X\_padded**[**i**:**i**+**W\_rot**.**shape**[**O**],**

j**:**j**+**W\_rot**.**shape**[**l**]]**

res**[-**1**]** **.**append**(**np**.sum(**X\_sub **\*** W\_rot**))**

**return(**np**.**array**(**res**))**

Х **=** **([**1**,** З**,** 2**,** 4**],** **(**5**,** 6**,** 1**,** 3**],** **[**1**,** 2**,** О**,** 2**),** **(**3**,** 4**,** 3**,** 2**]]**

W **=** **[** **(**1**,** О**,** 3**],** **[**1**,** 2**,** 1**],** **(**0**,** 1**,** 1**]]**

**print** **(**'Реализация Conv2d: \n'**,**

cvl2d**(**X**,** W**,** p**=(**l**,** 1**),** s**={**l**,** 1**)))**

Реализация Conv2d**:**

**[** **[** 11. 25. 32. 13.**]**

**[** 19. 25. 24. 13.**]**

**[** 13. 28. 25. 17.**)**

**[** 11. 17. 14. 9.**))**

**print** **(**'Результаты SciPy: \n'**,**

scipy**.**signal**.**convolve2d**(**X**,** W**,** mode**=**'same'**))**

Результаты SciPy**:**

**[** **[**11 25 32 13**)**

**[**19 25 24 13**]**

**[**13 28 25 17**)**

**[**111 714 9**]]**

2.3 Слои подвыборки

Подвыборка в сверточных нейронных сетях используется в двух формах операций объединения: объединение по максимуму (max pooling) и объединение по среднему (mean-pooling или average-pooling).

Объединяющий слой записывается с использованием Рn1\*n2. В данном случае подстрочный индекс описывает количество смежных пикселей в каждом измерении (размер близлежащей области, где выполняется операция получения среднего или максимума). Происходит обращение на такую близлежащую область, которая называется размер объединения. Максимальное значение операция объединения по максимуму получает из близлежащей области пикселей, а операция объединения по среднему ищет их среднее значение.

Объединение по максимуму вносит локальную инвариантность. Из данного следует, что маленькие изменения в ближайшей локальной области не влияет на результат объединения по максимуму. Таким образом, генерирование более устойчивых к шуму во входных данных признаков опирается на инвариантность,

При этом из-за объединения происходит уменьшение размера признаков. В итоге происходит увеличение эффективности вычислений. Также из-за уменьшения числа признаков степень переобучения может значительно уменьшиться.

2.4 Работа с множественными входными или цветовыми каналами

Сверточный слой может иметь входом один и более двумерных массивов или матриц с измерениями N1 \* N2. В пример можно привести размерность изображения в пикселях (ширина и высота). Эти матрицы N1 \* N2 называются каналами. Сверточные слои при традиционной реализации получают на входе представления тензора третьего ранга (например, массива XN1\*N2 \* Свх, который является трехмерным, где Свх - количество входных каналов).

Теперь рассмотрим следующий пример. В качестве входа в первый слой сети CNN будет изображение. При этом Свх = 3 (для красного, зеленого и синего цветовых каналов в RGB), так как изображение цветное и цветовой режим RGB. Но если изображение представлено в серых оттенках, то Свх = 1, потому что есть имеется один канал со значениями интенсивности пикселей в оттенках серого.

Имея входом изображения их можно преобразовывать в массивы NumPy с использованием типа данных uint8 (8 - битное целое число без знака). Это сделано для сокращения расхода памяти в сравнении, например, с 16-,32- или 64-битными целочисленными типами. Значения 8-битных беззнаковых целых чисел имеют значения в диапазоне [0, 255]. Этого хватает, чтобы хранить информацию о пикселях изображений RGB, которые также имеют значения в том же самом диапазоне.

Пример кода, реализующий считывание изображения:

**import** tensorflow as tf

img\_example **=** tf**.**io**.**read\_file **(**'example. png'**)**

image **=** tf**.**image**.**decode\_image**(**img\_example**)**

**print** **(**'Формат данного изображения: '**,** image**.**shape**)**

Формат данного изображения**:** **(**220**,** 220**,** 3**)**

Теперь пример кода с использованием imageio.

**import** imageio

image **=** imageio**.** imread **(**'example. png'**)**

**print(**'Фopмaт изображения:'**,** img**.**shape**)**

Формат изображения**:** **(**220**,** 150**,** 3**)**

**print** **(**'Количество каналов:'**,** image**.**shape **[**2**])**

Количество каналов**:** 3

**print** **(**'Изображение имеет тип данных: ' **,**image**.**dtуре**)**

Изображение имеет тип данных**:** uint8

**print(**image**[**100**:**102**,** 100**:**102**,** **:))**

**[[** **[**179 134 110**]**

**(**182 136 112**]]**

**[(**180 135 11**]**

**(**182 137 113**]]]**

Чтобы операция свертки работала на все входные каналы, необходимо ее выполнять для каждого канала отдельно и и затем складываем результаты вместе, используя суммирование матриц. Следующее уравнение вычисляет общий результат предварительной активации:

Для заданного образца Xn1\*n2\*Cвх, матрицы ядра Wm1\*m2\*Cвх и значения смещения b

Предварительная активация:

Карта признаков:

A представляет собой финальный результат и в то же время является картой признаков. Сверточный слой сети в основном имеет более одной карты признаков. При использовании нескольких карт признаков тензор ядра становится четырехмерным, таким образом: ширина\* высота\* Свх\*Свых. Здесь ширина\*высота - размер ядра, Свх – количество входных каналов и Свых - количество выходных карт признаков. Обновим предыдущее уравнение включив него количество выходных карт признаков:

Для заданного образца Xn1\*n2\*Cвх, матрицы ядра Wm1\*m2\*Cвх\*Cвых и значения смещения bСвых

Рассмотрим следующий пример, представленный на рис. 2.3. На нем изображен сверточный слой, за которым следует объединяющий слой. На входе мы имеем 3 канала. При этом тензор ядра является четырехмерным.

Все матрицы ядра описываются как m1 \* m2. Мы имеем 3 точно таких матрицы, на входной канал приходится по 1 матрице. При этом на каждое из 5 пять ядер приходится по одному из выходных карт признаков. В конце расположен один объединяющий слой, предназначенный для подвыборки.

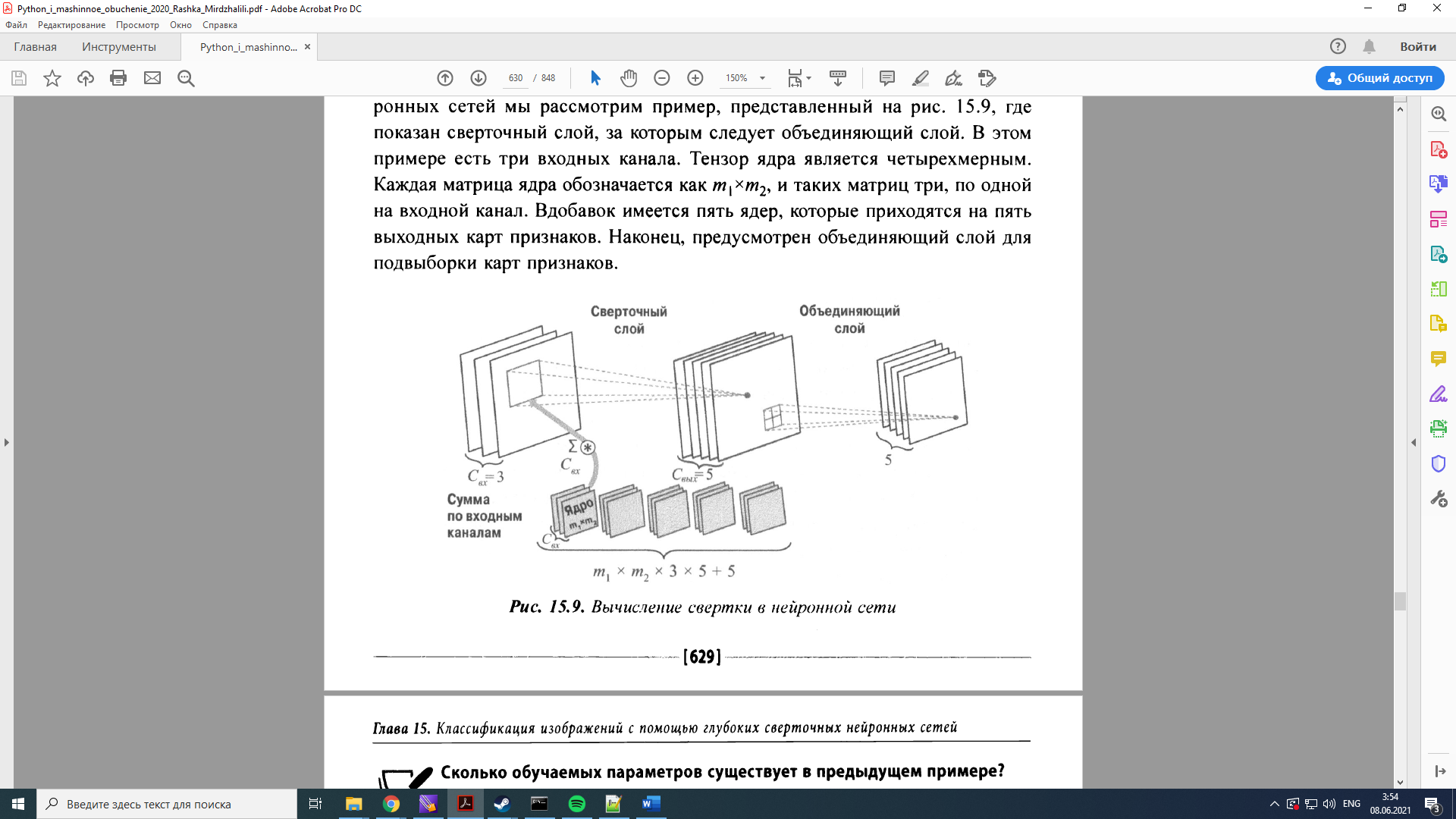


Рисунок 2.3 - Вычисление свертки в нейронной сети

2.5 Регуляризация нейронной сети с помощью отключения

Для регуляризации (глубоких) нейронных сетей в последние годы появилась популярная методика во избежание их переобучения,

которая называется отключением (dropout).

Dropout или отключение в основном используется в скрытых элементах более высоких слоев. Данная конструкция устроена следующим образом: во время стадии обучения нейронной сети некоторая доля скрытых элементов на каждой итерации случайным образом откидывается с вероятностью Ротбрасывания (или сохраняется с вероятностью Рсохранения = 1 - Ротбрасывания). Данная вероятность отключения часто выбирается вариантом р = 0.5 и задается пользователем, когда определенная часть нейронов на входе сети отбрасывается. Чтобы учесть отброшенные или недостающие нейроны, ассоциированные с оставшимися нейронами веса заново масштабируются.

Результатом использования случайного отключения является вынуждение сеть изучать избыточное представление данных. Из этого следует что, на активацию любого набора скрытых элементов сеть не может полагаться, поскольку они исходя из концепции могут быть отключены в любой момент во время обучения, и ей приходиться разыскивать наиболее надежные и общие шаблоны в имеющихся данных.

Случайное отключение такого типа способны эффективно препятствовать переобучению. На рис. 2.4 иллюстрируется применение отключения с вероятностью = 0.5 во время стадии обучения, благодаря чему половина нейронов случайным образом из активных преобразуются в неактивные (при каждом прямом проходе процесса обучения на случайной основе выбираются отбрасываемые элементы). Однако из-за работы выработки прогноза в вычислении предварительных активационных данных будут участвовать все нейроны следующего слоя.

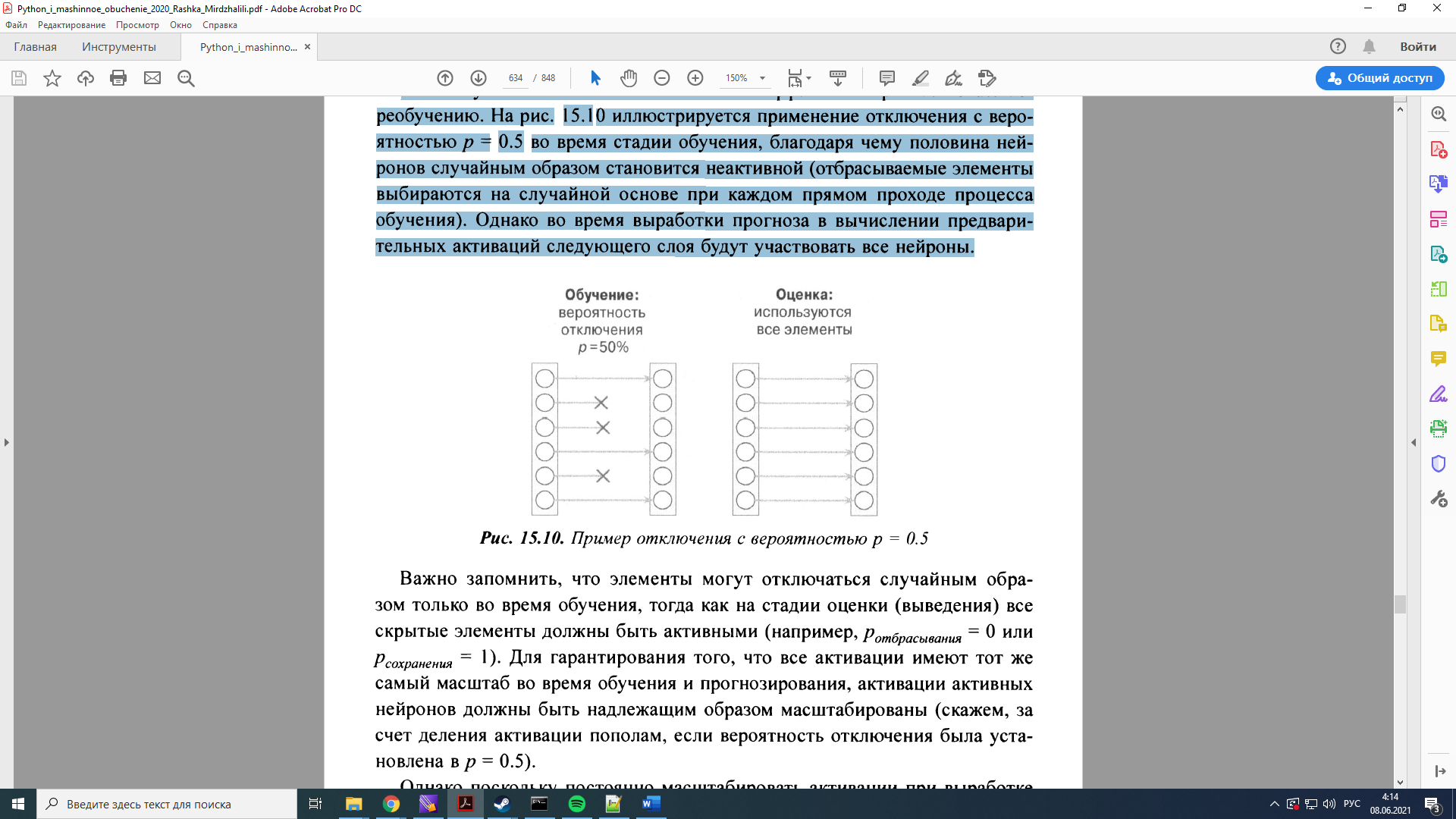


Рисунок 2.4 - Пример отключения с вероятностью р = 0.5.

Только во время обучения элементы могут отключаться случайным образом, но на стадии выведения или оценки все скрытые элементы должны быть активными (например, pотб = 0 или pсохр = 1, где pотб – вероятность отбрасывания, pсохр – вероятность сохранения). Чтобы обеспечить гарантирование того, что во время обучения и прогнозирования все активации имеют тот же самый масштаб, необходимо реализовать масштабирование активации активных нейронов специальным образом (например, можно использовать деления активации пополам, если вероятность dropout была установлена в р = 0.5).

2.6 Функции потерь для классификации

В задачах классификации используются различные функции потерь для обучения модели. Эта зависимость исходит от типа задачи (двоичная или многоклассовая) и типа выхода (логиты или вероятности). Функцией с единственным выходным элементом потерь для двоичной классификации выступает двоичная перекрестная энтропия. Для многоклассовой классификации, в тоже время используется категориальная перекрестная энтропия.

Для потери в виде категориальной перекрестной в АРl-интерфейсе Keras API энтропии исходя из формата представления достоверных меток имеются два варианта: в целочисленном виде (например, у=2) и в унитарном коде (например, [0, 0, 1, 0]). В контексте Keras такое представление также называется "разреженным".

Ниже на рис. 2.5 показаны доступные в Keras три функции потерь: двоичная классификация, многоклассовая классификация с достоверными метками в унитарном коде и многоклассовая классификация с целочисленными (разреженными) метками. Все три функций потерь обладают возможностью получения прогнозов в форме логитов или вероятностей членства в классах.

Обычно, по причинам численной устойчивости, вычисление потери в виде перекрестной энтропии с указанием логитов предпочтительнее

вероятностей членства в классах.

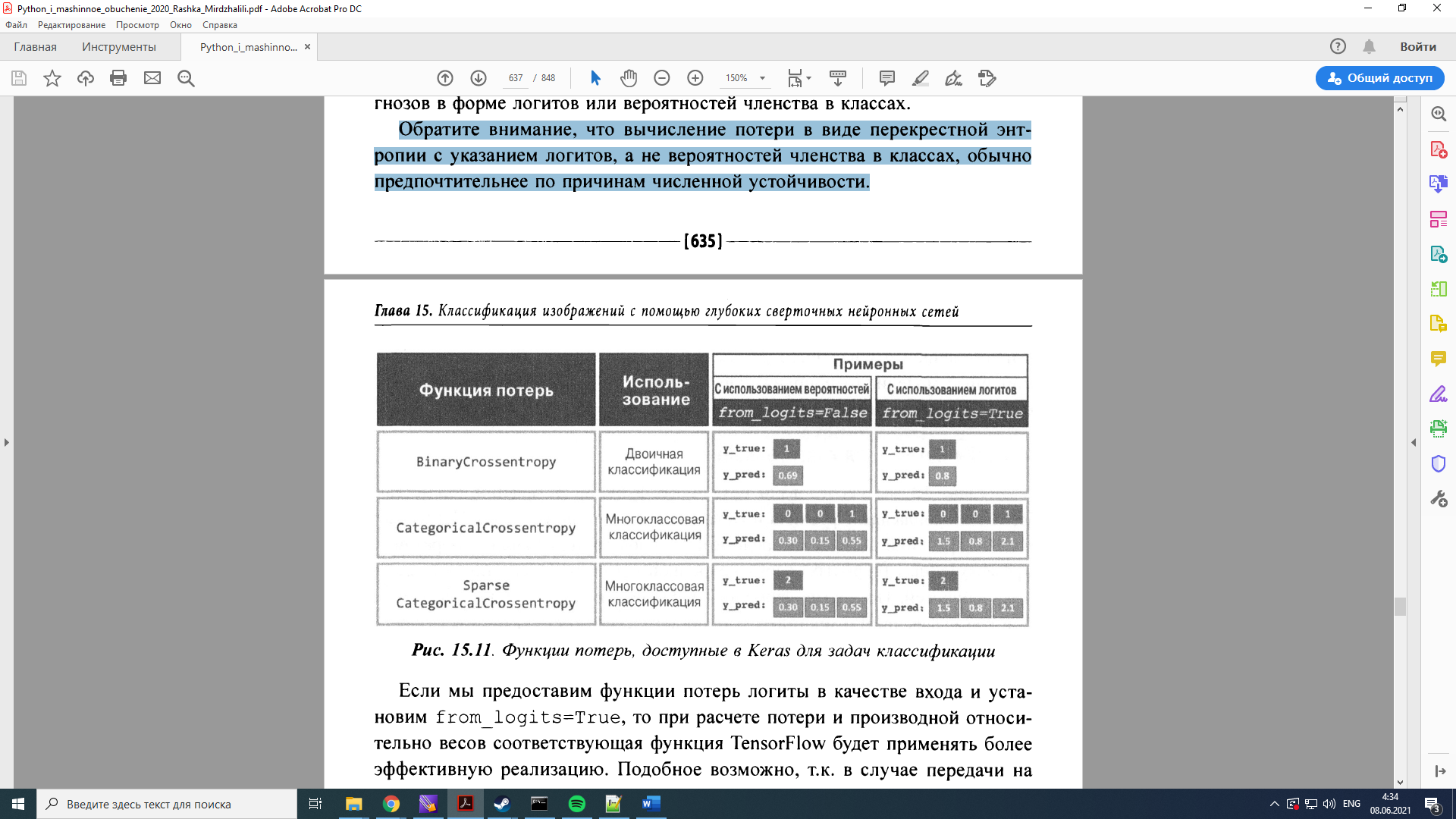


Рисунок 2.5 - Функции потерь для задач классификации

Представим для функции потерь логиты в качестве входа и установим параметр from \_ logits=True. В данном случае при расчете потери и производной относительно весов соответствующая функция Tensorflow будет использовать наиболее подходящую и эффективную реализацию. Такая ситуация реальна, так как в случае передачи на входе логитов определенные математические члены сокращаются и потому не требуют явного вычисления.

Следующий код показывает, как пользоваться функциями потерь с двумя форматами, где функции потерь на входе имеют либо вероятности нахождения в имеющихся классах, либо описанные ранее логиты:

**import** tensorflow\_datasets **as** dataset

# Перекрестная энтропия двоичная

bce\_probas **=**

tf**.** keras**.** losses**.** BinaryCrossentropy**(**from\_logits**=False)**

bce\_logit **=**

tf**.**keras**.**losses**.**BinaryCrossentropy**(**from\_logits**=** **True** **)**

logits **=** tf**.**constant**([**0.8**])**

probas **=** tf**.**keras**.**activations**.**sigmoid**(**logits**)**

tf**.print** **(**

'ДПЭ (с вероятностями ) : {:. 4f}'**.** **format(**

bce\_probas**(**y\_true**=[**l**],** y\_pred**=**probas**)),**

'(с логитами): {:.4f } '**.format(**

bce\_logits**(**y\_true**=[**l**],** y\_pred**=**logits**)))**

ДПЭ **(**с вероятностями**):** 0.3711 **(**с логитами**):** 0.3711

####### Перекрестная энтропия категориальная

Prob\_cce **=** tf**.**keras**.**losses**.**CategoricalCrossentropy**(**

**from** \_logi ts**=**E'alse)

logits\_cce **=** tf**.**keras**.**losses**.**CategoricalCrospentropy**(**logits=True**)**

logits **=** tf**.**constant**([[**l.5**,** 0.8**,** 2.1**]])**

probas **=** tf**.**keras**.**activations**.**softmax**(**logits**)**

tf**.print** **(**

'КПЭ (с вероятностями): {: .4f)' **.format(**

cce\_probas**(**y\_true**=[**0**,** 0**,** 1**],** y\_pred**=**probas**)** **),**

' (с логитами): {:. 4f)'**.** **format** **(**

cce\_logits**(**y\_true**=[**0**,** 0**,** 1**],** y\_pred**=**logits**)))**

КПЭ **(**с вероятностями**):** 0.5996 **(**с логитами**):** 0.5996

####### Разреженная категориальная перекрестная

энтропия **(**разреженная КПЭ**)**

sp\_cce\_probas **=** tf**.**keras**.**losses**.**

SparseCategoricalCrossentropy**(**

from\_logits**=False)**

sp\_cce\_logits **=** tf**.**keras**.**losses**.**

SparseCategoricalCrossentropy**(**

from\_logits**=True)**

tf**.print** **(**

'Разреженная КПЭ (с вероятностями): {:.4f) '**.format(**

sp\_cce\_probas**(**y\_true**=[**2**],** y\_pred**=**probas**)),**

' (с логитами): {:. 4f)'**.** **format** **(**

sp\_cce\_logits**(**y\_true**=[**2**],** y\_pred**=**logits**)))**

Разреженная КПЭ **(**с вероятностями**):** 0**.**6 **(**с логитами**):** 0**.**6

Необходимо также отметить, что иногда встречаются определенные реализации, в которых для двоичной классификации применяется потеря в виде категориальной перекрестной энтропии. В основном, когда у нас имеется задача двоичной классификации, такая модель возвращает одиночное выходное значение для каждого образца. Такой одиночный выход модели можно интерпретировать как некоторая вероятность положительного класса (например, класса 1), Р[класс = 1 ]. В задаче бинарной классификации получаем Р[класс =0] = 1 - Р[класс = 1]; чтобы получить вероятности отрицательного класса нам не нужен второй выходной элемент. Иногда для каждого обучающего образца возвращают два выхода и интерпретируют каждый класс и их вероятности следующим образом: Р [класс = 0] против Р[класс = 1].

В данном случае чтобы нормализировать выходы (чтобы их сумма была равно 1) надо пользоваться многопеременную функцией, и перекрестная категориальная энтропия оказывается подходящей функцией потерь.

# ГЛАВА 3. ОПИСАНИЕ РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1 Использованные средства

Для написания программного обеспечения был использован язык программирования Python 3.7 и следующие библиотеки:

Numpy – библиотека для работы с многомерными массивами

Pandas – библиотека ля обработки и анализа данных

Ffmpeg, Pydub – для реализации конвертера в .wav формат

PyWavelets, Scaleogram – библиотеки, для работы с вейвлет преобразованием и построеним спектрограммы.

SciPy – библиотека, для считывания данных с .wav файлов

TensorFlow – библиотека для реализации нейронных сетей.

PyQt5 – библиотека для реализации интерфейса программы.

Созданное приложение реализует весь требуемый функционал, а именно:

Конвертирует имеющиеся записи кашлей в .wav формат, производит вейвлет преобразование записей, отделяет записи кашля от речи и посторонних шумов, классифицирует записи по заболеваниям (с разделением на обычные, продуктивные, свистящие, а также записи Covid больных), делает предсказания по тестовым записям на принадлежность к какому-либо классу.

3.2 Структура программного обеспечения

Программное обеспечение состоит из 4 файлов .py:

main.py – главный файл нашего ПО, реализующий его основной функционал (конвертирование файлов, вейвлет преобразование, обучение нейронных сетей и предсказание)

func.py – файл, содержащий функции для реализации вейвлет преобразования и машинного обучения.

converter.py – файл, содержащий конвертер в .wav формат

form.py – файл для реализации интерфейса программы

3.3 Функционал программного обеспечения

main.py

Импортируем необходимые функции:

**from** pylab **import** **\***

**from** converter **import** convert

**from** func **import** wavelet**,** neuralNet**,**wavelet\_solo**,** learn

Конвертер для преобразования файлов в .wav формат:

formats\_to\_convert **=** **[**'.m4a'**]**

convertM4A**=**convert**(**formats\_to\_convert**)**

Прописываем пути к нашим файлам:

dir\_name1**=**"sounds/Ковид/"

dir\_name2**=**"sounds/Продуктивные/"

dir\_name3**=**"sounds/Свистящие/"

dir\_name4**=**"sounds/Обычные/"

Получаем список файлов в имеющихся папках для упрощения работы

вейвлет преобразования:

names1 **=** os**.**listdir**(**dir\_name1**)**

names2 **=** os**.**listdir**(**dir\_name2**)**

names3 **=** os**.**listdir**(**dir\_name3**)**

names4 **=** os**.**listdir**(**dir\_name4**)**

Выполняем вейвлет преобразование и формируем датасет:

wavelet1**=**wavelet**(**dir\_name1**,**names1**)**

wavelet2**=**wavelet**(**dir\_name2**,**names2**)**

wavelet3**=**wavelet**(**dir\_name3**,**names3**)**

wavelet4**=**wavelet**(**dir\_name4**,**names4**)**

Обучаем наши нейронные сети

learn\_1 **=** learn**()**

learn\_2 **=** learn**()**

Вызываем функцию, выполняющую предсказание по записи:

NN **=** neuralNet**()**

converter.py

**import** os

**import** argparse

**import** ffmpeg

**from** pydub **import** AudioSegment

**def** convert **(**A**,**dirpath**):**

**for** **(**dirpath**,** dirnames**,** filenames**)** **in** os**.**walk**(**"dirpath"**):**

**for** filename **in** filenames**:**

**if** filename**.**endswith**(tuple(**A**)):**

filepath **=** dirpath **+** '/' **+** filename

**(**path**,** file\_extension**)** **=** os**.**path**.**splitext**(**filepath**)**

file\_extension\_final **=** file\_extension**.**replace**(**'.'**,** ''**)**

**try:**

track **=** AudioSegment**.**from\_file**(**filepath**,**

file\_extension\_final**)**

wav\_filename **=** filename**.**replace**(**file\_extension\_final**,** 'wav'**)**

wav\_path **=** dirpath **+** '/' **+** wav\_filename

**print(**'CONVERTING: ' **+** **str(**filepath**))**

file\_handle **=** track**.**export**(**wav\_path**,** **format=**'wav'**)**

os**.**remove**(**filepath**)**

**except:**

**print(**"ERROR CONVERTING " **+** **str(**filepath**))**

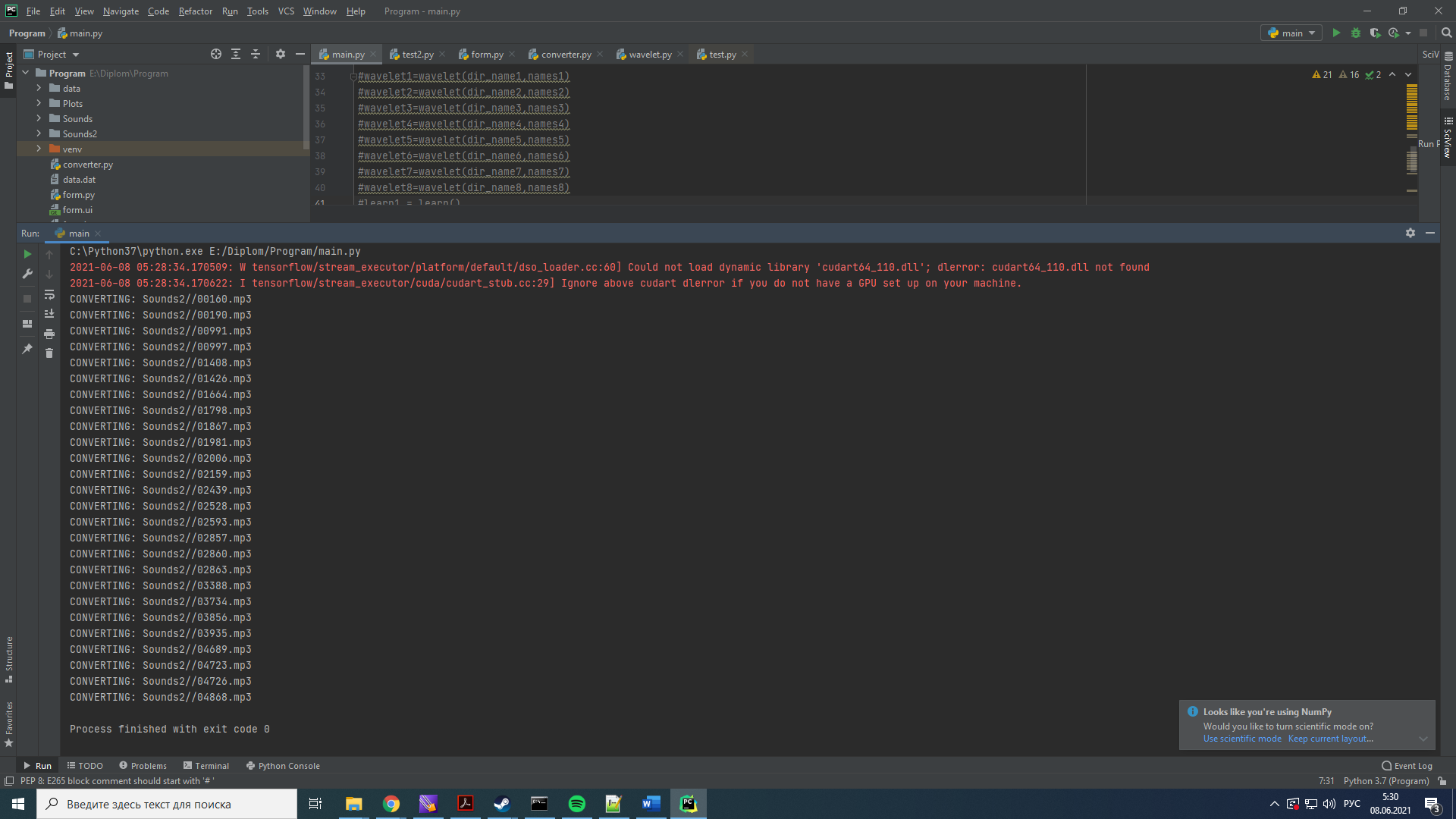


Рис. 3.1 Пример работы конвертера

Конвертер успешно конвертирует .m4a и .mp3 файлы в .wav и заменяет их в папке.

func.py

**def** wave**(**signal**,** scales**,** time**,** cmap **=** “jet”**,**

ylabel **=** 'Мощность'**,**

xlabel **=** 'Время'

title **=** 'Вейвлет-преобразование'**,):**

dt **=** time**[**1**]** **-** time**[**0**]**

**[**coef**,** freq**]** **=** pywt**.**cwt**(**signal**,** scales**,** waveletname**,** dt**)**

period **=** 1. **/** freq

lvls **=** **[**2**\*\*-**4 **,** 2**\*\*-**3 **,** 2**\*\*-**2 **,** 2**\*\*-**1 **,** 2**\*\***0 **,** 2**\*\***1 **,** 2**\*\***2 **,** 2**\*\***3**]**

ctrlvls **=** log2**(**lvls**)**

power **=** **(abs(**coef**))** **\*\*** 2

ax,fig **=** subplots**(**figsize**=(**10**,** 10**))**

im **=** ax**.**contourf**(**log2**(**period**),** log2**(**power**),** time**,** ctrlvls**,** extend**=**'both'**,**cmap**=**cmap**)**

yticks **=** 2**\*\***arange**))**

ax**.**setyticks**(**log2**(**yticks**))**

ax**.**setyticklabels**(**yticks**)**

ax**.**invert\_yaxis**()**

ax**.**set\_title**(**title**)**

ax**.**set\_ylabel**(**ylabel**)**

ax**.**set\_xlabel**(**xlabel**)**

yticks **=** 2**\*\***arange

ylim **=** ax**.**get\_ylim**()**

ax**.**setylim**(**ylim**[**0**],** **0)**

cbar **=** fig**.**add\_axes**([**0.8**,** 0.4**,** 0.04**,** 0.3**])**

show**()**

**(**rate**,** X**)** **=** wavfile**.**read**(**fullname**)**

X **=** X**[:,** **:**1**]**

data **=** pd**.**DataFrame**(**data**=**X**,** dtype**=float)**

df\_nino **=** pd**.**read\_csv**(**data**)**

N **=** data**[**0**]**

t0**=**0

dt**=**0.00004

signal **=** data**.**squeeze**()**

scales **=** arange**(**1**,** 128**)**

plot\_wave**(**time**,** signal**,** scales**)**

Данное вейвлет преобразование и построение масштабограммы использовалось в начале работы, но затем было принято решение перейти к библиотеке Scaleogram, так как она позывает большее быстродействие при идентичном результате. В данном случае это играет ключевую роль, ведь для создания датасета нам необходимо обработать больше 1300 записей, сделать их вейвлет преобразование и построить масштабограммы.

Реализация вейвлет преобразования с помощью библиотеки Scaleogram.

Сначала считываем наш файл, выделяем 1 канал (это делается на случай, если запись в стерео), делаем вейвлет преобразование и строим скалограмму. Затем полученная скалограмма сохраняется.

**def** wavelet**(**dir\_name**,**names**):**

**for** name **in** names**:**

fullname **=** os**.**path**.**join**(**dir\_name**,** name**)**

**(**rate**,** X**)** **=** wavfile**.**read**(**fullname**)**

name\_split **=** fullname**.**split**(**'/'**)**

length **=** X**.**shape**[**0**]** **/** rate

time **=** np**.**linspace**(**0.**,** length**,** X**.**shape**[**0**])**

t0=0

X **=** X**[:,** **:**1**]**

data **=** pd**.**DataFrame**(**data**=**X**,** columns**=[**'JJJ'**],** dtype**=float)**

dt=0.00002

time **=** t0 **+** arange**(len(**data1**))** **\*** dt

wavelet\_used **=** 'cmor1-1.5'

scales **=** scg**.**periods2scales**(**np**.**arange**(**1**,** 32**))**

ax **=** scg**.**cws**(**time**,** data1**,** scales**=**scales**,** wavelet**=**wavelet**,**

cmap**=**"jet"**,** cbar**=None,** xlabel**=**""**,** yscale**=**"linear"**,**

title**=**'Вейвлет преобразование'**)**

pad\_inches **=** 0

axis**(**'off'**)**

os**.**mkdir**(**'Plots/'**+**name\_split**[**1**])**

savefig**(**'Plots/' **+** **str(**name\_split**[**1**])+**'/' **+** name **+** '.png'**,** bbox\_inches**=**'tight'**,** pad\_inches **=** 0**)**

Пример полученного файла:

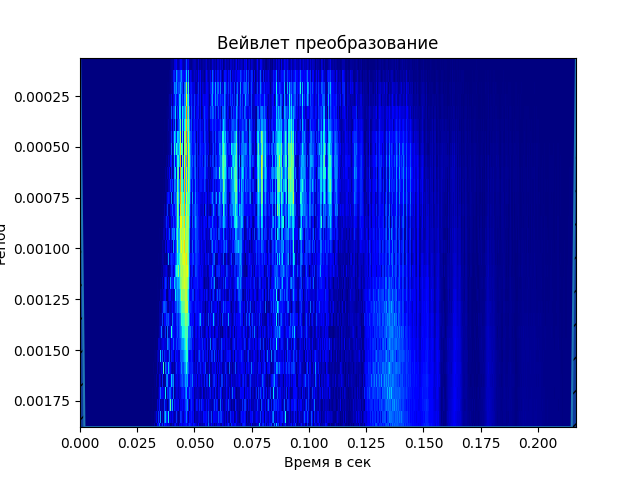


Рисунок 3.2 - Вейвлет - скалограмма записи кашля

Аналогичная вышеописанной функция, предназначение которой делать только одно вейвлет преобразование выбранного файла, в отличии от целой папки с файлами. Данная функция используется для упрощения работы с интерфейсом и помогает делать предсказания. Также она сохраняет визуализацию звукового файла для его отображения в дальнейшем.

**def** wavelet\_solo**(**fullname**):**

**(**rate**,** X**)** **=** wavfile**.**read**(**fullname**)**

name\_split **=** fullname**.**split**(**'/'**)**

length **=** X**.**shape**[**0**]** **/** rate

time **=** np**.**linspace**(**0.**,** length**,** X**.**shape**[**0**])**

plt**.**plot**(**time**,** X**)**

plt**.**xlabel**(**"Time [s]"**)**

plt**.**ylabel**(**"Amplitude"**)**

plt**.**title**(**'Звук'**)**

plt**.**savefig**(**'data/test/sound.png'**,**dpi**=**80**)**

**if** X**.**ndim **==** 2**:**

#print(X.ndim)

X **=** X**[:,** **:**1**]**

**else:**

**print(**X**.**ndim**)**

# plot(X)

data **=** pd**.**DataFrame**(**data**=**X**,** columns**=[**'JJJ'**],** dtype**=float)**

data1 **=** data**.**values**.**squeeze**()**

N **=** data1**.**size

t0 **=** 0**;**

dt **=** 0.00002

time **=** t0 **+** arange**(len(**data1**))** **\*** dt

wavelet **=** 'cmor1-0.5'

scales **=** scg**.**periods2scales**(**np**.**arange**(**1**,** 32**))**

ax **=** scg**.**cws**(**time**,** data1**,** scales**=**scales**,** wavelet**=**wavelet**,**

cmap**=**"jet"**,** cbar**=None,** xlabel**=**"Время в сек"**,** yscale**=**"linear"**,**

title**=**'Вейвлет преобразование'**)**

savefig**(**'data/test/show\_test.png'**)**

title**(**''**)**

axis**(**'off'**)**

savefig**(**'data/test/test.png'**,**bbox\_inches**=**'tight'**,** pad\_inches**=**0**)**

Для обучения используем функцию, основанную на TensorFlow

Входом нашей нейросети будут изображения размером 496\*369. Изображения цветные, соответственно количество каналов – 3.

**def** learn**():**

width**,** height **=** 496**,** 369

train\_dir **=** 'data/train3/'

validation\_dir **=** 'data/validation3/'

train **=** 1143

validation **=** 336

epochs **=** 20

batch\_size **=** 32

**if** K**.**image\_data\_format**()** **==** 'channels\_first'**:**

input\_shape **=** **(**3**,** width**,** height**)**

**else:**

input\_shape **=** **(**width**,** height**,** 3**)**

Используем Sequental класс, добавляем сверточные и объединяющие слои

Ядро 5\*5, слои объединения с размером 2\*2.

model **=** Sequential**()**

model**.**add**(**Conv2D**(**32**,** **(**5**,** 5**),** input\_shape**=**input\_shape**,** activation**=**'relu'**))**

model**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

model**.**add**(**Conv2D**(**64**,** **(**5**,** 5**)),** activation**=**'relu'**)**

model**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

model**.**add**(**Conv2D**(**64**,** **(**5**,** 5**)),** activation**=**'relu'**)**

model**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

Добавляем полносвязный и 2 плотных слоя. Также добавляем слой отключения и компилируем нашу сеть.

model**.**add**(**Flatten**())**

model**.**add**(**Dense**(**256**))**

model**.**add**(**Activation**(**'relu'**))**

model**.**add**(**Dropout**(**0.5**))**

model**.**add**(**Dense**(**4**))**

model**.**add**(**Activation**(**'softmax'**))**

model**.compile(**loss**=**'categorical\_crossentropy'**,**optimizer**=**'adam'**,**metrics**=[**'accuracy'**])**

Создаем тренировочную и проверочную выборки из имеющихся у нас скалограмм.

train\_data **=** ImageDataGenerator**(**

rescale**=**1. **/** 255**,**

shear\_range**=**0.2**,**

zoom\_range**=**0.2**,**

horizontal\_flip**=True)**

test\_data **=** ImageDataGenerator**(**rescale**=**1. **/** 255**)**

train\_data\_gen **=** train\_data**.**flow\_from\_directory**(**

train\_dir**,**

target\_size**=(**width**,** height**),**

batch\_size**=**batch\_size**,**

class\_mode**=**'categorical'**)**

validation\_data\_gen **=** test\_data**.**flow\_from\_directory**(**

validation\_dir**,**

target\_size**=(**width**,** height**),**

batch\_size**=**batch\_size**,**

class\_mode**=**'categorical'**)**

Обучаем нашу модель на данных и сохраняем вес в файл.

model**.**fit**(**

train\_data\_gen**,**

epochs**=**epochs**,**

validation\_data**=**validation\_data\_gen**,)**

model**.**save\_weights**(**'data/new\_weights\_2net.h5'**)**

Пример работы данного фрагмента:

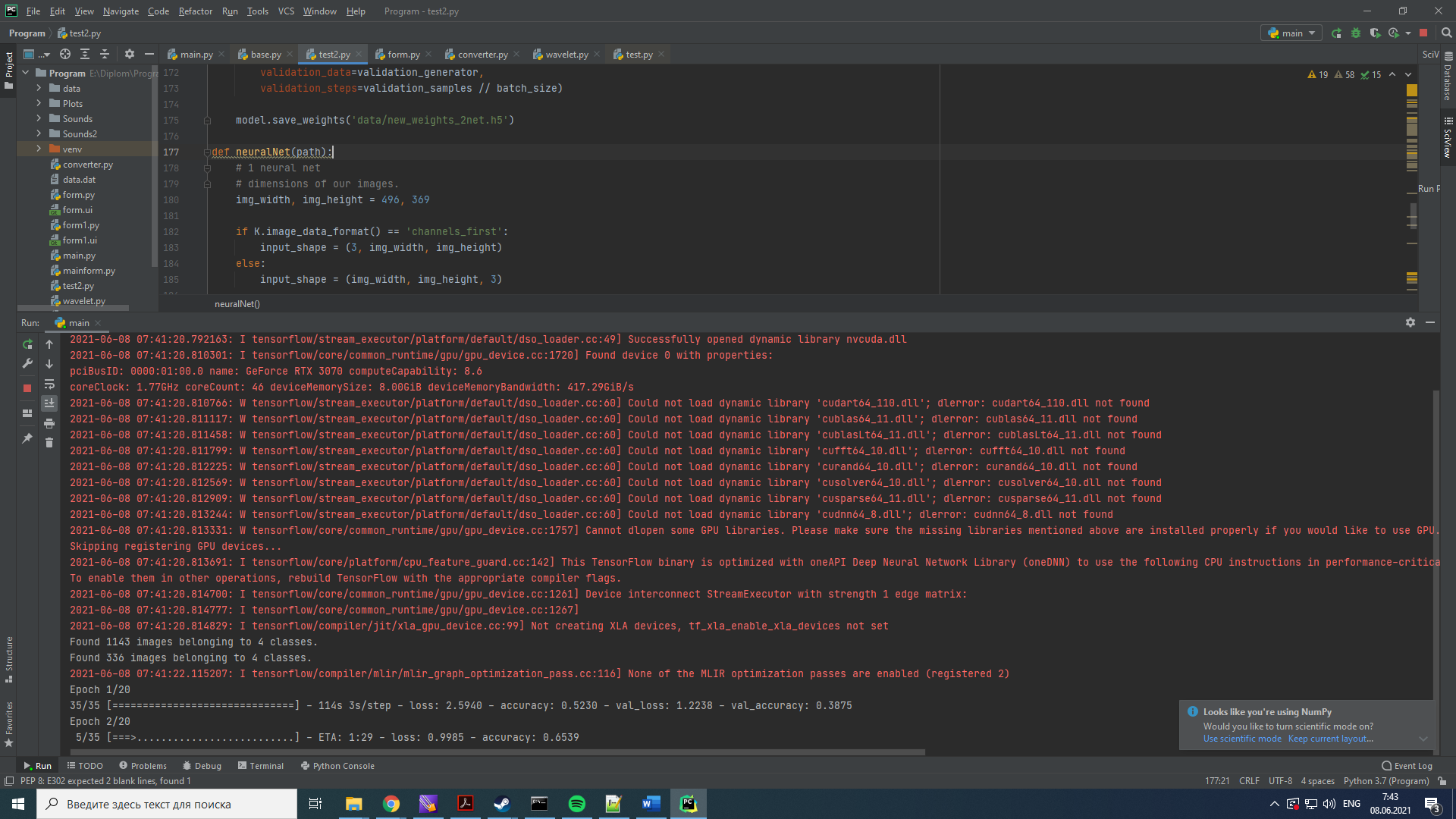


Рисунок 3.3 - Обучение нейросети

Функция neuralNet() компилирует 2 нейросети по алгоритму из функции learn() и загружает веса из файлов “weights1.h5” и “weights2.h5”.

**def** neuralNet**(**path**):**

# 1 neural net

# dimensions of our images.

img\_width**,** img\_height **=** 496**,** 369

**if** K**.**image\_data\_format**()** **==** 'channels\_first'**:**

input\_shape **=** **(**3**,** img\_width**,** img\_height**)**

**else:**

input\_shape **=** **(**img\_width**,** img\_height**,** 3**)**

model\_1 **=** Sequential**()**

model\_1**.**add**(**Conv2D**(**32**,** **(**3**,** 3**),** input\_shape**=**input\_shape**))**

model\_1**.**add**(**Activation**(**'relu'**))**

model\_1**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

model\_1**.**add**(**Conv2D**(**32**,** **(**3**,** 3**)))**

model\_1**.**add**(**Activation**(**'relu'**))**

model\_1**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

model\_1**.**add**(**Conv2D**(**64**,** **(**3**,** 3**)))**

model\_1**.**add**(**Activation**(**'relu'**))**

model\_1**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

model\_1**.**add**(**Flatten**())**

model\_1**.**add**(**Dense**(**64**))**

model\_1**.**add**(**Activation**(**'relu'**))**

model\_1**.**add**(**Dropout**(**0.5**))**

model\_1**.**add**(**Dense**(**2**))**

model\_1**.**add**(**Activation**(**'softmax'**))**

model\_1**.compile(**loss**=**'categorical\_crossentropy'**,**

optimizer**=**'adam'**,**

metrics**=[**'accuracy'**])**

# 2 neural net

model\_2 **=** Sequential**()**

model\_2**.**add**(**Conv2D**(**32**,** **(**3**,** 3**),** input\_shape**=**input\_shape**))**

model\_2**.**add**(**Activation**(**'relu'**))**

model\_2**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

model\_2**.**add**(**Conv2D**(**32**,** **(**3**,** 3**)))**

model\_2**.**add**(**Activation**(**'relu'**))**

model\_2**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

model\_2**.**add**(**Conv2D**(**64**,** **(**3**,** 3**)))**

model\_2**.**add**(**Activation**(**'relu'**))**

model\_2**.**add**(**MaxPooling2D**(**pool\_size**=(**2**,** 2**)))**

model\_2**.**add**(**Flatten**())**

model\_2**.**add**(**Dense**(**64**))**

model\_2**.**add**(**Activation**(**'relu'**))**

model\_2**.**add**(**Dropout**(**0.5**))**

model\_2**.**add**(**Dense**(**7**))**

model\_2**.**add**(**Activation**(**'softmax'**))**

model\_2**.compile(**loss**=**'categorical\_crossentropy'**,**

optimizer**=**'adam'**,**

metrics**=[**'accuracy'**])**

# Prediction

class\_list\_1 **=** os**.**listdir**(**'data/validation1'**)**

class\_list\_1 **=** **sorted(**class\_list\_1**)**

class\_list\_2 **=** os**.**listdir**(**'data/validation2'**)**

class\_list\_2 **=** **sorted(**class\_list\_2**)**

model\_1**.**load\_weights**(**'data/weights1.h5'**)**

model\_2**.**load\_weights**(**'data/weights2.h5'**)**

img **=** image**.**load\_img**(**path**,** target\_size**=(**496**,** 369**))**

x **=** image**.**img\_to\_array**(**img**)**

x **=** np**.**expand\_dims**(**x**,** axis**=**0**)**

x **=** x **\*** 1. **/** 255 # rescale as training

Данный блок реализует предсказания по 2 нейросетям. Если файл, проходя через 1-ую нейросеть, не является кашлем, то 2 нейросеть не выполняет предсказание.

prediction\_1 **=** model\_1**.**predict**(**x**)**

**print(**prediction\_1**)**

**print(**class\_list\_1**[**np**.**argmax**(**prediction\_1**)])**

**if** class\_list\_1**[**np**.**argmax**(**prediction\_1**)]** **==** 'Cough'**:**

**print(**"That's a cough"**)**

result1**=** "That's a cough."

prediction\_2 **=** model\_2**.**predict**(**x**)**

**print(**prediction\_2**)**

**print(**class\_list\_2**[**np**.**argmax**(**prediction\_2**)])**

result2 **=** class\_list\_2**[**np**.**argmax**(**prediction\_2**)]**

**else:**

**print(**"You selected not a cough"**)**

result1**=**"You selected not a cough."

result2**=**''

result **=** result1**+**' '**+**'Prediction: ' **+** result2

**print(**result**)**

**return** result

form.py

Импортируем необходимые библиотеки

**from** PyQt5 **import** QtCore**,** QtGui**,** QtWidgets

**from** PyQt5**.**QtWidgets **import** **(**QApplication**,** QWidget**,** QFileDialog**,** QTextEdit**,** QPushButton**,** QLabel**,** QVBoxLayout**)**

**from** PyQt5**.**QtGui **import** QPixmap

**from** func **import\***

Следующий блок реализует форму и был создан с помощью Qt Designer. C помощью clicked.connect привязываем на кнопку фукнцию get\_image\_file

**class** **Ui\_Dialog(**QWidget**):**

**def** setupUi**(**self**,** Dialog**):**

Dialog**.**setObjectName**(**"Dialog"**)**

Dialog**.**resize**(**500**,** 300**)**

sizePolicy **=** QtWidgets**.**QSizePolicy**(**QtWidgets**.**QSizePolicy**.**Expanding**,** QtWidgets**.**QSizePolicy**.**Expanding**)**

sizePolicy**.**setHorizontalStretch**(**0**)**

sizePolicy**.**setVerticalStretch**(**0**)**

sizePolicy**.**setHeightForWidth**(**Dialog**.**sizePolicy**().**hasHeightForWidth**())**

Dialog**.**setSizePolicy**(**sizePolicy**)**

self**.**verticalLayout **=** QtWidgets**.**QVBoxLayout**(**Dialog**)**

self**.**verticalLayout**.**setObjectName**(**"verticalLayout"**)**

self**.**file\_choose **=** QtWidgets**.**QPushButton**(**Dialog**)**

self**.**file\_choose**.**setObjectName**(**"file\_choose"**)**

self**.**verticalLayout**.**addWidget**(**self**.**file\_choose**)**

self**.**label\_soundimage **=** QtWidgets**.**QLabel**(**Dialog**)**

sizePolicy **=** QtWidgets**.**QSizePolicy**(**QtWidgets**.**QSizePolicy**.**Expanding**,** QtWidgets**.**QSizePolicy**.**Expanding**)**

sizePolicy**.**setHorizontalStretch**(**0**)**

sizePolicy**.**setVerticalStretch**(**0**)**

sizePolicy**.**setHeightForWidth**(**self**.**label\_soundimage**.**sizePolicy**().**hasHeightForWidth**())**

self**.**label\_soundimage**.**setSizePolicy**(**sizePolicy**)**

self**.**label\_soundimage**.**setText**(**""**)**

self**.**label\_soundimage**.**setScaledContents**(True)**

self**.**label\_soundimage**.**setObjectName**(**"label\_soundimage"**)**

self**.**verticalLayout**.**addWidget**(**self**.**label\_soundimage**)**

self**.**label\_image **=** QtWidgets**.**QLabel**(**Dialog**)**

sizePolicy **=** QtWidgets**.**QSizePolicy**(**QtWidgets**.**QSizePolicy**.**Expanding**,** QtWidgets**.**QSizePolicy**.**Expanding**)**

sizePolicy**.**setHorizontalStretch**(**0**)**

sizePolicy**.**setVerticalStretch**(**0**)**

sizePolicy**.**setHeightForWidth**(**self**.**label\_image**.**sizePolicy**().**hasHeightForWidth**())**

self**.**label\_image**.**setSizePolicy**(**sizePolicy**)**

self**.**label\_image**.**setText**(**""**)**

self**.**label\_image**.**setScaledContents**(True)**

self**.**label\_image**.**setObjectName**(**"label\_image"**)**

self**.**verticalLayout**.**addWidget**(**self**.**label\_image**)**

self**.**label\_result **=** QtWidgets**.**QLabel**(**Dialog**)**

sizePolicy **=** QtWidgets**.**QSizePolicy**(**QtWidgets**.**QSizePolicy**.**Expanding**,** QtWidgets**.**QSizePolicy**.**Expanding**)**

sizePolicy**.**setHorizontalStretch**(**0**)**

sizePolicy**.**setVerticalStretch**(**0**)**

sizePolicy**.**setHeightForWidth**(**self**.**label\_result**.**sizePolicy**().**hasHeightForWidth**())**

self**.**label\_result**.**setSizePolicy**(**sizePolicy**)**

self**.**label\_result**.**setText**(**""**)**

self**.**label\_result**.**setScaledContents**(True)**

self**.**label\_result**.**setObjectName**(**"label\_result"**)**

self**.**verticalLayout**.**addWidget**(**self**.**label\_result**)**

self**.**file\_choose**.**clicked**.**connect**(**self**.**get\_image\_file**)**

self**.**retranslateUi**(**Dialog**)**

QtCore**.**QMetaObject**.**connectSlotsByName**(**Dialog**)**

**def** retranslateUi**(**self**,** Dialog**):**

\_translate **=** QtCore**.**QCoreApplication**.**translate

Dialog**.**setWindowTitle**(**\_translate**(**"Dialog"**,** "Dialog"**))**

self**.**file\_choose**.**setText**(**\_translate**(**"Dialog"**,** "Выбор файла"**))**

С помощью данной функции выбирается файл для предсказания. Выбрать можно .wav файл или картинку масштабограммы. Если был выбрал .wav файл, то сначала выполняется вейвлет преобразование с помощью функции wavelet\_solo(). Далее, используя функцию neuralNet(), делаем предсказания, выводим изображения и результат предсказания.

**def** get\_image\_file**(**self**):**

file\_name**,** \_ **=** QFileDialog**.**getOpenFileName**(**self**,** 'Open Image File'**,** r"E:\\Diplom\\Program\\data\\"**)**

**if** file\_name**.**endswith**(**'.wav'**):**

**print(**file\_name**)**

wavelet\_test **=** wavelet\_solo**(**file\_name**)**

NN **=** neuralNet**(**'data/test/test.png'**)**

pixmap**=**QPixmap**(**'data/test/sound.png'**)**

small\_pixmap**=**pixmap**.**scaled**(**500**,** 400**,** QtCore**.**Qt**.**KeepAspectRatio**,** QtCore**.**Qt**.**SmoothTransformation**)**

pixmap2**=**QPixmap**(**'data/test/show\_test.png'**)**

self**.**label\_soundimage**.**setPixmap**(**pixmap**)**

self**.**label\_image**.**setPixmap**(**pixmap2**)**

self**.**label\_result**.**setText**(**NN**)**

**else:**

self**.**label\_image**.**setPixmap**(**QPixmap**(**file\_name**))**

#self.label\_image2.setText(file\_name)

NN **=** neuralNet**(**file\_name**)**

self**.**label\_result**.**setText**(**NN**)**

Инициализируем запуск формы.

**if** \_\_name\_\_ **==** "\_\_main\_\_"**:**

**import** sys

app **=** QtWidgets**.**QApplication**(**sys**.**argv**)**

Dialog **=** QtWidgets**.**QDialog**()**

ui **=** Ui\_Dialog**()**

ui**.**setupUi**(**Dialog**)**

Dialog**.**show**()**

sys**.exit(**app**.**exec\_**())**

3.3 Интерфейс программного обеспечения

Интерфейс программы выглядит следующим образом:

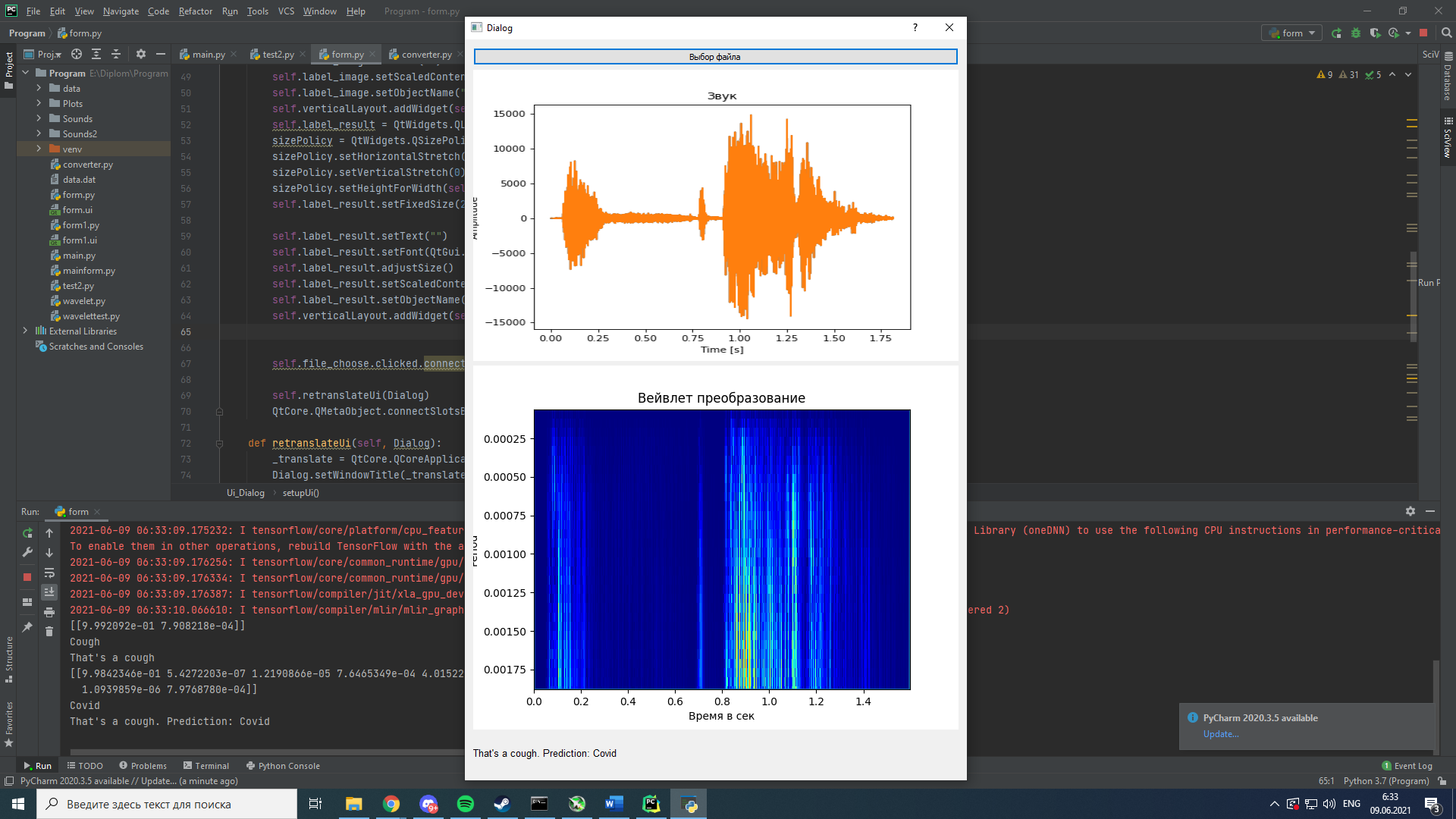


Рисунок 3.4 - Интерфейс рабочей программы

В верхней части программы находится кнопка, с помощью которой выбирается файл для предсказания. После выбора звукового файла выводятся 2 картинки: звуковая волна и спектрограмма. В самом низу находится результат предсказания нейросетей. Если 1 нейросеть классифицирует запись как “не кашель”, то вторая нейросеть не срабатывает и не выводит результат.

3.4 Примеры работы программы и обсуждение результатов

Сначала начнем с результатов обучения нейросетей. Точность 1 нейросети составляет 96%, второй – 85%. Данный результат является показателем хорошей работы сверточных нейросетей. Для сравнения точность обучения персептрона, способного классифицировать рукописные цифры с датасетом в 10000 записей в черно-белом формате составляет 90%. В нашем же случае мы используем цветные картинки масштабограмм и добиваемся схожих результатов при более сложных входных данных и сильно меньшем датасете. Очевидно, что точность 2 нейросети можно повысить практически до 99%, расширив датасет кашлей. В датасете было примерно 150 записей ковид больных и этого оказалось достаточно для получения хороших результатов.

Теперь рассмотрим несколько примеров скалограмм и визуально проанализируем их. Это поможет понять как нейросеть распознает разные типы кашлей и классифицирует их.

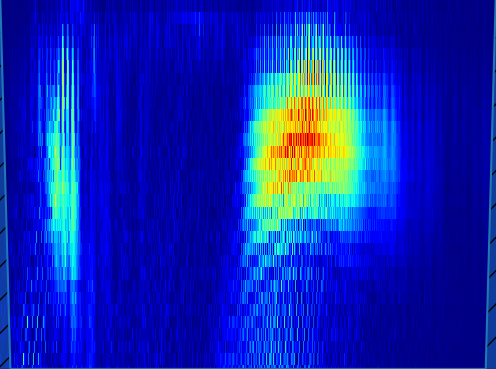


Рисунок 3.5 - Вейвлет - скейлограмма записи обычного кашля

При обычном кашле мы наблюдаем большое пятно, наиболее мощный сигнал находится внутри. Интенсивность распределения мощностей внутри пятна относительно равномерна и к центру увеличивается. Данные особенности характерны для большинства записей обычных кашлей.

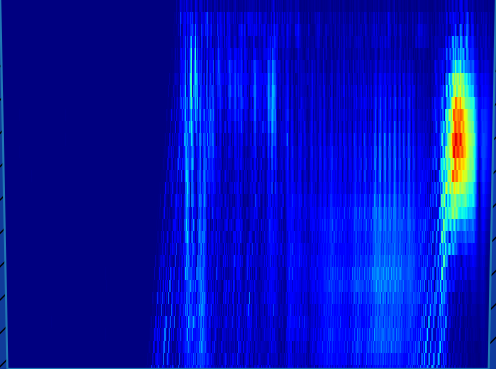


Рисунок 3.6 - Вейвлет - скейлограмма продуктивного кашля

Можно наблюдать, что пятно заметно уже, по сравнению с обычным кашлем. В остальном скейлограммы схожи.

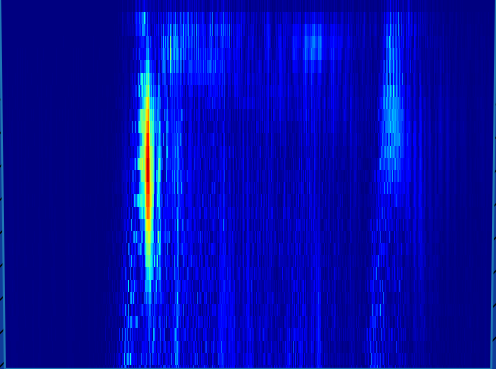


Рисунок 3.7 - Вейвлет - скалограмма свистящего кашля

При свистящем кашле это пятно еще более вытянуто по сравнению с продуктивным кашлем.

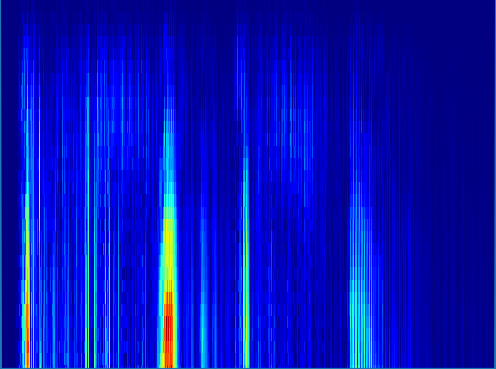


Рисунок 3.8 - Вейвлет - скейлограмма ковид кашля

В случае ковид кашля пятно вытянуто еще сильнее и представляет из себя полуовал.

Рассмотрев основные характеристики, можно понять как сверточная нейросеть классифицировала записи. Высокая точность при этом обьясняется особенностями сверточной нейросети, благодаря которым она может искать определенные контуры и особенности по всему изображению, даже если контур (например пятно или несколько пятен в нашем случае) меньше, чем в обучающей выборке. В этом состоит основное отличие сверточной нейросети от других типов нейросетей, что позволяет ей классифицировать изображения с высокой точностью.

# 

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы было реализовано приложение анализа респираторной активности, которое позволяет конвертировать имеющиеся записи кашлей в .wav формат, производить вейвлет преобразование записей, отделять записи кашля от речи и посторонних шумов, классифицировать записи по заболеваниям (с разделением на обычные, продуктивные и свистящие, а также записи Covid больных), делать предсказания по тестовым записям на принадлежность к какому-либо классу.

В ходе дипломной работы были решены следующие задачи:

1) Реализован алгоритм построения вейвлет-скалограммы

2) Создан датасет вейвлет-скалограмм для нейросетей

3) Созданы и обучены сверточные нейросети для анализа изображений, способные делать предсказания.

4) Произведен подбор оптимальных параметров для обучения нейросетей.

5) Произведен анализ работы сверточных нейросетей.

6) Реализован интерфейс для удобного отображения звуковых волн, вейвлет-скейлограмм и результатов предсказаний нейросетей.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кристалинский Р. Е. Преобразование Фурье и Лапласа в системах компьютерной математики. / Кристалинский Р. Е. М.: Горячая линия. Телеком.- 2006, 216 с. -URL: https://booktech.ru/books/matematika/15742-preobrazovaniya-fure-i-laplasa-v-sistemah-kompyuternoy-matematiki-2006-r-e-kristalinskiy

2. И. Добеши. Десять лекций по вейвлетам. / И. Добеши Пер. с англ. Е. В. Мищенко. Под ред. А. П. Петухова. М.:РХД, 2001, 464 с. -URL: http://www.manybooks.org/auth/12767/book/53468/dobeshi\_i/desyat\_lektsiy\_po\_veyvletam

3. К. Чуи. Введение в вейвлеты. / К. Чуи. Пер. с англ. под ред. Я. М. Жилейкина. М.: Мир, 2001, 412 с. -URL: http://www.library.ugatu.ac.ru/pdf/diplom/Chui\_Vvedenie\_v\_vejvlety\_2001.pdf

4. В. И. Воробьев. Теория и практика вейвлет - преобразований./ В. И. Воробьев, В. Г. Грибунин С. Пб.: ВУС.- 1999.-URL: https://www.studmed.ru/vorobev-vi-gribunin-vg-teoriya-i-praktika-veyvlet-preobrazovaniya\_0cab1ecc808.html

5. Дьяконов В. П. Вейвлеты. От теории к практике. Изд. 2-е переработанное и дополненное. М.:СолонПресс.- 2004,400 с. -URL: https://www.litres.ru/vladimir-dyakonov-2/veyvlety-ot-teorii-k-praktike-8337226/

6. Дьяконов В. П. Компьютерная математика. Теория и практика. / Дьяконов Нолидж.- 2001, 1296 с -URL: http://window.edu.ru/resource/378/20378/files/0101\_116.pdf

7. Себастьян Рашка Python и машинное обучение: машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow/ Себастьян Рашка, Вахид Мирджалили 2, 3-е изд.: Пер. с англ. - СПб. : ООО "Диалектика", 2020. - 848 с.: ил. - Парал. тит. англ. -URL: http://www.williamspublishing.com/Books/978-5-907203-57-0.html

8. Шолле Франсуа Глубокое обучение на Python./ Шолле Франсуа — СПб.: Питер, 2018. — 400 с.: ил. — (Серия «Биб-лиотекапрограммиста») -URL: https://codernet.ru/books/python/glubokoe\_obuchenie\_na\_python\_sholle\_fransua/