Реферат

Выпускная квалификационная работа состоит из 99 страниц, 39 рисунков, 28 формул, 27 источников литературы.

Ключевые слова: нейронные сети, рекуррентные нейронные сети, сверточные нейронные сети, классификация, электрокардиограмма.

Актуальность: нерешенность вполне задачи диагностики сердечно-сосудистых заболеваний (ошибки 1-го, 2-го родов) и, в этой связи, весьма актуально повышение точности диагностики, снижение вероятности совершить экспертом диагностические ошибки.

Цель работы: разработка многозначного классификатора, способного с определенной точностью относить данные ЭКГ к пяти классам: нормальный, инфаркт миокарда, изменение сегмента ST/T, нарушение проводимости, гипертрофия.

Задачи:

1) обзор и обработка набора данных ЭКГ сигналов;

2) выбор и разработка моделей нейронных сетей;

3) обучение и тестирование моделей.

Основные результаты: цель достигнута, все задачи решены.

Практическая значимость: полученную модель можно использовать для диагностики заболеваний по данным ЭКГ. Точность модели была получена в окрестности 80%. Результаты работы могут носить рекомендательный характер и составлять элемент алгоритмического аппарата соответствующей системы поддержки принятия решений.

Abstract

The graduate qualification work consists of 99 pages, 39 figures, 28 formulas, 27 sources of literature.

Keywords: neural networks, recurrent neural networks, convolutional neural networks, classification, electrocardiogram.

Relevance: the unsolved problem of diagnostics of cardiovascular diseases (errors of the 1st, 2nd labor) and, in this connection, it is very actual to increase the accuracy of diagnostics, to decrease the probability of making diagnostic errors by an expert.

Purpose of the work: development of a multivalued classifier capable of classifying ECG data with a certain accuracy into five classes: normal, myocardial infarction, ST/T segment change, conduction disturbance, hypertrophy.

Objectives:

1) review and processing of a dataset of ECG signals;

2) selection and development of neural network models;

3) training and testing of the models.

Main results: the goal was achieved, all tasks were solved.

Practical significance: the obtained model can be used to diagnose diseases from ECG data. The accuracy of the model was obtained in the neighborhood of 80%. The results of the work can be of a recommendatory nature and constitute an element of the algorithmic apparatus of the corresponding decision support system.

Содержание

Перечень сокращений

1D – one-dimensional (одномерный);

2D – two-dimensional (двумерный);

BCE - Binary Cross-Entropy (бинарная кросс-энтропия);

CNN - Convolutional Neural Network (сверточная нейронная сеть);

GAF – Gramian Angular Fields (грамианские угловые поля);

GRU – Gated Recurrent Unit (управляемый рекуррентный блок);

MTF - Markov Transition Field (марковское переходное поле);

NN – Neural Network (нейронная сеть);

RNN – Recurrent Neural Network (рекуррентная нейронная сеть);

RP – Recurrence Plot (график повторения);

UC – use case (вариант использования);

МО – машинное обучение;

ПО – программное обеспечение;

ЭКГ – электрокардиограмма;

Введение

Сердечно-сосудистые заболевания являются основной причиной смерти во всем мире [1]. Правильная диагностика работы сердце – ключ к правильно подобранному курсу лечения.

Электрокардиография – одна из основных диагностических мероприятий при обследовании сердца. Этот метод используется для регистрации и изучения электрических полей, возникающих при работе сердца. Результатом электрокардиографии является получение электрокардиограммы (ЭКГ).

Электрокардиограмма — это запись электрической активности сердца. Он предоставляет информацию о частоте сердечных сокращений, ритме и проведении электрических импульсов в сердце.

По данным ЭКГ можно распознать множество заболеваний, не только сердечных. Расшифровать показания ЭКГ для выявления патологий довольно сложная задача. 5-6 зубцов; 12 отведений (может быть и больше); интервалы между зубцами, комплексами зубцов, сегментами; высота зубцов; относительная высота зубцов; в каждом отведении своя картина и свои нормы показателей; связи между отведениями и зубцами; и многие другие характеристики – всё это надо учитывать при анализе ЭКГ [2].

Ручная интерпретация ЭКГ сигнала – сложная и утомительная работа, поэтому существует вероятность человеческой ошибки в процессе анализа даже для экспертов с многолетним стажем.

Целью данной дипломной работы является автоматизация диагностики заболевания по данным электрокардиограммы на основе алгоритмов машинного обучения. Полученная система может стать помощником в заключение диагноза экспертом.

Актуальность задачи автоматизации диагностики заболеваний по данным ЭКГ обусловлена несколькими факторами:

1) нерешенностью вполне задачи диагностики практически для любых заболеваний (ошибки 1-го, 2-го родов) и, в этой связи, весьма актуально повышение точности диагностики, снижение вероятности совершить экспертом диагностические ошибки;

2) потребностью создания базовых алгоритмов для создания и внедрения автоматизированных систем диагностики, позволивших бы улучшить доступность качественной диагностики в удаленных и малонаселенных регионах, где не хватает квалифицированных врачей;

2) быстрым ростом публикаций с новыми методами анализа данных, главным образом, основанных на алгоритмах машинного обучения и их комбинациях, корректное использование которых способствует развитию научных исследований в области биомедицинской инженерии и медицинской информатики.

Традиционные методы классификации сигнала ЭКГ в основном основаны на ручной обработке или ручном извлечении признаков такими методами, как:

- методы цифровой фильтрации [3],
- сочетание экспертных методов [3],
- пороговые методы [5],
- анализ главных компонент [6],
- преобразования Фурье [7],
- вейвлет-преобразования [8].

Некоторые из классификаторов, используемых с этими извлеченными признаками, — это

- машины опорных векторов [9],
- скрытые марковские модели [10],
- нейронные сети [11].

Главным недостатком этих традиционных методов является разделение части извлечения признаков и части классификации паттернов. Кроме того, эти методы требуют экспертных знаний о входных данных [13].

Современные исследователи данной задачи сосредоточили внимание на использование глубоких нейронных сетей разных архитектур для ее решения [13], [23].

В данном курсовом проекте реализуются и сравниваются три сети для достижения цели:

- рекуррентные GRU и двунаправленный GRU;
- сверточный AlexNet.

Так как сверточным сетям необходимы изображения, исходные данные преобразуются в двумерные представления тремя разными способами, которые характеризуют их (данные) с разных сторон.

1. Постановка задачи

Целью проекта является разработка многозначного классификатора, способного с определенной точностью относить данные ЭКГ к пяти классам: нормальный, инфаркт миокарда, изменение сегмента ST/T, нарушение проводимости, гипертрофия.

Классификатором является глубокая нейронная сеть.

Задачи для достижения цели:

1) анализ предметной области;
2) обзор существующих способов решения;
3) разработка ПО для удобной обработки данных и настройки параметров моделей МО конкретно для данной задачи;
4) обзор и обработка набора данных ЭКГ сигналов;
5) выбор и разработка моделей машинного обучения;
6) обучение и тестирование разработанных моделей;

В результате обученные модели будут доступны для диагностирования новых данных.

2. Анализ предметной области

Сокращению любой мышцы сопутствует электрические изменения, именуемые "деполяризацией", и эти изменения могут быть зафиксированы с помощью электродов, наложенных на поверхность тела. Так как при этом регистрируются все мышечные сокращения, для записи электрических изменений, связанных с сокращениями сердечной мышцы, пациент должен быть полностью расслаблен и никакие скелетные мышцы не должны в этот момент сокращаться [2].

Запись электрической активности сердца называется электрокардиограммой.

Чтобы получить ЭКГ с 12 отведениями, 10 электродов размещаются на конечности и грудь (Рисунок 1). Четыре электрода размещаются на левых и правых руках и ногах. Остальные шесть размещаются на передней поверхности грудной клетки и обозначаются V1 – V6. Расположение этих электродов следующее:

- V1 – четвертое межреберье справа от грудины;
- V2 – четвертое межреберье слева от грудины;
- V4 – пятое межреберье по среднеключичной линии;
- V3 – середина пути между V2 и V4;
- V5 – пятое межреберье, на том же самом уровне как электрод V4, но расположен по левой передней подмышечной линии[1];
- V6 – средняя подмышечная линия[2], на том же самом уровне, что и V4 и V5.

[1] Левая передняя подмышечная линия - воображаемая вертикальная линия, которая простирается от передней складки подмышки.

[2] Средняя подмышечная линия - воображаемая вертикальная линия, проведенная с середины подмышки.

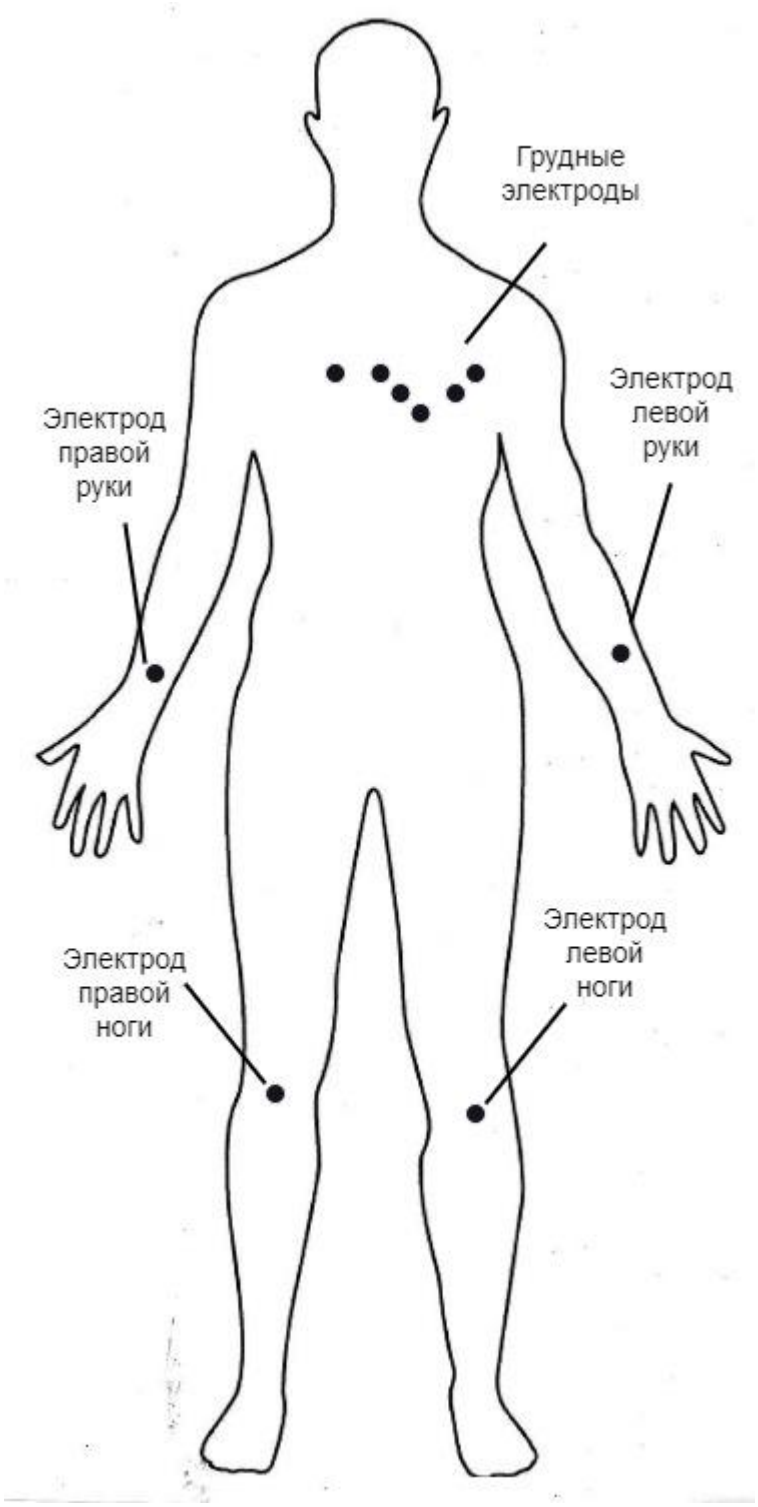

Рисунок 1. Стандартное расположение 10 электродов для записи 12 отведений ЭКГ.

Чтобы измерить любой вид электрической активности, требуются два электрода, чтобы измерительный прибор мог определить разность потенциалов между ними. Кардиограмма традиционно использует 12 отведений для определения активности сердца. Они обозначаются как I, II, III,

aVR, aVL, aVF, V1, V2, V3, V4, V5, V6 [14]. Отведения классифицируются следующим образом:

1) стандартные (двухполюсные) отведения. Регистрация стандартных отведений от конечностей проводится при попарном подключении электродов:

- I стандартное отведение — левая рука (+) и правая рука (-);
- II стандартное отведение — левая нога (+) и правая рука (-);
- III стандартное отведение — левая нога (+) и левая нога (-).

Электроды накладываются на левой руке, правой руке и левой ноге. На правую ногу накладывается 4-й электрод для подключения к заземляющему проводу;

2) усиленные однополюсные отведения от конечностей. Однополюсные отведения характеризуются наличием только одного активного — положительного — электрода, отрицательный электрод индифферентен и представляет собой «объединенный электрод Гольберга», который образуется при соединении через дополнительное сопротивление двух конечностей:

- aVR — отведение от правой руки;
- aVL — от левой руки;
- aVF — от левой ноги;

3) грудные отведения. Грудные отведения в ЭКГ являются однополюсными. Активный электрод присоединяется к положительному полюсу электрокардиографа, а объединенный от конечностей тройной индифферентный электрод — к отрицательному полюсу аппарата. Грудные отведения принято обозначать буквой V:

- V1 — активный электрод располагают в IV межреберье у правого края грудины;
- V2 — в IV межреберье у левого края грудины;
- V3 — между IV и V межреберьями по левой окологрудинной линии;
- V4 — в V межреберье по левой среднеключичной линии;
- V5 — в V межреберье по передней подмышечной линии;
- V6 — и V межреберье по средней подмышечной линии.

Запись электрокардиограммы проводится при спокойном дыхании пациента. Сначала — в I, II, III стандартных отведениях, далее — в усиленных однополюсных отведениях от конечностей (aVR, aVL, aVF), затем — в грудных отведениях V1. V2, V3, V4, V5, V6. В каждом из отведений следует регистрировать не менее 4-х сердечных циклов [15].

Таким образом, ЭКГ имеет следующий вид (Рисунок 2)

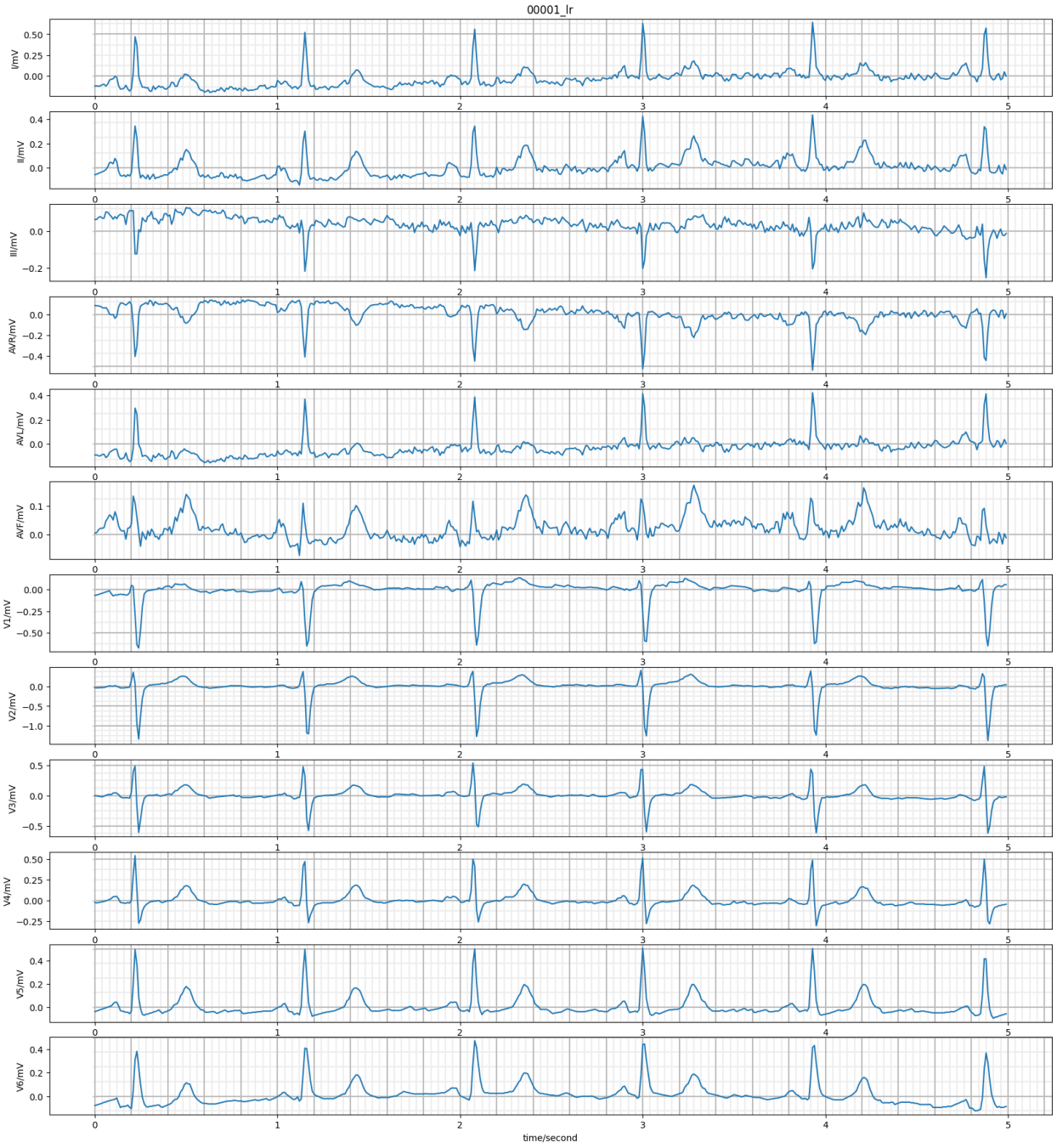

Рисунок 2. Пример ЭКГ. Нормальный синусовый ритм

На записи ЭКГ различают колебания P, Q, R, S и T (иногда и U, который следует после зубца T), которые называются зубцами (Рисунок 3).

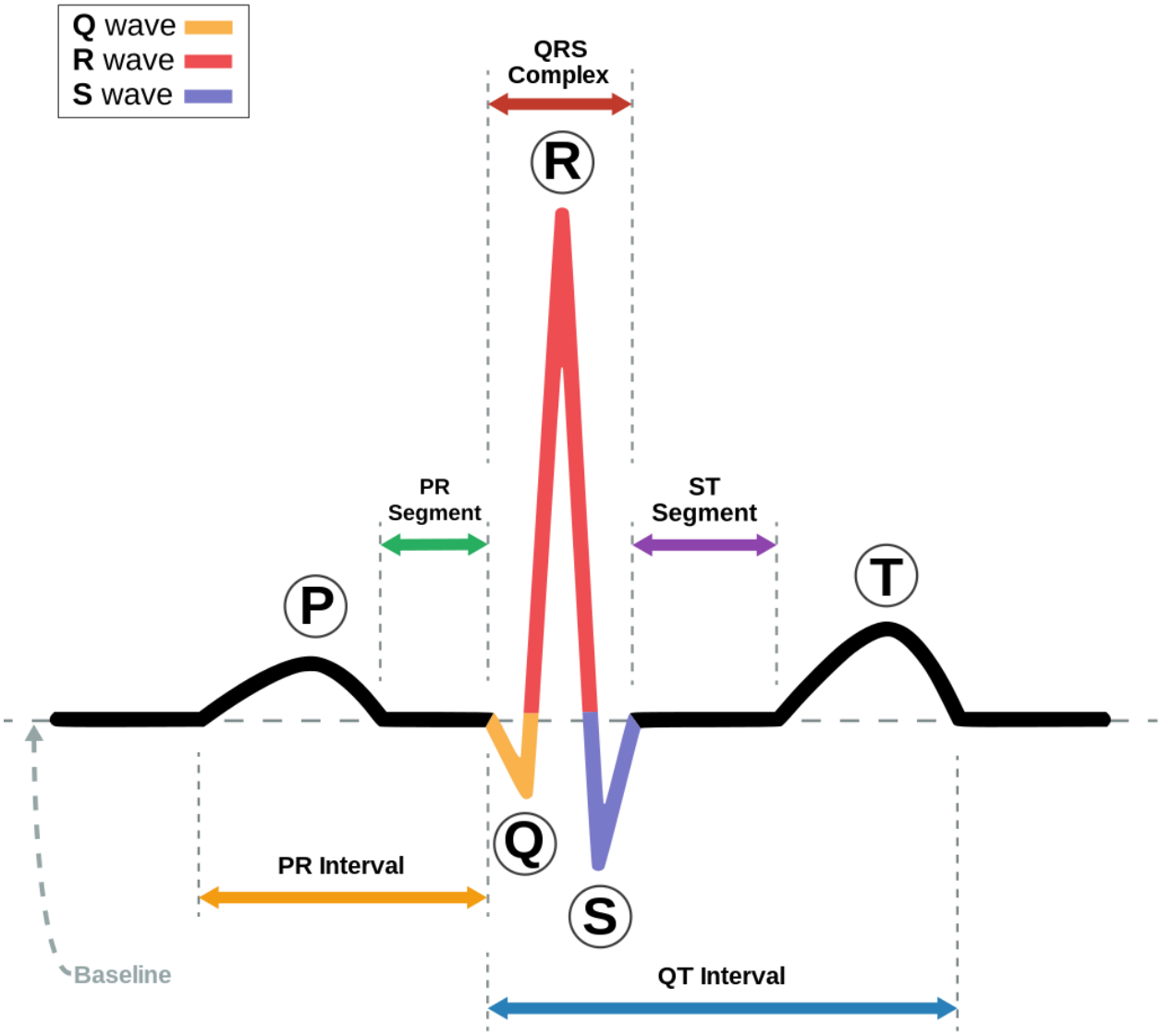

Рисунок 3. Базовая форма нормальной ЭКГ[3]

Электрическая активность, произведенная деполяризацией предсердий, регистрируется на ЭКГ в виде зубца P. Далее идет желудочковая деполяризация, которая приводит к большому отклонению, названному комплексом QRS. Первое отрицательное отклонение называют зубцом Q, первое положительное отклонение – зубец R, а любое отрицательное отклонение после R – зубец S. После сегмента QRS регистрируется

[3] https://commons.wikimedia.org/wiki/File:SinusRhythmLabels.svg#/media/File:SinusRhythmLabels.svg/2

изоэлектрический период, называемый ST сегментом. Наконец, когда сокращения желудочков закончены, они начинают реполяризовываться и возвращаться в их первоначальное состояние, чтобы подготовиться к следующему циклу деполяризации/сокращения. Желудочковая реполяризация сопровождается зубцом Т [14].

Анализ ЭКГ – это распознавание паттернов, т.е. отнесение электрокардиографических образов (форма зубцов, комплексов и их сочетания) к определенной патологии [2].

3. Требования к ПО

3.1. Общее описание

ПО, о котором пойдет речь в данном разделе напрямую не связан с целью проекта, но представляет лишь графический интерфейс для настройки гиперпараметров нейронных сетей и их обучения, а также для обработки данных. ПО предназначено для программистов.

ПО имеет 5 основных страниц:

1) "Обработка данных" – предназначена для скачивания и обработки датасетов;

2) "Нейронные сети"– предназначена для выбора нейронной сети, которая будет обучаться;

3) "Гиперпараметры" – предназначена для установки параметров обучения;

4) "Обучение модели" – предназначена для запуска процесса обучения;

5) "Результаты обучения" – предназначена для просмотра результатов обучения;

3.2. Операционная среда

- Операционная система: Windows 11, версия 23H2;
- Система запускается как скрипт python. Версия Python 3.12.1.

3.3. Ограничения дизайна и реализации

- Весь код должен быть написан на языке Python с использованием библиотеки CustomTkinter
- Код должен быть форматирован согласно стандарту PEP8.

3.4. Требования к странице "Обработка данных"

Рисунок 4. Диаграмма вариантов использования для страницы "Обработка данных"

Таблица 1. Варианты использование страницы "Обработка данных"

UC-1	При нажатии кнопки "Скачать датасет" открывается диалоговое окно. В диалоговом окне требуется указать url-адрес к zip-каталогу датасета в интернете, а также имя директории, куда будет распакован zip-каталог. После нажатия кнопки "Скачать" начинается скачивание. Если такое имя уже существует, выводится текст:

	"Датасет с таким именем уже существует" и скачивание не начинается. Прогресс скачивание отображается под кнопкой "Скачать". Датасет сохраняется по пути "./data/raw/", где "." корень проекта.
UC-2	Пользователь выбирает частоту дискретизации 100 Гц или 500 Гц.
UC-3	Пользователь выбирает, какие из 12 отведений будут участвовать в обработке данных и обучении сетей.
UC-4	Пользователю предлагается два вида обработки данных: 1D и 2D. При нажатии кнопки "Файл для обработки данных", открывается python-файл в редакторе для соответствующего вида обработки с функциями подготовки данных. При нажатии кнопки "Обработать данные" открывается диалоговое окно, в котором пользователь может выбрать датасет из "./data/raw/", и указать имя директории (не полный путь), в котором будут сохранены подготовленные данные. При нажатии кнопки "Обработать" начинается процесс обработки. Если имя директории, в котором будут сохранены данные уже существует, то выводиться сообщение "Такое имя уже существует" и процесс обработки не начинается. Обработанные данные хранятся на путях "./data/processed/1D" и "./data/processed/2D", соответственно выбранному пользователем 1D или 2D обработке.
UC-5	Переходит на страницу выбора нейронной сети.

3.5. Требования к странице "Нейронные сети"

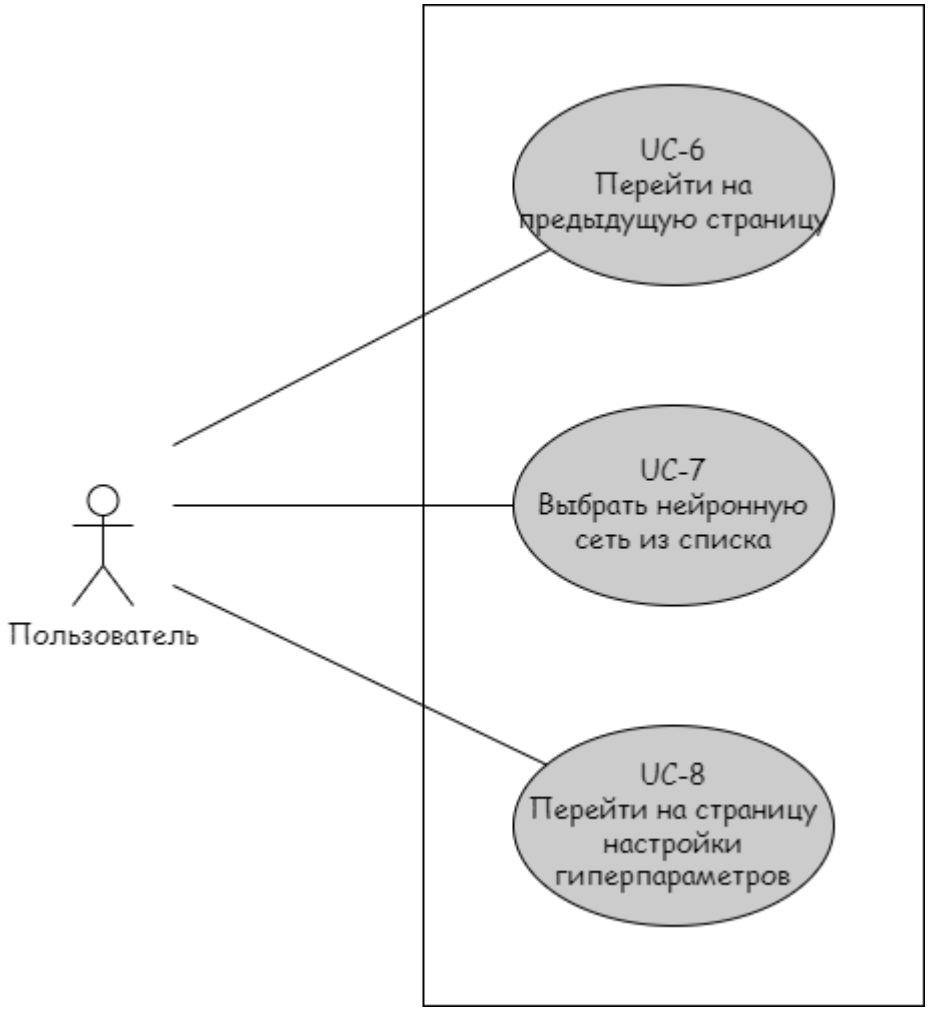

Рисунок 5. Диаграмма вариантов использования для страницы "Нейронные сети"

Таблица 2. Варианты использование страницы "Нейронные сети"

UC-6	Возвращается на страницу "Обработка данных".
UC-7	Пользователь выбирает нейронную сеть для обучения. Список формируется из имеющихся реализаций в "./src/nn1d/" и "./src/nn2d/".
UC-8	Переходит на страницу настройки гиперпараметров обучения.

3.6. Требования к странице "Гиперпараметры"

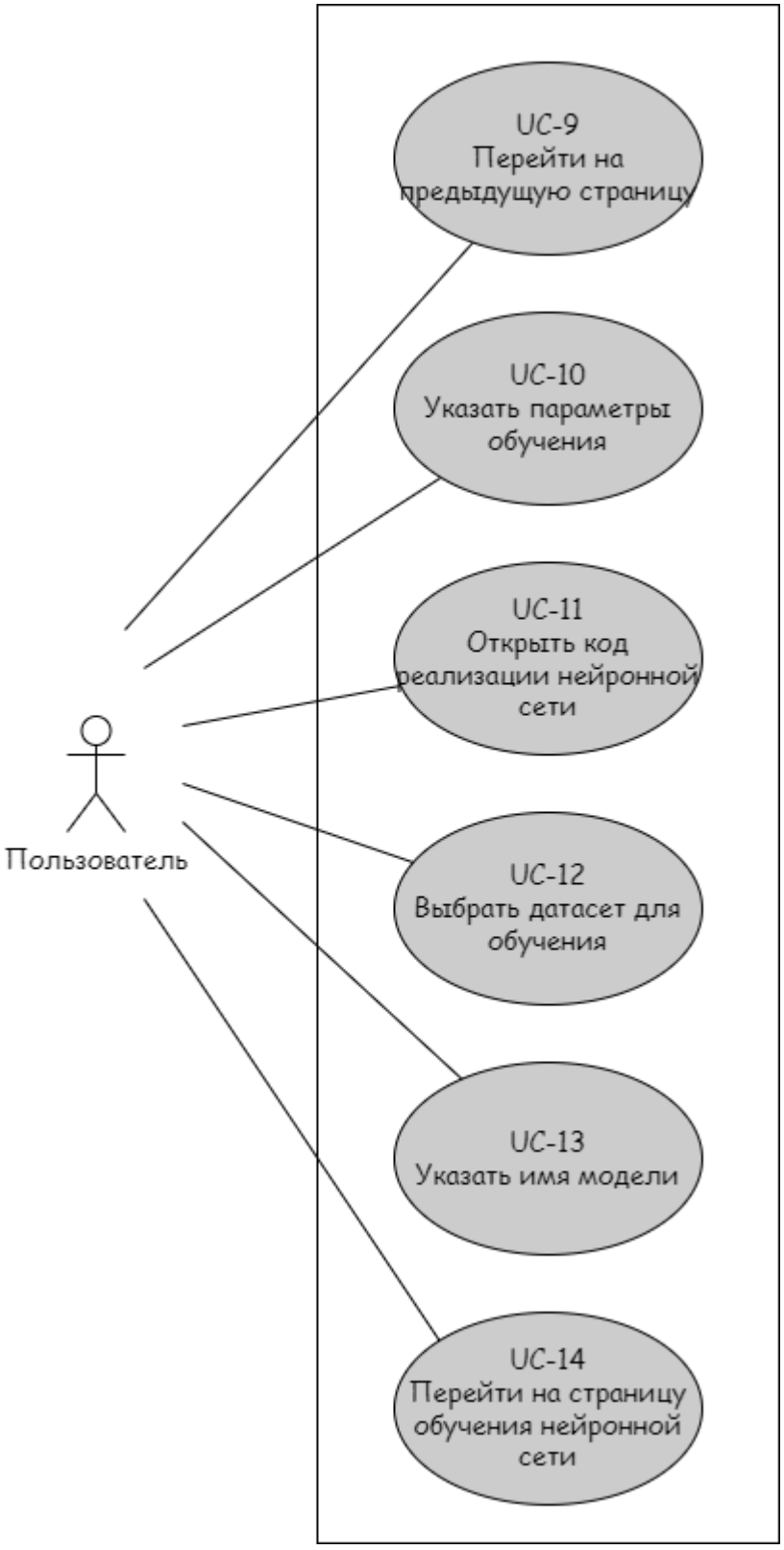

Рисунок 6. Диаграмма вариантов использования для страницы "Гиперпараметры"

Таблица 3. Варианты использование страницы "Гиперпараметры"

UC-9	Возвращается на страницу "Нейронные сети".
UC-10	Пользователю предлагается ввести следующие параметры: • epochs – количество эпох обучения (целое положительное число), • batch_size – размер мини-пакета (целое положительное число), • learning_rate – скорость обучения (число в диапазоне [0, 1]), • l2_decay – L2-регуляризация (число в диапазоне [0, 1]), • optimizer – выпадающий список с выбором между "adam" и "sgd", • device – выпадающий список с выбором между "cuda", "cpu" и "mps". При нарушении ограничений красным подсвечивается соответствующий параметр.
UC-11	Открывает в редакторе python-файл с код реализации выбранного на предыдущей странице нейронной сети.
UC-12	Пользователю предлагается выбрать из списка возможных подготовленные данные. Список составляется из обработанных данных, которые находятся на путях "./data/processed/1D" и "./data/processed/2D", соответственно тому, какая нейронная сеть была выбрана на предыдущей странице: из "./src/nn1d/" или "./src/nn2d/".
UC-13	Пользователю предлагается ввести имя модели, под которым будет сохранена модель и результаты обучения. Если такое имя уже существует в "./models" или в "./reports", то поле подсвечивается красным и выводиться сообщение: "Такое имя уже существует". При отсутствии имени поле также подсвечивается красным и выводиться сообщение: "Недопустимое имя".

UC-14	Переходит на страницу обучения модели. Если не выбраны подготовленные данные, или нарушены ограничения параметров или имени модели, то переход не происходит.

3.7. Требования к странице "Обучение модели"

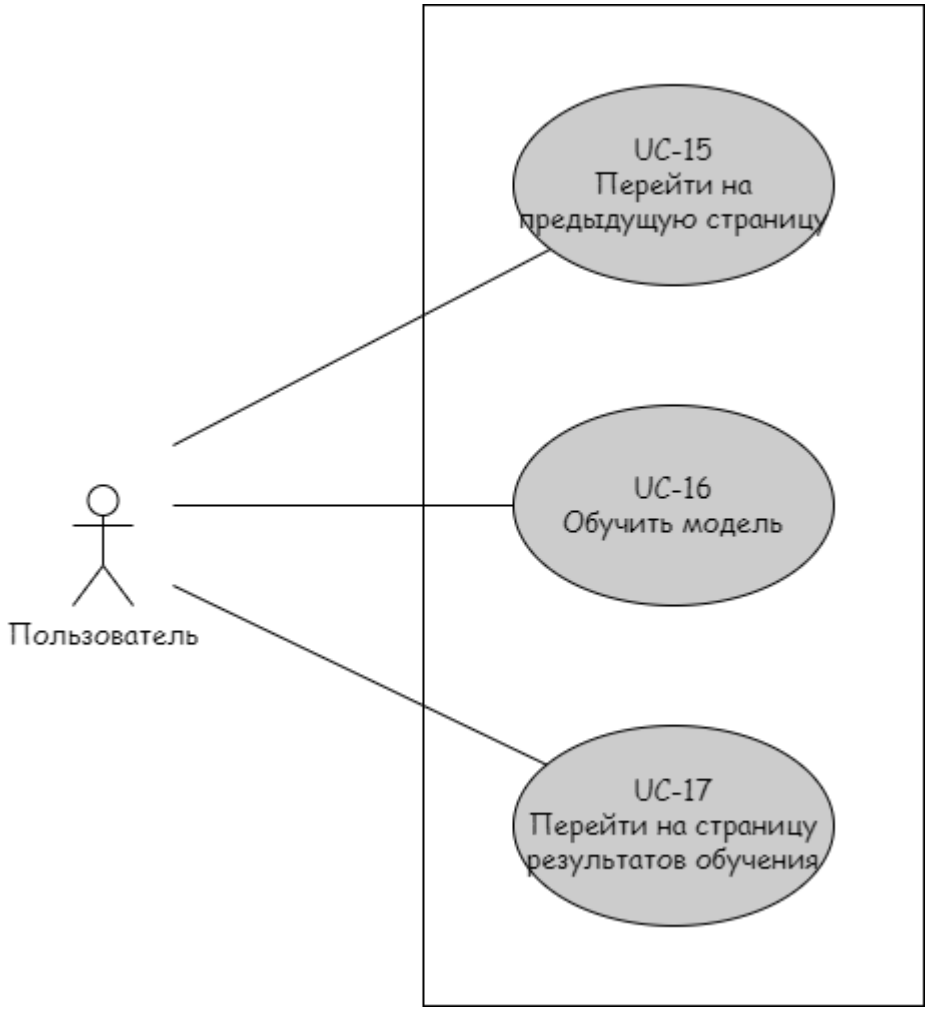

Рисунок 7. Диаграмма вариантов использования для страницы "Обучение модели"

Таблица 4. Варианты использование страницы "Обучение модели"

UC-15	Возвращается на страницу "Гиперпараметры".
UC-16	При нажатии кнопки "Обучить модель" начинается процесс обучения модели. Варианты UC-15 и UC-16 становятся недоступны.

	Отображается прогресс обучения. По завершении обучения, UC-17 становиться доступным.
UC-17	Переходит на страницу результатов обучения. Недоступно до тех пор, пока не завершится обучение, начатое в UC-16.

3.8. Требования к странице "Результаты обучения"

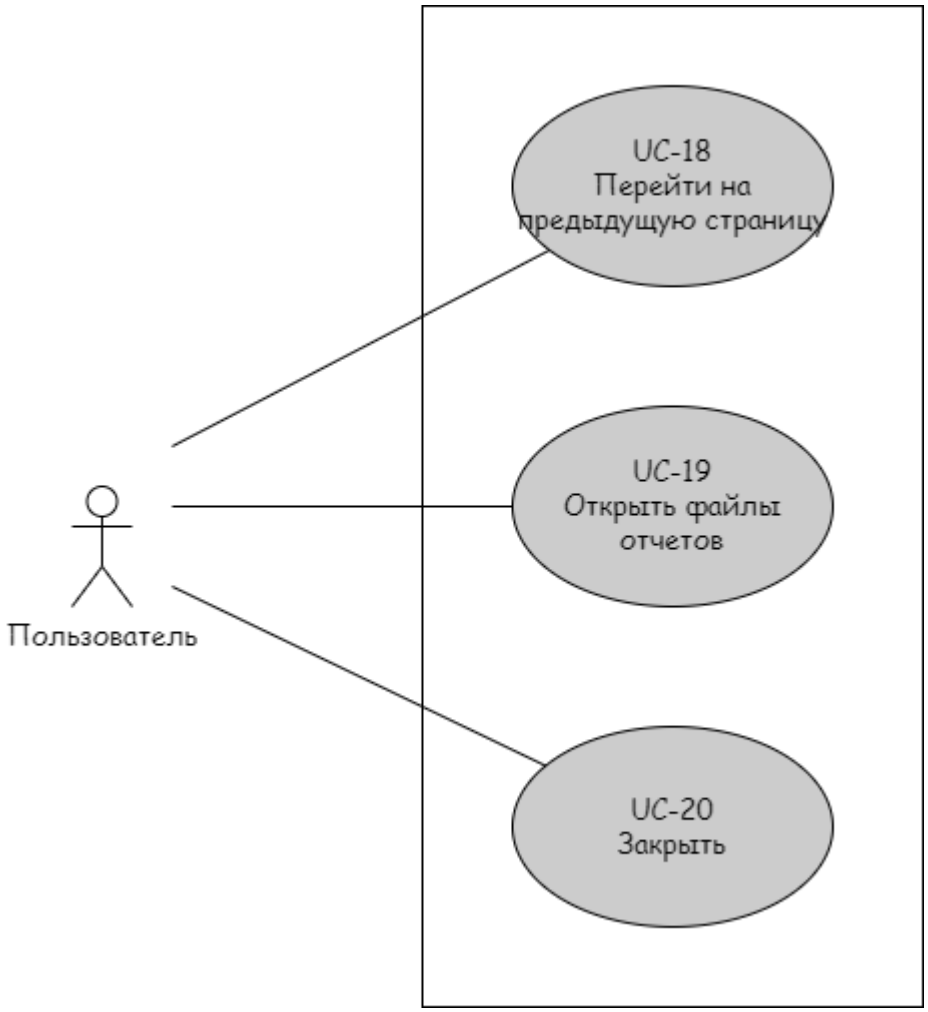

Рисунок 8. Диаграмма вариантов использования для страницы "Результаты обучения"

Таблица 5. Варианты использование страницы "Результаты обучения"

UC-15	Возвращается на страницу "Обучение модели".
UC-16	Пользователь может открывать соответствующие результаты обучения из "./reports/model_name/". (Пояснение: по завершении

	обучения отчеты сохраняются в "./reports/model_name/", где "model_name", указанное в UC-13) .
UC-17	Закрывает приложение

4. Результат разработки ПО

Код представлен в Приложении А.

4.1. Страница "Обработка данных"

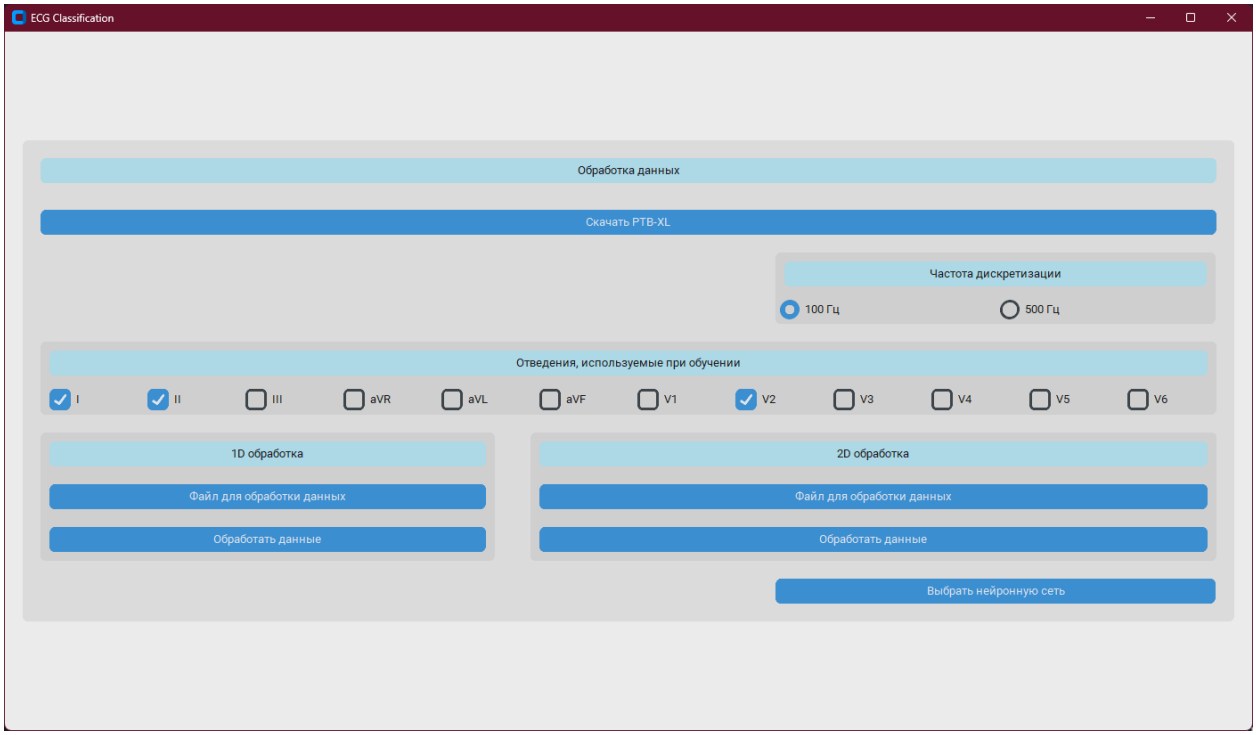

Рисунок 9. Страница "Обработка данных"

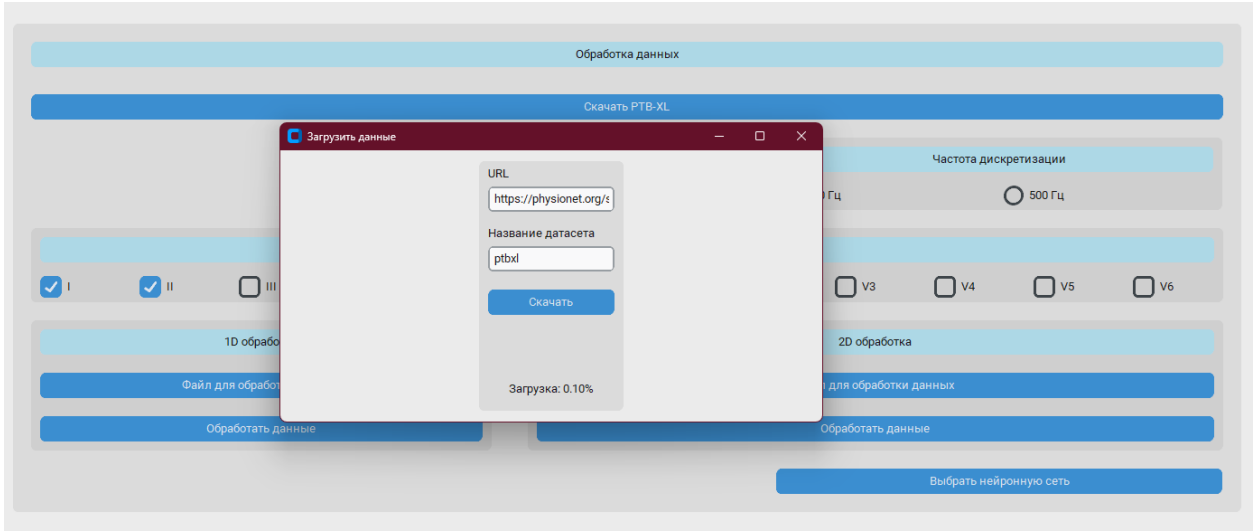

Рисунок 10. Диалоговое окно скачивания данных из интернета

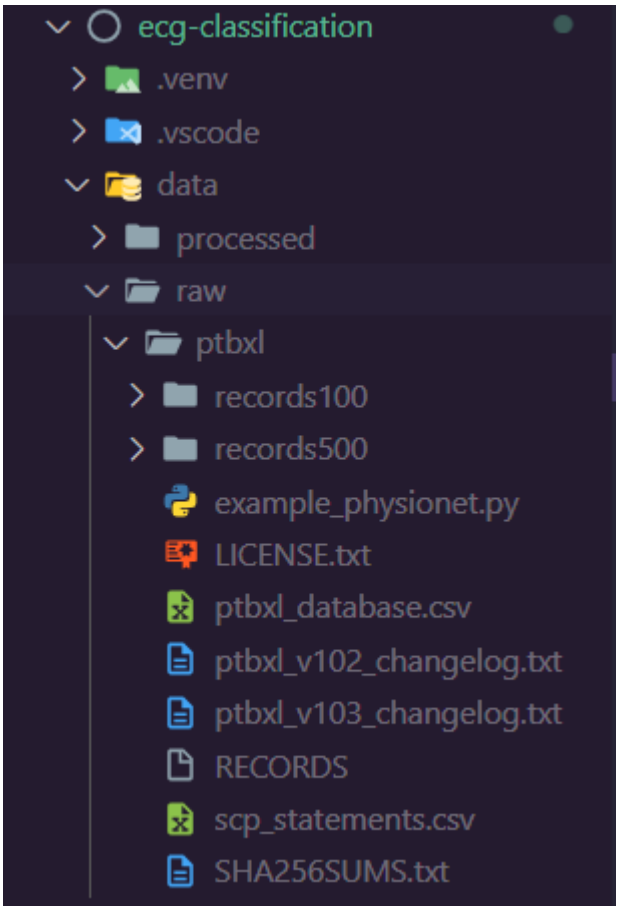

Рисунок 11. Данные скачаны

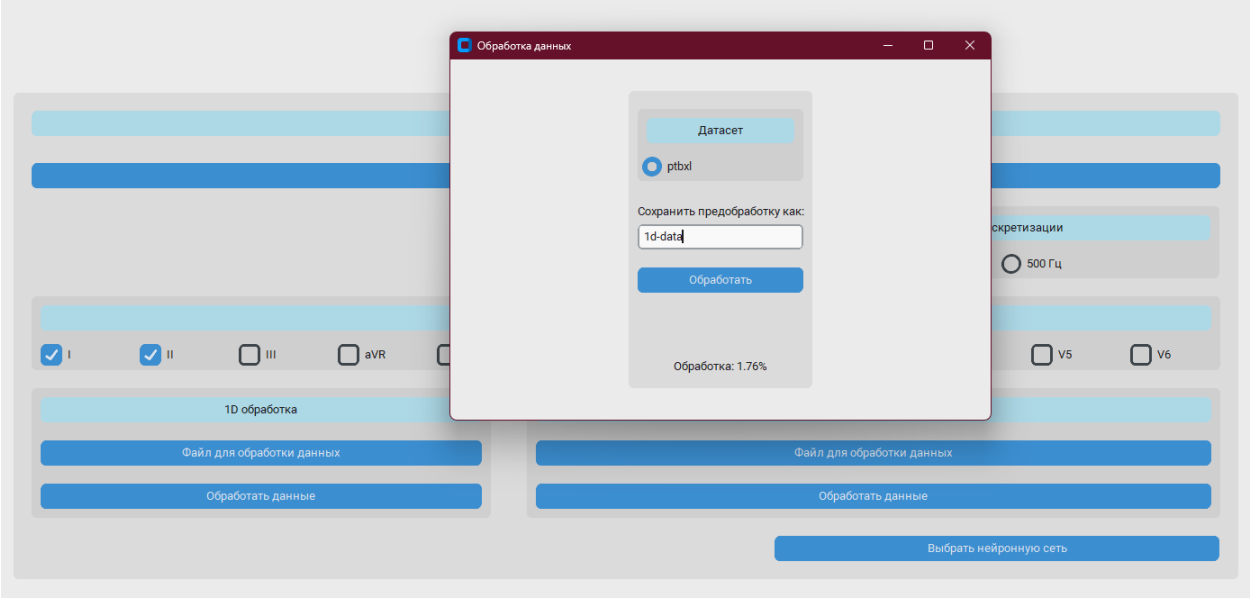

Рисунок 12. Диалоговое окно обработки данных

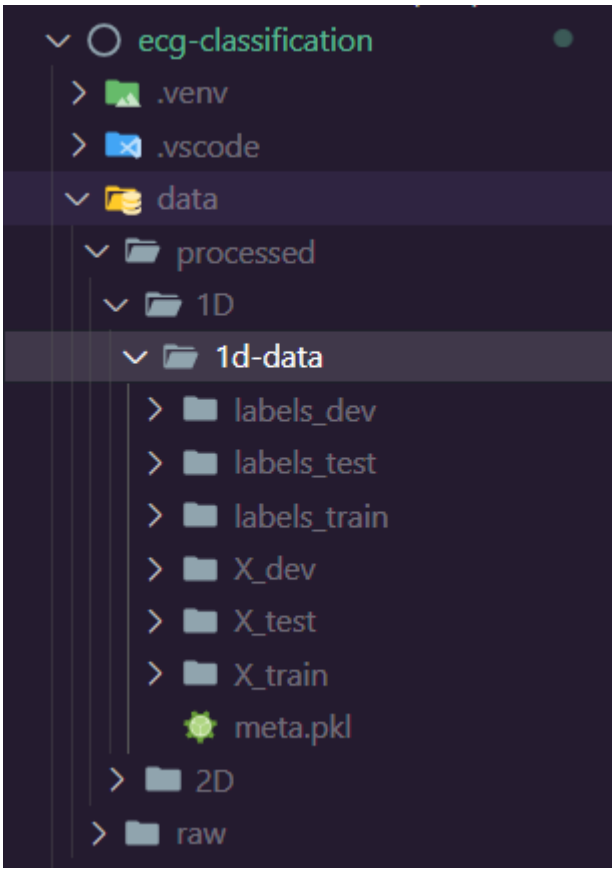

Рисунок 13. Данные обработаны

4.2. Страница "Нейронные сети"

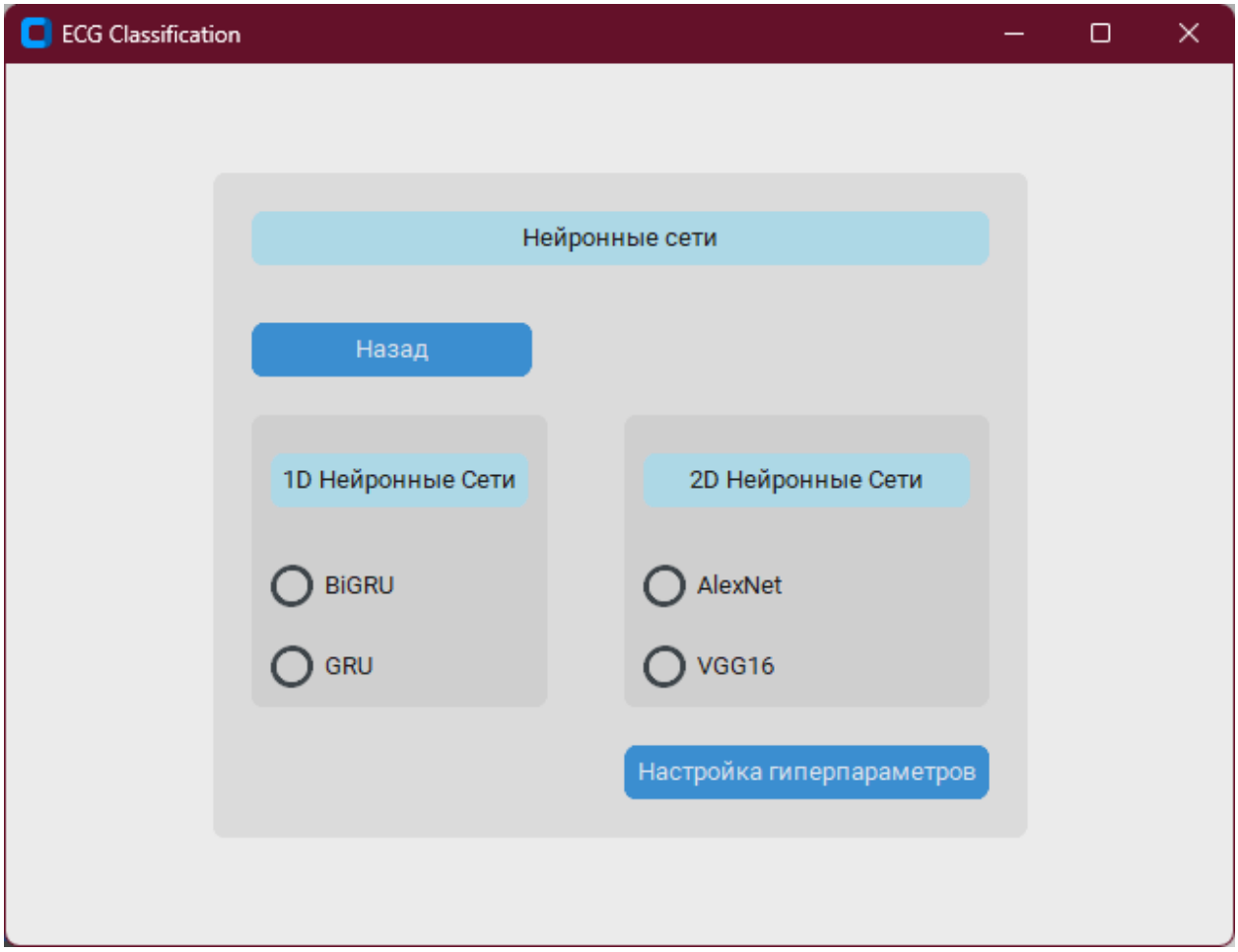

Рисунок 14. Страница "Нейронные сети"

4.3. Страница "Гиперпараметры"

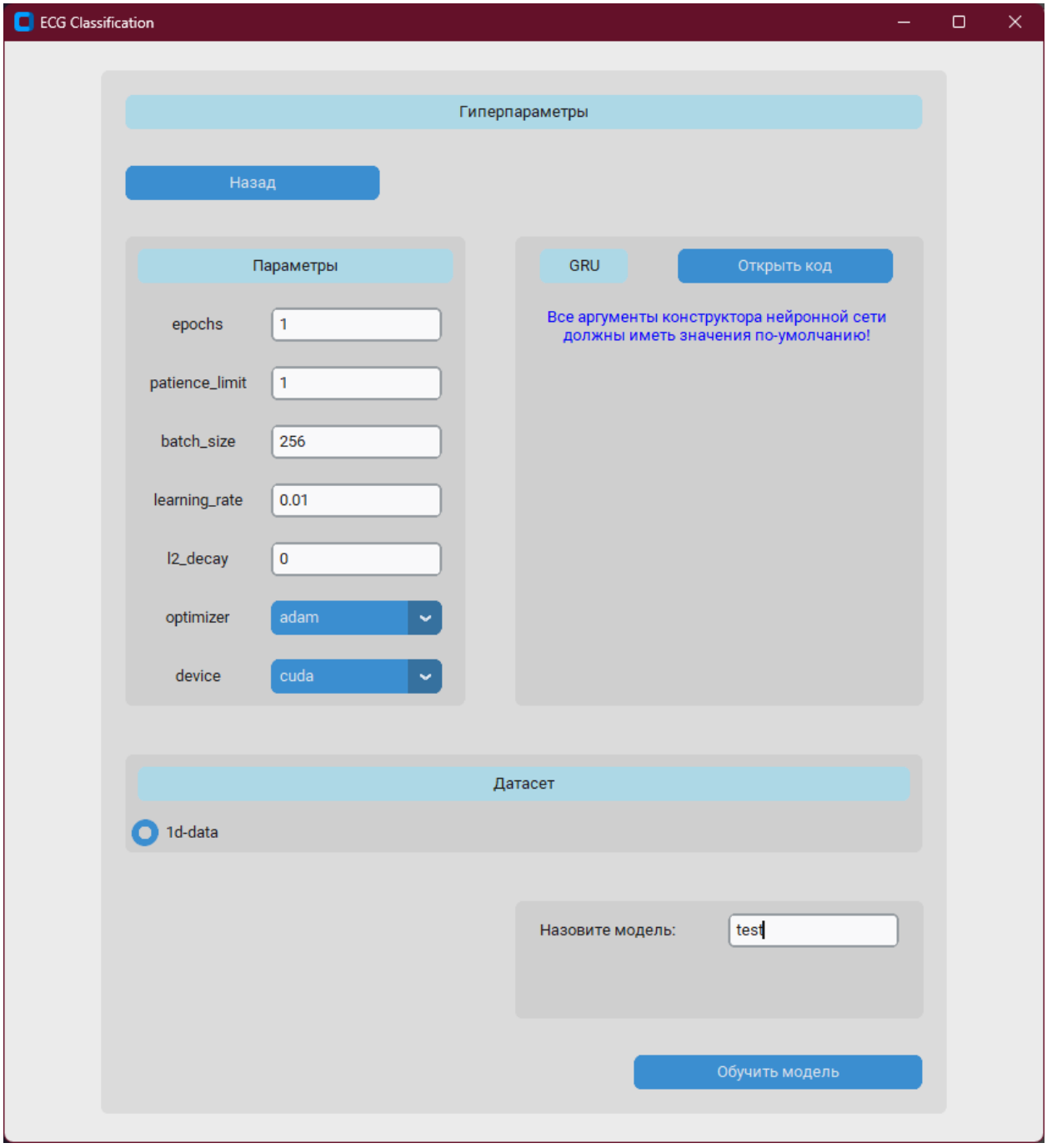

Рисунок 15. Страница "Гиперпараметры"

4.4. Страница "Обучение моделей"

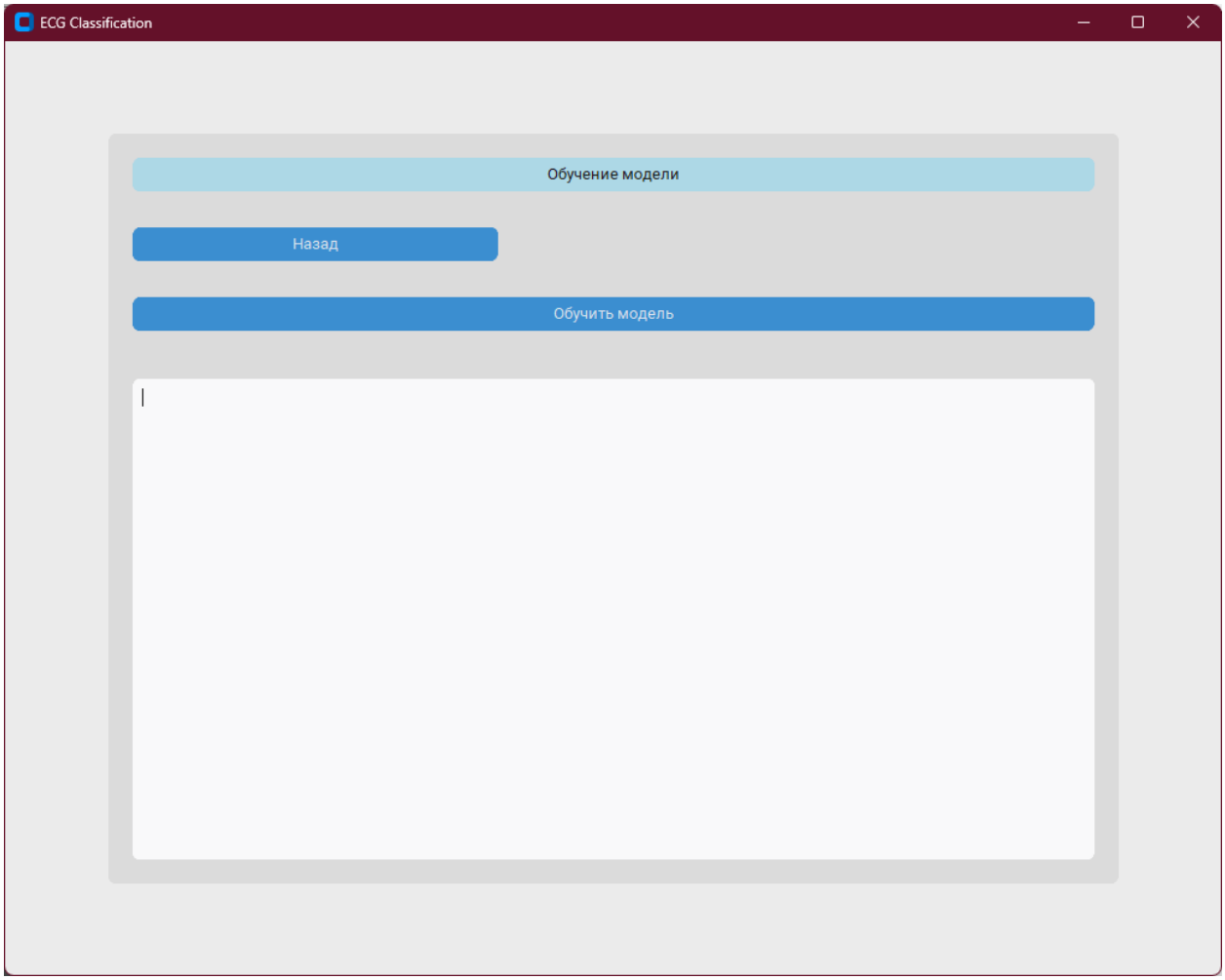

Рисунок 16. Страница "Обучение модели"

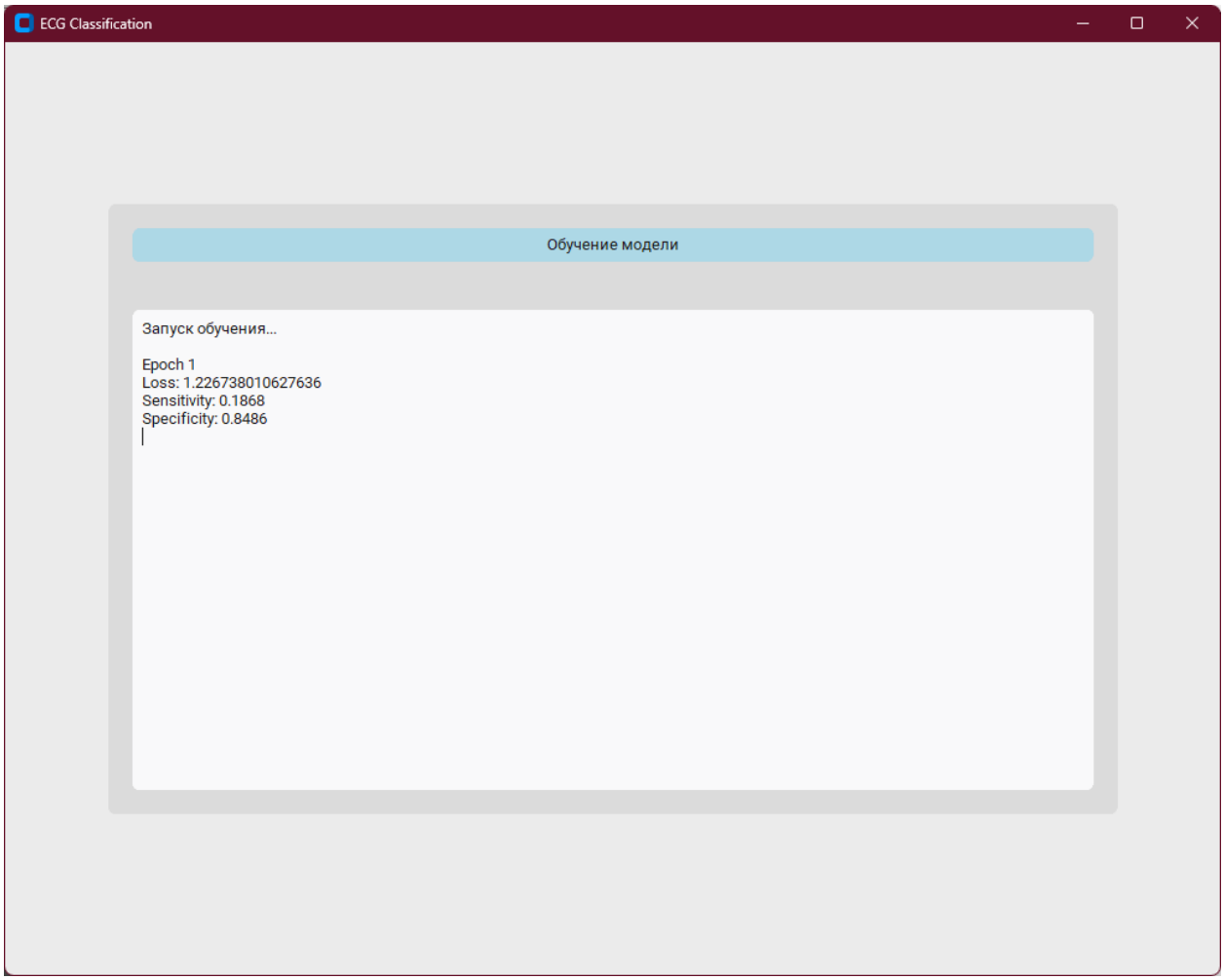

Рисунок 17. Процесс обучения

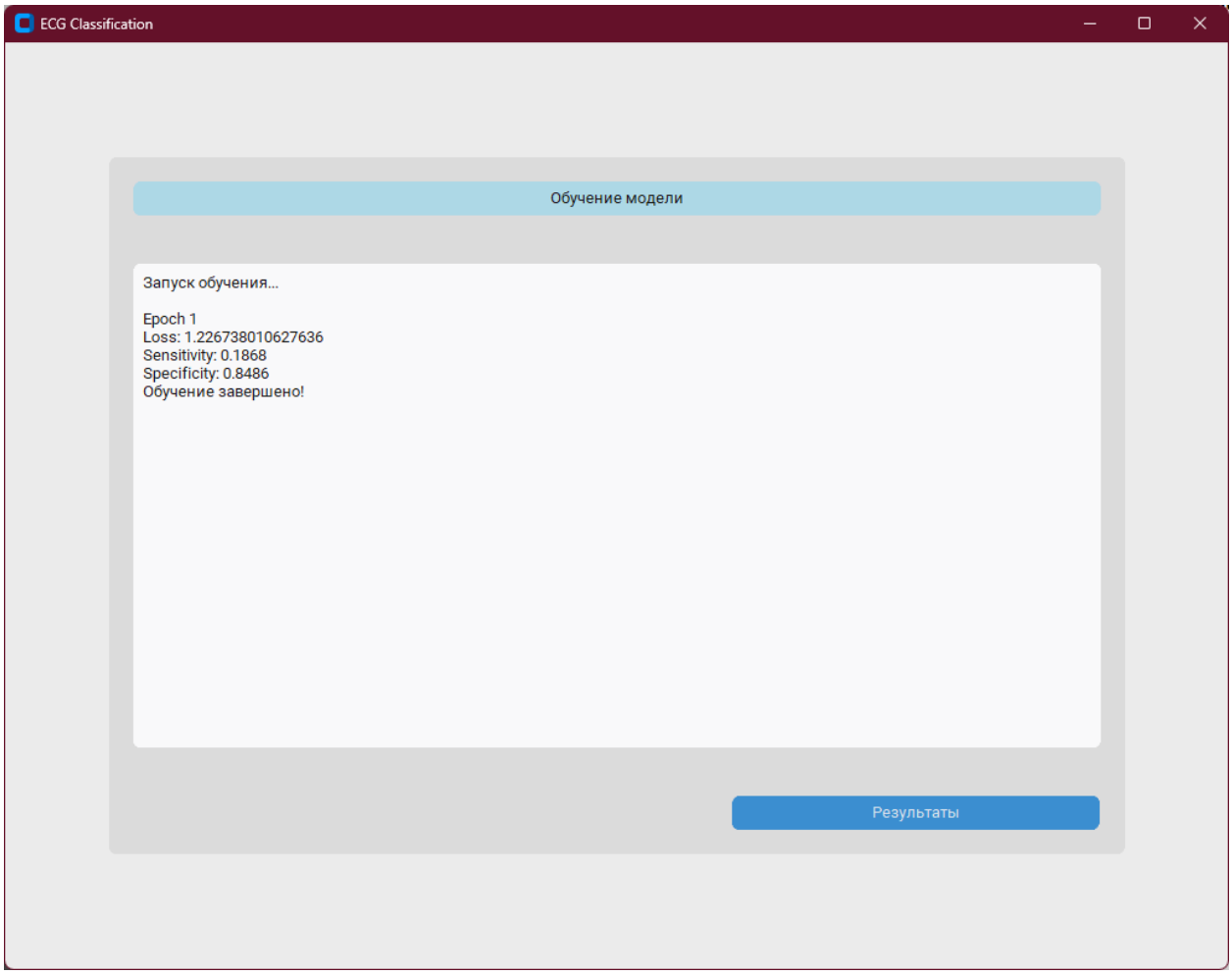

Рисунок 18. Завершение обучения

4.5. Страница "Результаты обучения"

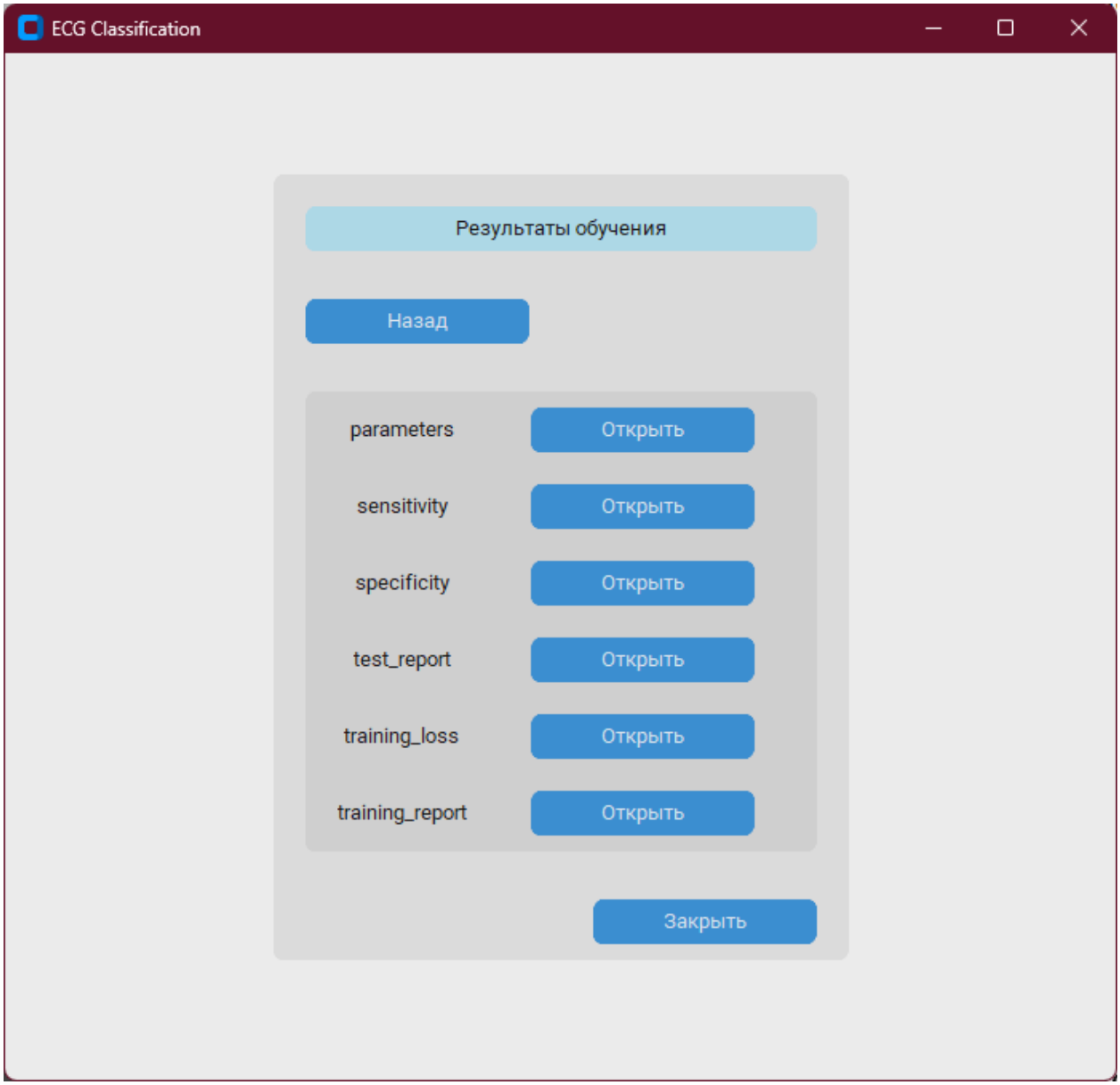

Рисунок 19. Страница "Результаты обучения"

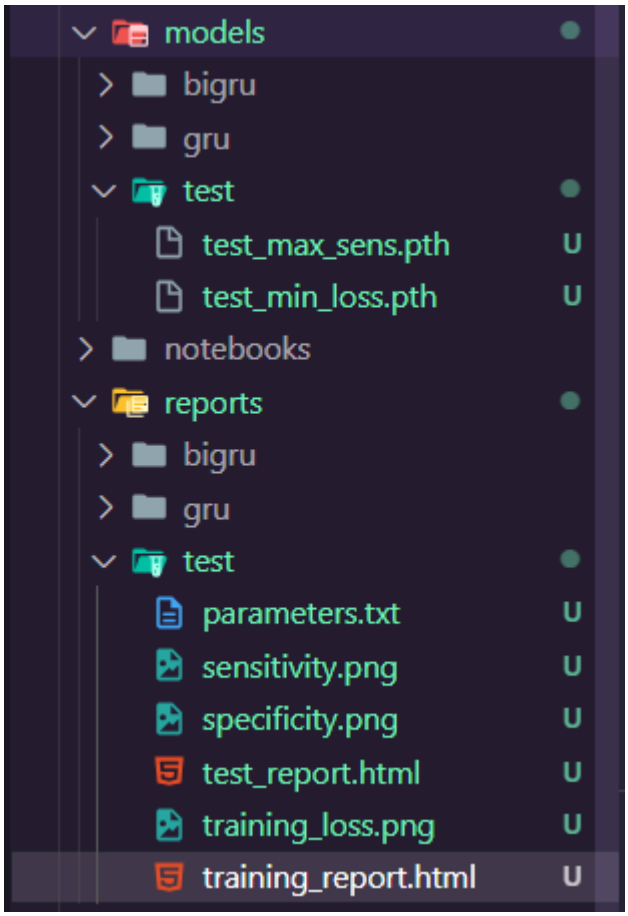

Рисунок 20. Результаты обучения сохранены

	TP	TN	FP	FN	Sensitivity	Specificity	G-mean
MI	2.2%	27.96%	1.08%	37.0%	1.45%	99.27%	12.02%
STTC	95.33%	11.0%	92.98%	11.88%	66.6%	38.4%	50.57%
CD	1.37%	28.37%	3.78%	33.52%	1.01%	97.53%	9.92%
HYP	1.1%	32.67%	2.16%	17.61%	1.53%	98.76%	12.28%
all	100.0%	100.0%	100.0%	100.0%	19.9%	84.04%	40.9%

Рисунок 21. Все отчеты можно открыть в редакторах или браузере

5. Данные

В целях обучения моделей в данной работе используется база данных PTB-XL, a large publicly available electrocardiography[4].

Исходные данные:

- название базы данных: PTB-XL, a large publicly available electrocardiography dataset;
- база данных специально подготовлена для машинного обучения;
- 21799 записей от 18869 пациентов;
- продолжительность записей - 10 секунд;
- 52% мужчин и 48% женщин;
- возраст от 0 до 95 (медиана 62, interquantile диапазон 22);
- 12 отведений (I, II, III, AVL, AVR, AVF, V1, V2, V3, V4, V5, V6) с электродами сравнения на правой руке;
- распределение диагнозов (диагнозы сгруппированы в суперклассы) (Таблица 1) (Рисунок 22):

Таблица 6. Распределение диагнозов

Количество записей	Класс	Описание
9514	NORM	Normal ECG
5469	MI	Myocardial Infarction
5235	STTC	ST/T Change
4898	CD	Conduction Disturbance
2649	HYP	Hypertrophy

[4] https://physionet.org/content/ptb-xl/1.0.3/

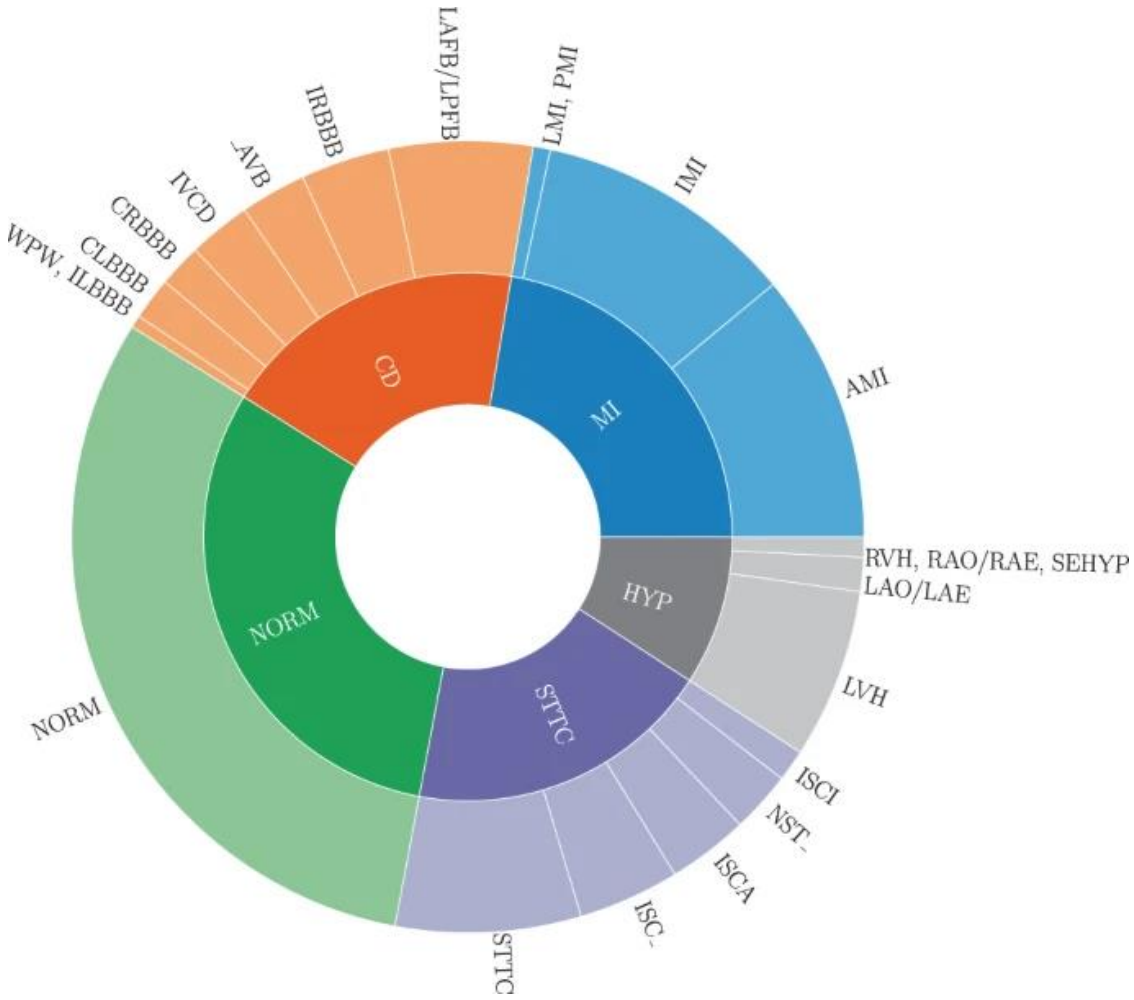

Рисунок 22. Графическая сводка набора данных PTB-XL с точки зрения диагностических суперклассов и подклассов, определение используемых аббревиатур см. в Таблице 7[5]

Таблица 7. Описание аббревиатур SCP-ECG для супер- и подклассов[6]

		Acronym	SCP statement Description
Superclasses		NORM	Normal ECG
		CD	Conduction Disturbance
		MI	Myocardial Infarction
		HYP	Hypertrophy
		STTC	ST/T change
Subclasses	NORM	NORM	Normal ECG
	CD	LAFB/LPFB	left anterior/left posterior fascicular block

[5] https://www.nature.com/articles/s41597-020-0495-6/figures/1
[6] https://www.nature.com/articles/s41597-020-0495-6/tables/6

		IRBBB	incomplete right bundle branch block
		ILBBB	incomplete left bundle branch block
		CLBBB	complete left bundle branch block
		CRBBB	complete right bundle branch block
		_AVB	AV block
		IVCB	non-specific intraventricular conduction disturbance (block)
		WPW	Wolff-Parkinson-White syndrome
	HYP	LVH	left ventricular hypertrophy
		RHV	right ventricular hypertrophy
		LAO/LAE	left atrial overload/enlargement
		RAO/RAE	right atrial overload/enlargement
		SEHYP	septal hypertrophy
	MI	AMI	anterior myocardial infarction
		IMI	inferior myocardial infarction
		LMI	lateral myocardial infarction
		PMI	posterior myocardial infarction
	STTC	ISCA	ischemic in anterior leads
		ISCI	ischemic in inferior leads
		ISC_	non-specific ischemic
		STTC	ST-T changes
		NST_	non-specific ST changes

Содержимое базы данных

• Каждая запись представлена в Waveform Database (WFBD) формате, которая состоит из двух (иногда трех) файлов:

– `.dat` - двоичный файл, содержащий образцы оцифрованных (дискретизированных) сигналов (samples of digitized signals);

– `.hea` - короткий текстовый файл, описывающий содержимое связанных файлов сигналов (.dat файлов);

– точность до 16 бит при разрешении 1 мкВ / LSB (16 bit precision at a resolution of 1μV/LSB);

– частота дискретизации 500 Гц (`records500/`) и те же данные с частотой дискретизации 100 Гц (`records100/`).

• Соответствующие метаданные хранятся в `ptbxl_database.cvs` с одной строкой на запись, идентифицируемую `ecg_id`. Файл содержит 28 столбцов, которые можно разделить на:

a. Идентификаторы (Identifiers):

– `ecg_id` - уникальный идентификатор записи;

– `patient_id` - идентификатор соответствующего пациента;

– `filename_hr` - путь к исходной записи (500 Гц);

– `filename_lr` - путь к записи с пониженной частотой дискретизации (100 Гц).

b. Общие метаданные (General Metadata): демографические и регистрационные (recording) метаданные

– возраст (age);

– пол (sex);

– рост (height);

– вес (weight);

– медсестры (nurse);

– сайт (site);

– устройство (device);

– дата записи (recording_date).

c. Отчеты ЭКГ (Ecg statements):

– `scp_code` - SCP-ECG (стандарт такой есть) отчеты (т.е. диагнозы) в виде словаря с записями вида *statement: likelihood*, где вероятность установлена равной 0, если неизвестно (SCP-ECG statements as a

dictionary with entries of the form statement: likelihood, where likelihood is set to 0 if unknown);

– `report` - отчет в виде строки;
– `heart_axis`;
– `infarction_stadium1`;
– `infarction_stadium2`;
– `validated_by`;
– `second_opinion`;
– `initial_autogenerated_report`;
– `validated_by_human`.

d. Метаданные сигнала (Signal Metadata):

– `static_noise` - статический шум сигнала;
– `burst_noise` - импульсный шум сигнала;
– `baseline_drift` - смещение (дрейф) нулевой (базовой) линии;
– `electrodes_problems`;
– `extra_beats` - для подсчета дополнительных систол и кардиостимулятора образцы сигналов, указывающих на активный кардиостимулятор.

e. Крос-валидационные сгибы (Cross-validation Folds):

– `start_fold` - рекомендуемые 10-кратные разбиения при обучающем тестировании (`strat_fold`), полученные с помощью стратифицированной выборки с учетом распределения пациентов, т. е. Все записи конкретного пациента были отнесены к одному и тому же сгибу. Записи в fold 9 и 10 прошли по крайней мере одну оценку человеком и, следовательно, имеют особенно высокое качество маркировки. Поэтому мы предлагаем использовать fold 1-8 в качестве обучающего набора, fold 9 в качестве проверочного набора и fold 10 в качестве тестового набора.

- Вся информация, относящаяся к используемой схеме аннотаций, хранится в специальном `scp_statements.csv`:
  - категория (category), к которой может быть отнесен каждый отчет (diagnostic, form and/or rhythm);
  - иерархическая организация для диагностических заключений делиться на `diagnostic_class` и `diagnostic_subclass`.

6. Предобработка данных

Обработка данных происходит согласно статье [13]. Одномерные NN (в данной работе это RNN) используют исходное одномерное представление данных. Двумерные NN (CNN) требуют двумерного представления, т.е. необходимо исходный 1D сигнал преобразовать в 2D. Меткам нужна дать числовые значения. Код представлен в Приложении Б.

6.1. 1D обработка

1D обработка заключается в следующем:

1) мы будем использовать данные с частотой дискретизации 100 Гц, поскольку частота 90 Гц была признано минимальной для выявления форм волны в сигнале (P, QRS и T) [18];

2) будем использовать только сигналы трех отведений: I, II, V2. Использование всех 12 отведений потребовало бы больших вычислительных ресурсов. Исходя из [19] по данным этих трех отведений были успешно реконструированы 12 отведений. Следовательно, они содержат большую часть релевантной информации из всех каналов;

3) применим полосовой фильтр Баттерворта второго порядка с частотой среза верхних частот 1 Гц для подавления блуждания базовой линии и частотой среза низких частот 45 Гц для устранения высокочастотных шумов [20]-[22];

4) z-нормализация, чтобы уменьшить влияние выбросов.

Для обучение одномерных нейронных сетей этого достаточно. Но для двумерных сетей необходимо преобразовать одномерный сигнал в двумерный.

6.2. 2D обработка

2D обработка заключается в преобразовании 1D представления в изображение, как это было сделано в [23]. Чтобы сохранить корреляционную зависимость между сэмплами, применяются Gramian Angular Field (GAF), Recurrence Plot (RP) и Markov Transition Field (MTF). Автор статьи [23], что эти

преобразования лучше, по сравнению с другими преобразованиями, такими как короткопериодное преобразование Фурье или вейвлет-преобразование.

Gramian Angular Field

GAF – это изображение, полученное из временного ряда, представляющее собой некую временную корреляцию между каждой парой значений временного ряда. Представляет изображение в угловой системе, взамен простой прямоугольной. Пусть E - сигнал ЭКГ из n значений (samples), такое, что $E = \{s_1, s_2, \dots, s_n\}$. Нормализовав E получим $\overline{E}$. Нормализованную ЭКГ можно отобразить на угловую систему следующим образом: значения сигнала преобразуем в угловые значения (angular cos) через $arccos$, а временную метку - в радиус.

$$\left.\begin{array}{c} \beta = arccos(s_{i0}) \\ R = \dfrac{t_i}{C} \end{array}\right\}, \tag{1}$$

где s_{i0} - нормализованное значение сигнала; t_i - временная метка для s_{i0}; $i = \{1,2,\dots,n\}$; C - константа, регулирующая радиус.

Чтобы наилучшим образом показать корреляцию между двумя значениями разных временных меток, используются два метода: суммирования и разности. В данной работе используется метод суммирования, который имеет вид:

$$\begin{aligned} Grammian\,field = &\quad cos\left(\beta_i + \beta_j\right); \\ Grammian\,field = &\quad \overline{E}^T . \overline{E} - \sqrt{I - \overline{E}^{T^2}} . \sqrt{I - \overline{E}^2}, \end{aligned} \tag{2}$$

где $i, j = \{1,2,\dots,n\}$, I - единичный вектор (unit row vector).

Recurrence Plot

RP – это изображение, представляющее расстояние между траекториями (временными точками), извлеченными из исходного временного ряда

$$\text{R-plot} = \alpha\left(\lambda - \left|\left|s_i - s_j\right|\right|\right), \tag{3}$$

где λ - предельное значение; $i, j = \{1,2,\dots,n\}$, α - функция Хевисайда, которая имеет вид

$$\alpha(x) = \begin{cases} 0, & x < 0; \\ 1, & x \geq 0. \end{cases} \quad (4)$$

Markov Transition Field

MTF – это изображение, полученное из временного ряда, представляющее собой поле вероятностей перехода для дискретизированного временного ряда. Он показывает, насколько связаны между собой две произвольные значения временного ряда, относительно того, как часто они появляются рядом друг с другом во временном ряду.

Первый шаг преобразования - разложение каждого значения временного ряда по квантилям (или бинам). Например, если у нас есть временной ряд с непрерывными значениями от 0 до 1, и мы задаем квантиль, равный 10, то значения ряда будут распределены по 10 интервалам (например, 0-0.1, 0.1-0.2, …, 0.9-1.0).

Далее формируется матрица переходов состояний:

$$A_{ij} = P(s_t = j | s_{t-1} = i). \quad (7)$$

Т.е. A_{ij} - это вероятность перехода из состояния i в состояние j. Оцениваем это значение методом максимального правдоподобия: A_{ij} равно отношению количества переходов из i в j на общее число раз, когда мы находились в состоянии i.

Заметим, что если у нас Q квантилей, то A - это матрица $Q \times Q$.

Эта матрица учитывает только вероятности переходов между бинами, независимо от времени, и это приводит к потере информации о пространственных характеристиках сигнала. Для устранения этого недостатка матрицу A преобразуют в марковское поле переходов M:

$$M_{kl} = A_{q_k q_l}, \quad (8)$$

где q_k - квантиль для x_k; q_l - квантиль для x_l. Индексы при A относятся к состояниями, а индексы при M - к временным меткам.

A_{ij} - вероятность перехода из состояния i в состояние j.

M_{kl} - вероятность одноступенчатого (one-step) перехода из бина для x_k в бин для x_l. Т.е. x_k и x_l рассматриваются как две точки в временном ряде в произвольные моменты времени k и l.

q_k и q_l - соответствующие квантили.

M_{kl} - это вероятность прямого одноступенчатого (direct one-step) (т.е. марковского) перехода от q_k к q_l во временном ряду [24].

1) во-первых, используем все те же три отведения I, II, V2 с частотой дискретизации 100 Гц и тот же полосовой фильтр Баттерворта, что и при 1D обработке (пункты 1-3 1D обработки);
2) нормализуем сигнал, так, чтобы все значения были в интервале от [0:1];
3) для каждого из трех отведений мы создаем по три изображения размерностью (1000, 1000) (итого 3x3). Объединив их получим одно изображения размерностью (9, 1000, 1000).
4) вышеупомянутые три изображения получаем, используя GAF, RP и MTF, согласно статье [23];
5) уменьшения размера изображения методом интерполяции до размеров 224×224.

Результат преобразований показан на Рисунке 23 (без обрезания размера изображения):

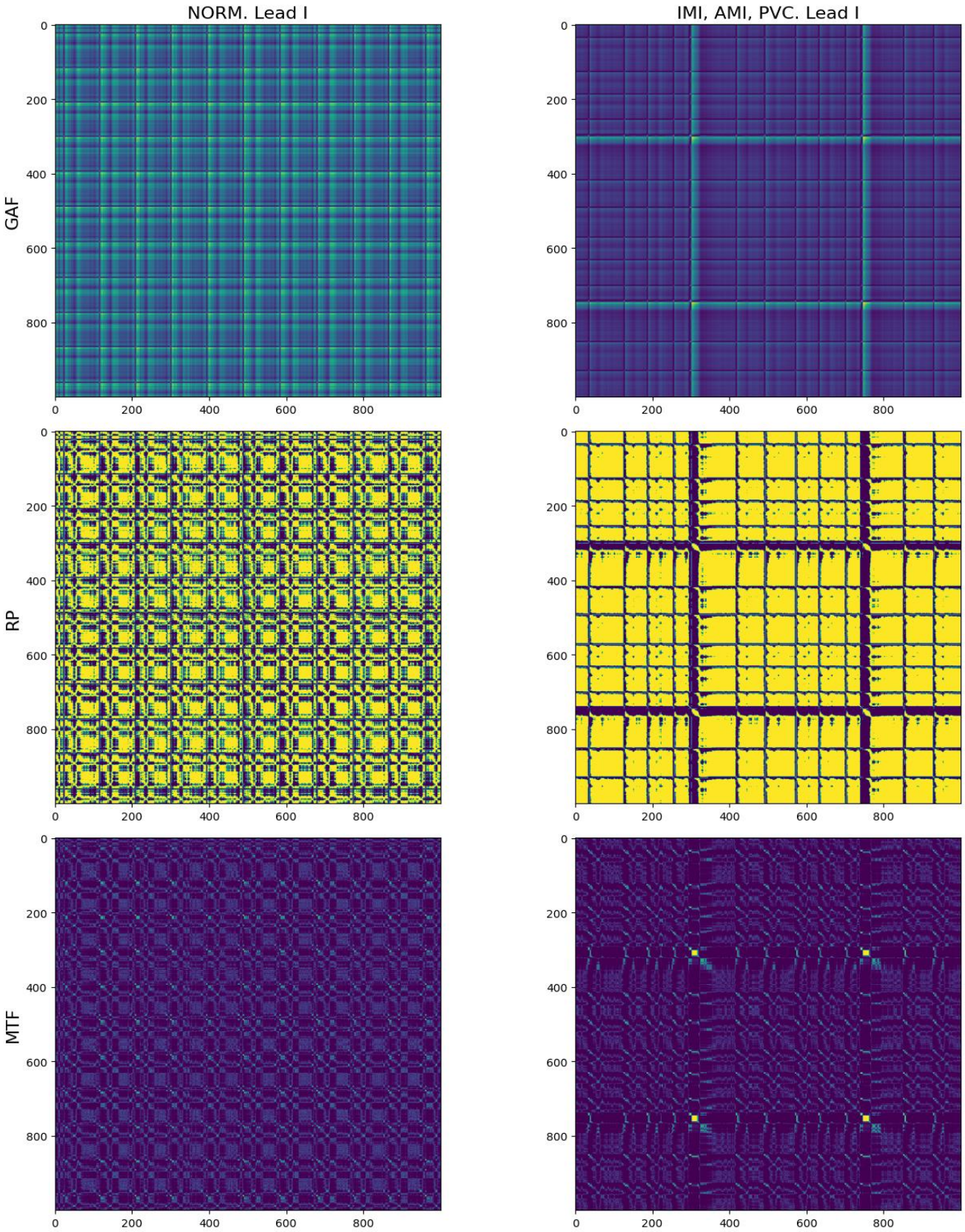

Рисунок 23. GAF, RP и MTF изображения для сигнала в I отведении образца с диагнозом NORM и образца с диагнозами IMI и AMI.

6.3. Метки

Метки создаются по следующему принципу:

1) добавим к каждой метке информацию о том, к какому классу (поле `diagnostic_class` в `scp_statements.csv`) относится диагноз (напр., диагнозы *NDT, NST_, DIG, LNGQT* относятся к классу *STTC*). По этим классам и будет происходить классификация;

2) представим метку (меткой является `diagnostic_class`) как бинарный вектор из 4 элементов. Комбинации нулей и единиц в этом векторе будет соответствовать класс(-у) диагноза(-ов). Таблица соответствия (Таблица 8).

Таблица 8. Таблица соответствия диагнозов и индексов вектора метки

index / diagnostic class	0	1	2	3
NORM	0	0	0	0
MI	1	0	0	0
STTC	0	1	0	0
CD	0	0	1	0
HYP	0	0	0	1

Пример комбинаций классов (Таблица 9):

Таблица 9. Пример комбинации классов диагнозов

index / diagnostic class	0	1	2	3
MI & HYP	1	0	0	1
STTC & CD & HYP	0	1	1	1

6.4. Разбиение на обучающую, валидационную и тестовую наборы

Разделим записи на обучающий, валидационный и тестовый наборы согласно предложенному самим поставщиком данных разбиению [16] (см. также [17]). Каждая запись относится к 1 из 10 складок (fold) (поле `strat_fold` в файле `ptbxl-database.csv`). Рекомендуется записи, относящиеся к 1-8 складкам, использовать в качестве обучающего набора; записи, относящиеся к 9-ой складке - в качестве валидационного набора; записи, относящиеся к 10-ой складке - в качестве тестового.

7. Архитектуры нейронных сетей

Реализации нейронных сетей представлена в Приложении В.

7.1. Рекуррентные нейронные сети

Рекуррентные нейронные сети (РНС, англ. Recurrent neural network, RNN) — это семейство нейронных сетей для обработки последовательных данных.

Простейшая сеть RNN с одним нейроном выглядит следующим образом (Рисунок 24):

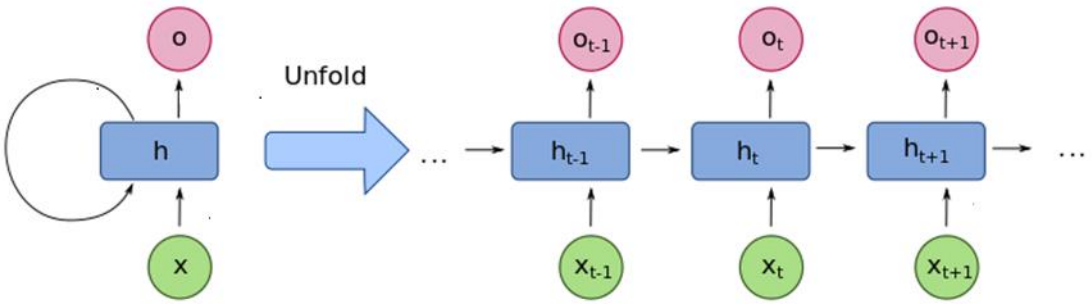

Рисунок 24. Рекуррентный нейрон (слева), развернутый во времени (справа)[7]

Каждый рекуррентный нейрон имеет два набора весов: один для входов x_t и один для выходов предыдущего временного шага o_t: w_x и w_o. Если говорить о слое рекуррентных нейронов, тогда все весовые векторы можно поместить в матрицы W_x и W_0.

Выходной вектор целого рекуррентного слоя можно вычислить следующим образом:

$$o_t = \phi(W_x^T x_t + W_o^T o_{t-1} + b), \quad (9)$$

где $b -$ вектор смещения, $\phi(\cdot) -$ функция активации.

Вычисление выхода рекуррентного слоя сразу для целого мини-пакета, помещая все входы на временном шаге t во входную матрицу

[7] https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg

На первом временном шаге, при $t = 0$, предполагается, что все предшествующие выходы нулевые [25] - [26].

Ячейка памяти – часть нейронной сети, которая сохраняет состояние через временные шаги. Ячейка памяти на временном шаге *t*, обозначается как h_t ("h" означает "hidden" – "скрытое"), является функцией от некоторых входов на данном временном шаге и ее состояния на предыдущем временном шаге: $h_t = f(h_{t-1}, x_t)$. Выход на временном шаге *t*, обозначаемый o_t, также представляет собой функцию от предыдущего состояния и текущих входов. Для базовых ячеек, которые обсуждались до сих пор, выход просто равен состоянию, но в более сложных ячейках это не всегда так. Это означает, что скрытое состояние ячейки и ее выход могут отличаться (Рисунок 24).

Возможно построение двунаправленной рекуррентной сети. При такой конструкции два скрытых слоя RNN которые читают входные данные одна в прямом направлении, другая в обратном. Выходом на каждом временном шаге будет конкатенация выходов этих двух слоев. Такой подход полезен, если важно учитывать информация из прошлого и будущего одновременно.

Обучение рекуррентных сетей происходит методом обратного распространения ошибки (backpropagation). Веса регулируются с целью минимизации функции потерь между прогнозами модели и правильными метками. Алгоритм обратного распространения ошибки (backpropagation) вычисляет градиенты функции потерь относительно каждого веса и обновляет их с использованием метода оптимизации, такого как Adam или другие.

7.2. Gated Recurrent Unit

Из-за трансформаций, которым подвергаются данные при проходе через сеть RNN, на каждом временном шаге определенная информация утрачивается. Через некоторое время состояние сети RNN практически не содержит следов первых входов. Чтобы справиться с проблемой, были

предложены разнообразные типы ячеек с долговременной памятью. Одна из них ячейка управляемого рекуррентного блока (Gated Recurrent Unit - GRU).

GRU – упрощенная версия LSTM, которая не рассматривается в данном проекте, так как исходя из [13], GRU имеет лучшие показатели качества. Архитектура GRU показана на Рисунке 11.

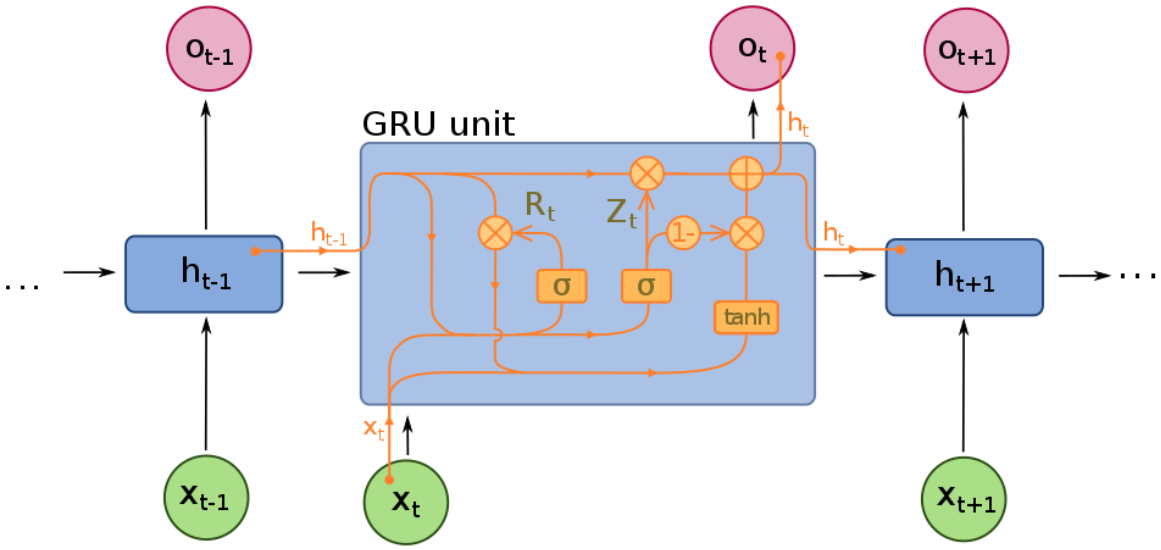

Рисунок 25. Ячейка GRU[8]

Обозначения на Рисунке 25:

- $\otimes$ – шлюз. Данные, проходя через шлюз, поэлементно умножаются на 0 или 1. Если на 0 – данные забываются, на 1 - сохраняются;
- Z_t – контроллер шлюза забывания и входного шлюза. Если Z_t выдает 1, то в шлюзе забывания будет 1, а во входном шлюзе 0 (см. элемент (1-) на Рисунке 25). Если Z_t выдает 0, то в шлюзе забывания будет 0, а во входном шлюзе 1;
- R_t – контроллер шлюза. Управляет тем, какие части h_{t-1} будут показаны в главном слое $tanh$;
- $tanh$ – главный слой. Базовая ячейка RNN;
- σ – сигмоидальная функция активация;
- $\oplus$ – сложение;

[8] https://en.wikipedia.org/wiki/Recurrent_neural_network#/media/File:Gated_Recurrent_Unit.svg

Когда предыдущее состояние h_{t-1} поступает в ячейку GRU, оно проходит через шлюз R_t, и попадает в слой $tanh$, вместе с входными данными x_t. Результат $tanh$ передается во входной шлюз и складываются с h_{t-1}, который прошел через шлюз забывания Z_t. Полученный результат будет состоянием ячейки на данном временном шаге (h_t) и ее выходом (o_t).

Благодаря такой конструкции ячейки сеть способна сохранять более важную информацию и забывать менее важную.

7.3. Сверточные нейронные сети

Сверточные нейронные сети (Convolutional Neural Network, CNN) — это тип искусственных нейронных сетей, используемых для анализа данных с топологической структурой, таких как изображения. CNN применяются для распознавания и классификации образов, объектов и паттернов в данных.

Работа сверточной нейронной сети обычно интерпретируется как переход от конкретных особенностей изображения к более абстрактным деталям, и далее к ещё более абстрактным деталям вплоть до выделения понятий высокого уровня. При этом сеть самонастраивается и вырабатывает сама необходимую иерархию абстрактных признаков (последовательности карт признаков), фильтруя маловажные детали и выделяя существенное.

СНС состоят из различных типов слоев, каждый из которых выполняет специфические функции для анализа входных данных. Основные слои СНС включают слои:

- свёртки,
- пулинга,
- полносвязные слои.

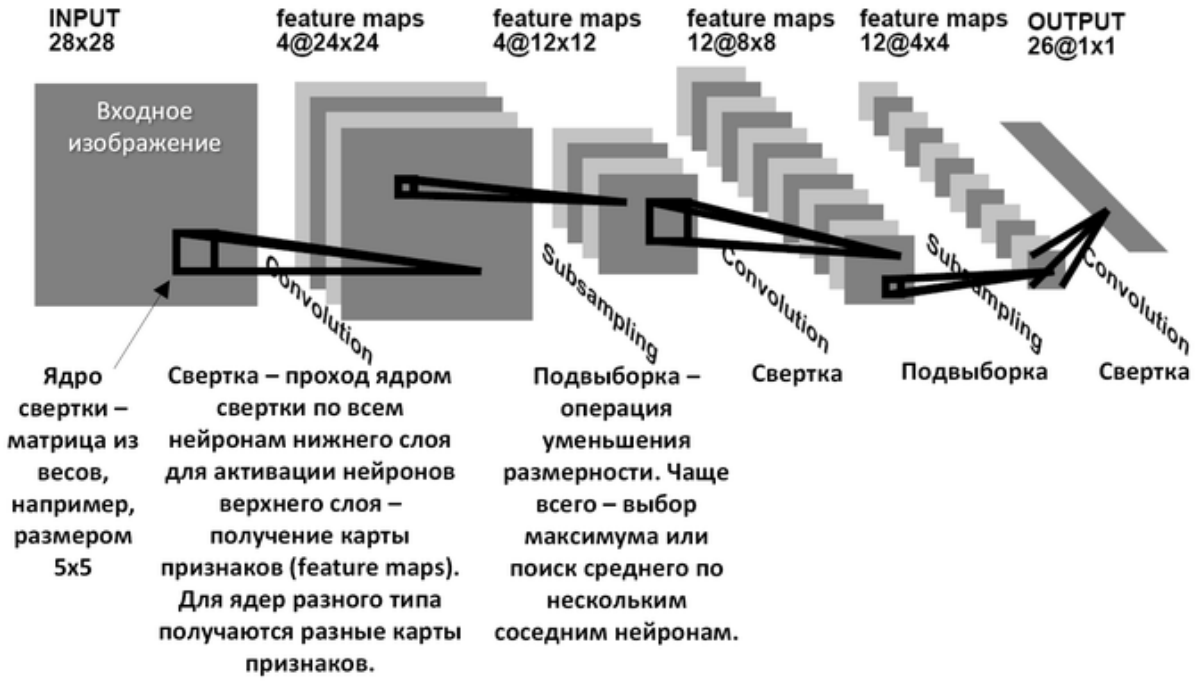

Рисунок 26. Архитектура CNN[9]

1) Слой свёртки (convolutional layer) является основным строительным блоком СНС. Этот слой проходит по всему изображению. Каждый нейрон этого слоя принимает во внимание не все выходные данные предыдущего слоя, а прямоугольную область (например, 5x5), в биологии называемую рецепторным полем. Смещение от одной прямоугольной области к другой называется страйдом. Веса нейронов могут быть представлены как небольшие изображения размером рецепторного поля. Такие наборы весов называются фильтрами или ядрами (Рисунок 26). Слой с нейронами, использующими один и тот же фильтр, выдает карту признаков, выделяющую области изображения, которые больше всего активируют фильтр. Фильтры можно определять вручную, но на практике это делается алгоритмом обучения. Подводя итог, слой свертки выдает множества карт признаков на число ядер свертки.

[9] https://commons.wikimedia.org/wiki/File:%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%81%D0%B2%D0%B5%D1%80%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D0%B9_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B5%D1%82%D0%B8.png

2) Слой пулинга или субдискретизирующий слой (pooling или subsampling) сжимает входное изображения для сокращения вычислительных нагрузок и количества параметров (тем самым уменьшая риск переобучения). Как и в слое свертки, каждый нейрон пулинга связан с выходами ограниченного числа нейронов из предыдущего слоя, которые расположены внутри небольшого прямоугольного рецепторного поля. Однако нейрон пулинга не имеет весов, он лишь агрегирует входы с применением функции агрегирования, такой как максимум или среднее.

3) Полносвязный слой. После нескольких этапов свёртки и пулинга изображение преобразуется от сетки с высоким разрешением к более абстрактным картам признаков. На каждом следующем уровне количество каналов увеличивается, а размер изображения в каждом канале уменьшается. В итоге получается большой набор каналов, каждый из которых содержит небольшое количество данных (иногда даже один параметр), которые представляют самые абстрактные характеристики исходного изображения. Эти данные затем объединяются и передаются в обычную полносвязную нейронную сеть, которая может состоять из нескольких слоёв. В полносвязных слоях уже нет пространственной структуры пикселей, и размерность этих слоёв относительно мала по сравнению с количеством пикселей исходного изображения.

Обучение CNN происходит также методом обратного распространения ошибки (backpropagation). Веса регулируются с целью минимизации функции потерь между прогнозами модели и правильными метками. Алгоритм обратного распространения ошибки (backpropagation) вычисляет градиенты функции потерь относительно каждого веса и обновляет их с использованием метода оптимизации, такого как Adam или другие.

7.4. AlexNet

AlexNet — это глубокая сверточная нейронная сеть, разработанная Алексеем Крижевским, Ильей Сутскевером и Джеффри Хинтоном. Эта модель была представлена в 2012 году и завоевала первое место на конкурсе ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012, значительно улучшив точность классификации изображений по сравнению с предыдущими методами.

AlexNet имеет пять сверточных и три полносвязных слоев. После каждого из них применяется функция активации (кроме последнего полносвязного). Дропоут в 0.5 применяется между первым и вторым полносвязными слоями в целях уменьшения риска переобучения. Размерность выхода последнего слоя равна количеству классов (если не учитывать размер мини-пакета), т.е. 4 в данном случае (не 5, так как отсутствие всех меток – это 5-ый класс) [25], [26]. Её архитектура представлена на (Рисунок 27).

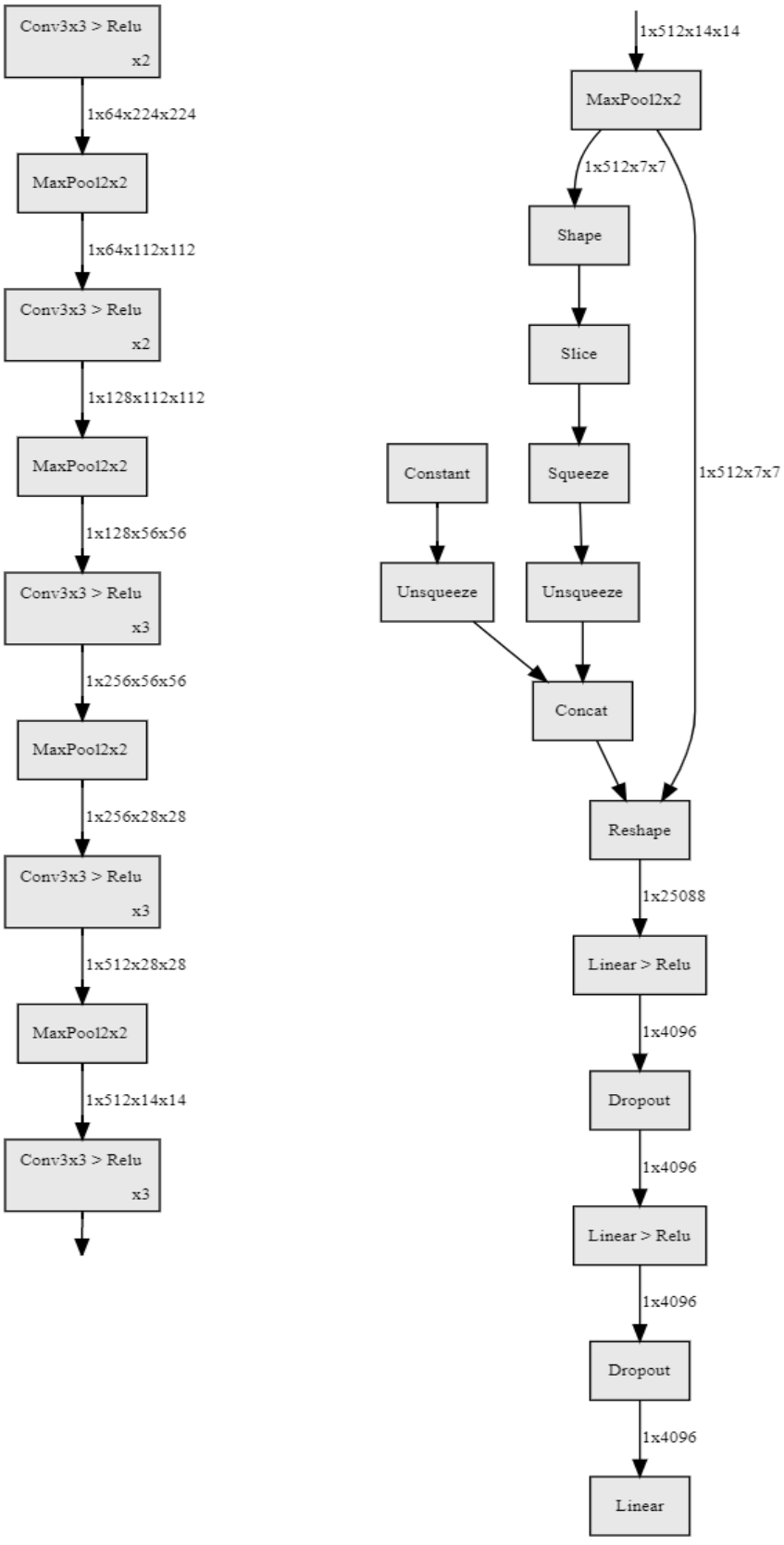

Рисунок 27. Архитектура сети AlexNet

8. Описание параметров сетей

8.1. Функция потерь

Как мы уже говорили ранее (см. раздел 6.3. Метки), наши метка представлена в виде четырехэлементного бинарного массива, где 1 – означает принадлежность классу, а 0 – обратное. У нас задача многозначной классификации. Поэтому мы будем применять функцию потерь: бинарную кросс-энтропию (Binary Cross-Entropy, BCE)

BCE — это функция потерь, которая используется в задачах бинарной классификации. Она измеряет расхождение между предсказанными вероятностями и истинными бинарными метками (0 или 1). Эта функция потерь оценивает, насколько хорошо модель предсказывает вероятность принадлежности к одному из двух классов.

В PyTorch.nn имеется функция потерь BCEWithLogitsLoss[], которая перед тем, как произвести вычисление BCE, применяет ко всем входным элементом (логиты)

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ \dots & \dots & \dots & \dots \\ x_{b1} & x_{b2} & x_{b3} & x_{b4} \end{bmatrix}, \text{где } b - \text{размер батча;}$$

сигмоидную функцию $\sigma(x)$

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}, \tag{10}$$

чтобы преобразовать значения в вероятности:

$$\hat{y} = \sigma(x) = \begin{bmatrix} \sigma(x_{11}) & \sigma(x_{12}) & \sigma(x_{13}) & \sigma(x_{14}) \\ \sigma(x_{21}) & \sigma(x_{22}) & \sigma(x_{23}) & \sigma(x_{24}) \\ \dots & \dots & \dots & \dots \\ \sigma(x_{b1}) & \sigma(x_{b2}) & \sigma(x_{b3}) & \sigma(x_{b4}) \end{bmatrix}.$$

Затем вычисляется бинарная кросс-энтропия для каждой метки. Обозначим истинные метки как y:

$$y = \begin{bmatrix} y_{11} & y_{12} & y_{13} & y_{14} \\ y_{21} & y_{22} & y_{23} & y_{24} \\ \dots & \dots & \dots & \dots \\ y_{b1} & y_{b2} & y_{b3} & y_{b4} \end{bmatrix}.$$

BCE вычисляется по формуле:

$$L(y, \hat{y}) = -(y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})). \quad (11)$$

Применив эту функцию ко всем элементам матриц y и $\hat{y}$ получаем матрицу потерь:

$$L = \begin{bmatrix} L(y_{11}, \hat{y}_{11}) & L(y_{12}, \hat{y}_{12}) & L(y_{13}, \hat{y}_{13}) & L(y_{14}, \hat{y}_{14}) \\ L(y_{21}, \hat{y}_{21}) & (y_{22}, \hat{y}_{22}) & L(y_{23}, \hat{y}_{b3}) & L(y_{24}, \hat{y}_{24}) \\ \dots & \dots & \dots & \dots \\ L(y_{b1}, \hat{y}_{b1}) & (y_{b2}, \hat{y}_{b2}) & L(y_{b3}, \hat{y}_{b3}) & L(y_{b4}, \hat{y}_{b4}) \end{bmatrix}.$$

Затем происходит усреднение потерь:

$$Loss = \frac{1}{b \cdot 4} \sum_{i=1}^{b} \sum_{j=1}^{4} L\left(y_{ij}, \hat{y}_{ij}\right), \quad (12)$$

что и будет окончательным результатом.

Если обобщить все вышеизложенные шаги, то получаем функцию потерь вида:

$$Loss = \frac{1}{b \cdot 4} \sum_{i=1}^{b} \sum_{j=1}^{4} -\left(y_{ij} \cdot \log\left(\sigma(x_{ij})\right) + \left(1 - y_{ij}\right) \cdot \log(1 - \sigma(x_{ij})\right). \quad (13)$$

В нашем датасете классы несбалансированные, из-за чего функция потерь может смещаться. Для устранения этого недостатка добавляют веся к каждой положительной метке. Тогда функция потерь будет выглядеть следующим образом:

$$Loss = \frac{1}{b \cdot 4} \sum_{i=1}^{b} \sum_{j=1}^{4} -\left(p_{ij} y_{ij} \cdot \log\left(\sigma(x_{ij})\right) + \left(1 - y_{ij}\right) \cdot \log(1 - \sigma(x_{ij})\right). \quad (14)$$

Именно это функция применяется в данном проекте.

Веса же мы берем такие:

$$P = \left[\frac{n}{n_{mi}}, \frac{n}{n_{sttc}}, \frac{n}{n_{cd}}, \frac{n}{n_{hyp}}\right],$$

где

- $n -$ общее кол-во образцов,
- n_{mi} – кол-во образцов с меткой, принадлежащих классу MI,
- n_{sttc} – кол-во образцов с меткой, принадлежащих классу STTC,
- n_{cd} – кол-во образцов с меткой, принадлежащих классу CD,
- n_{hyp} – кол-во образцов с меткой, принадлежащих классу HYP.

8.2. Функции активации

Функция активации в нейронных сетях — это математическая функция, которая применяется к выходу каждого нейрона (или узла) в сети. Она преобразует входное значение нейрона в выходное, которое затем передается на вход следующего слоя сети или используется как окончательный результат. Основная цель функции активации — добавить нелинейность в модель, что позволяет нейронной сети обучать и моделировать сложные данные.

Описание функций активаций, используемых в проекте:

- линейный выпрямитель (ReLU):

$$f(x) = ReLU(x) = \max(0, x); \quad (15)$$

$$\text{область значений: } [0, \infty].$$

Используется в AlexNet;

- логистическая или сигмоид (sigmoid):

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}; \quad (16)$$

$$\text{область значений: } (0, 1).$$

Применяется в GRU и BiGRU в контролерах шлюзов.

- Гиперболический тангенс (tanh):

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}; \quad (17)$$

$$\text{область значений: } (-1, 1).$$

Применяется в GRU и BiGRU в качестве основной функции активации.

8.3. Оптимизаторы

Алгоритм обратного распространения ошибки используется для вычисления градиентов функции потерь по параметрам нейронной сети. Это позволяет обновлять веса сети, чтобы минимизировать функцию потерь.

Шаги алгоритма:

1) Прямое распространение (Forward Propagation):
   - Входные данные проходят через сеть, и на каждом слое вычисляются активации.
   - На выходном слое вычисляется предсказание сети.
2) Вычисление функции потерь (Loss Calculation):
   - Разница между предсказанием и реальным значением используется для вычисления ошибки (например, кросс-энтропия для классификации).
3) Обратное распространение ошибки (Backward Propagation):
   - Ошибка из выходного слоя передается назад через сеть.
   - На каждом слое вычисляется градиент ошибки по отношению к весам и смещениям.
4) Обновление весов (Weight Update):
   - Используя вычисленные градиенты, корректируем веса и смещения с помощью метода оптимизации (например, SGD или Adam).

Стохастический градиентный спуск (SGD)

SGD — это метод оптимизации, использующий случайные подвыборки данных для вычисления градиентов, что ускоряет обучение и позволяет обрабатывать большие датасеты.

Шаги алгоритма:

1) Выбор мини-партии (Mini-batch Selection). Из тренировочного набора данных случайным образом выбирается подвыборка (мини-пакет).

2) Вычисление градиентов (Gradient Calculation). Для каждой мини-пакет вычисляются градиенты функции потерь по весам.

3) Обновление параметров (Parameter Update). Параметры обновляются по формуле:

$$W = W - \eta \nabla L(W)W, \tag{18}$$

где η — скорость обучения, $\nabla L(W)$ — градиент функции потерь.

Преимущества и недостатки:

- Преимущества:
  - Быстрое обновление параметров.
  - Хорошая масштабируемость для больших наборов данных.
  - Снижает вероятность застревания в локальных минимумах.
- Недостатки:
  - Высокая дисперсия градиентов может затруднять достижение оптимума.
  - Может потребоваться настройка скорости обучения.

Adam (Adaptive Moment Estimation) — это алгоритм оптимизации, который объединяет идеи AdaGrad и RMSProp и автоматически адаптирует скорости обучения для каждого параметра.

Шаги алгоритма:

1) Инициализация параметров. Векторы моментов m_t и v_t инициализируются нулями.

2) Обновление моментов (Moment Updates). Первый момент (среднее градиентов):

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{19}$$

где β_1 — параметр сглаживания первого момента (обычно около 0.9).

Второй момент (среднее квадрата градиентов):

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 v_t, \tag{20}$$

где β_2 — параметр сглаживания второго момента (обычно около 0.999).

3) Коррекция смещения (Bias Correction). Скорректированные моменты:

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{21}$$

$$\widehat{v}_t = \frac{v_t}{1 - \beta_2^t}. \quad (22)$$

4) Обновление параметров (Parameter Update). Параметры обновляются по формуле:

$$W_t = W_{t-1} - \eta \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon}, \quad (23)$$

где η — скорость обучения, ϵ — малое число для предотвращения деления на ноль.

Преимущества и недостатки:

- Преимущества:
  - Адаптивные скорости обучения для каждого параметра.
  - Быстрое сходимость.
  - Хорошо работает на шумных данных и с большими параметрическими пространствами.
- Недостатки:
  - Могут потребоваться тонкая настройка гиперпараметров.
  - В некоторых случаях может не давать значительных преимуществ перед другими методами оптимизации.

9. Метрики качества для оценки моделей

Для оценки эффективности модели МО используются различные метрики качества. Они позволяют определить, насколько хорошо модель справляется со своей задачей.

Четыре основные метрики в задаче классификации это

- True Positive (TP) – истинно положительные предсказания. Это количество объектов, которые действительно принадлежат положительному классу и которые модель также корректно идентифицировала как принадлежащие этому классу;
- True Negative (TN) – истинно отрицательные предсказания. Это количество объектов, которые действительно принадлежат отрицательному классу и которые модель правильно идентифицировала как принадлежащие этому классу;
- False Positive (FP) – ложно положительные предсказания. Это количество объектов, которые действительно принадлежат отрицательному классу, но модель неверно их классифицировала как принадлежащие положительному классу;
- False Negative (FN) – ложно отрицательные предсказания. Это количество объектов, которые действительно принадлежат положительному классу, но модель неверно отнесла их к отрицательному классу.

Остальные метрики базируются на них.

Ниже приведен список важных оценок качества:

1) Чувствительность (Se, Sensitivity/Recall). Измеряет способность модели правильно идентифицировать положительные случаи из всех действительно положительных случаев

$$\text{Sensitivity} = \frac{TP}{TP + FN}. \quad (24)$$

2) Специфичность (Sp, Specificity). Измеряет способность модели правильно идентифицировать отрицательные случаи из всех действительно отрицательных случаев

$$Specifity = \frac{TN}{TN + FP}. \quad (25)$$

3) Точность (Ac, Accuracy). показывает долю правильных предсказаний модели среди всех предсказаний

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \quad (27)$$

4) G-среднее (G-mean, Geometric Mean) – измеряет баланс между чувствительностью и специфичностью, предоставляя более сбалансированное представление о качестве модели по сравнению с другими метриками, такими как accuracy, которые могут быть смещены в сторону доминирующего класса. G-mean особенно полезна при работе с несбалансированными наборами данных.

G-mean определяется как геометрическое среднее из чувствительности и специфичности:

$$G - mean = \sqrt{Sensitivity \times Specificity}. \quad (28)$$

Наши данные несбалансированные, поэтому применение Accuracy будет ошибочным, т.к. ее оценка смещается в сторону доминирующего класса.

Метрики Sensitivity и Specificity имеют важную роль в медицинской предметной области, так как при маленьком значении первой – увеличивается риск пропуска патологии, а при маленьком значении второй – увеличивается риск на подозрение патологии и назначении дополнительных обследований.

Исходя из вышеизложенных соображений, основной оценкой качества модели в данном проекте выступает Sensitivity.

10. Обучение и тестирование моделей

Обучение проводилось на NVIDIA GeForce GTX 1650, использованием библиотеки PyTorch. Код реализации обучения представлен в Приложении Г.

10.1. GRU

Параметры запуска обучения

```
model_parameters = {
    "input_size": 3,
    "hidden_size": 128,
    "num_layers": 2,
    "device": "cuda",
}
parameters = {
    "epochs": 15,
    "batch_size": 256,
    "learning_rate": 0.01,
    "l2_decay": 0.0,
    "optimizer": "adam",
    "device": "cuda",
}
```

Таблица 11. Сводка по эпохам GRU

	Training-loss	Validation-loss	Sensitivity	Specificity
1	1.2145	1.0748	0.8323	0.5257
2	0.9252	0.9262	0.6827	0.8431
3	0.8160	0.8141	0.8050	0.7813
4	0.7618	0.8420	0.8880	0.6418
5	0.7495	0.8711	0.7657	0.7955

	Training-loss	Validation-loss	Sensitivity	Specificity
6	0.7212	0.8494	0.7433	0.8418
7	0.7124	0.8113	0.8443	0.7113
8	0.7121	0.8507	0.7204	0.8468
9	0.6796	0.7748	0.8176	0.7925
10	0.6629	0.7923	0.8056	0.7957
11	0.6746	0.8013	0.8170	0.7828
12	0.6955	0.7817	0.8236	0.7747
13	0.6660	0.7923	0.8476	0.7599
14	0.6660	0.8153	0.8454	0.7383
15	0.6750	0.8010	0.8362	0.7558

Таблица 12. Тестирование GRU

	TP	TN	FP	FN	Sensitivity	Specificity	G-mean
MI	31.17%	23.4%	24.59%	24.92%	85.45%	76.33%	80.77%
STTC	30.64%	24.46%	22.82%	18.38%	88.68%	78.41%	83.39%
CD	25.46%	27.41%	14.38%	34.89%	77.42%	86.6%	81.88%
HYP	12.73%	24.73%	38.21%	21.81%	73.28%	68.7%	70.95%
all	100.0%	100.0%	100.0%	100.0%	82.45%	77.22%	79.79%

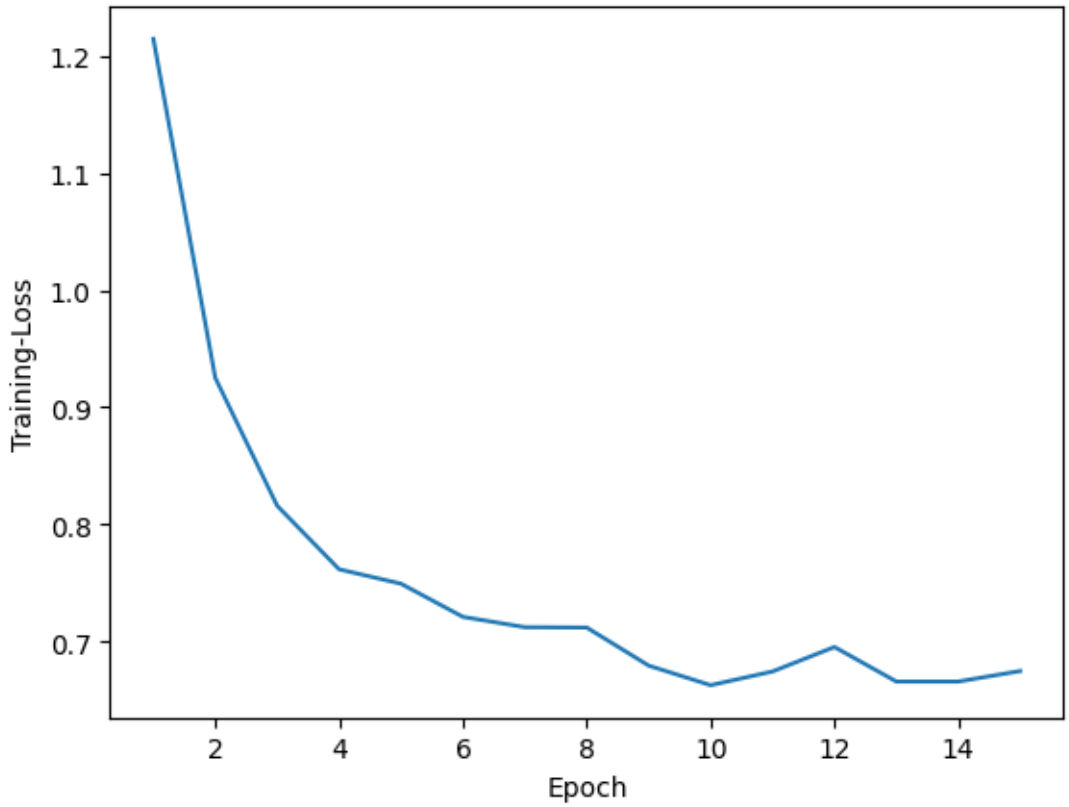

Рисунок 28. График функций потерь на тренировочном наборе GRU

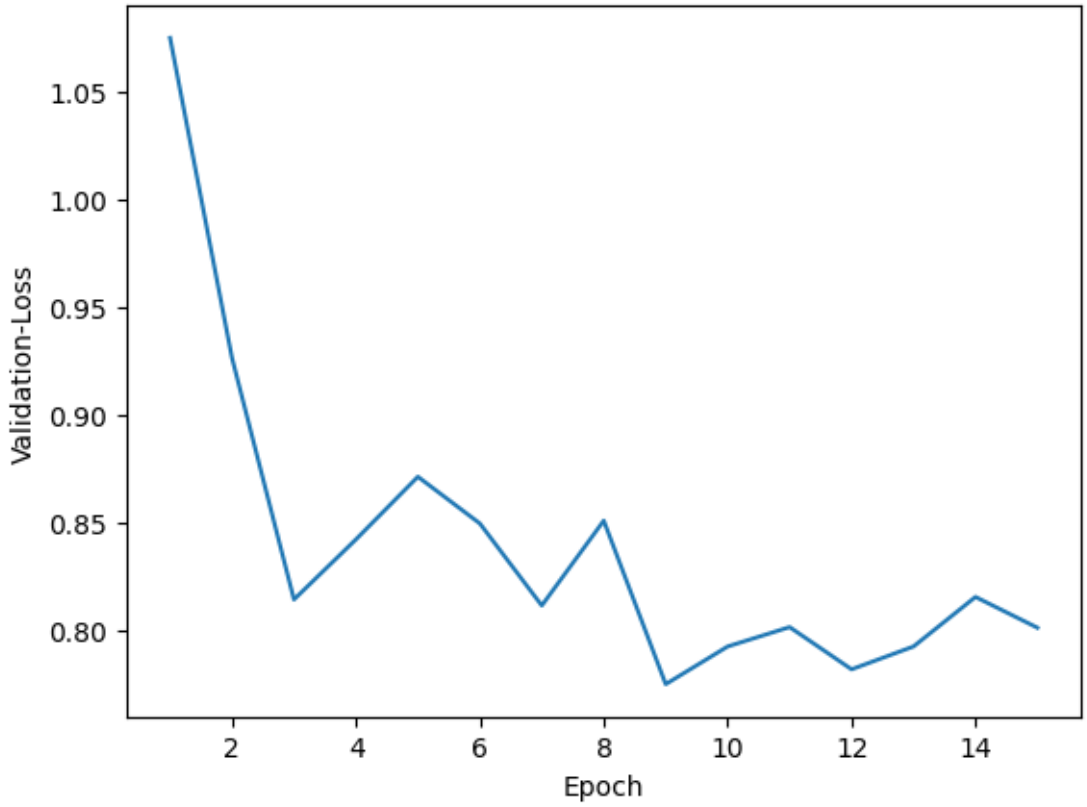

Рисунок 29. График функций потерь на валидационном наборе GRU

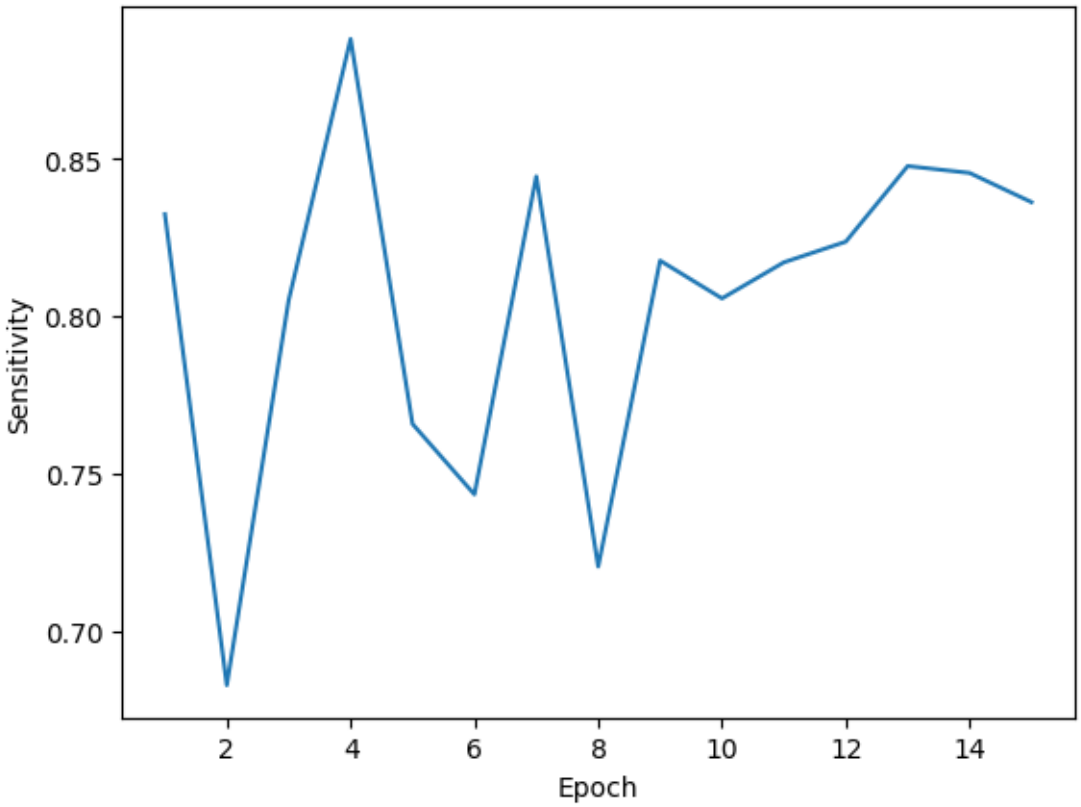

Рисунок 30. График Sensitivity GRU

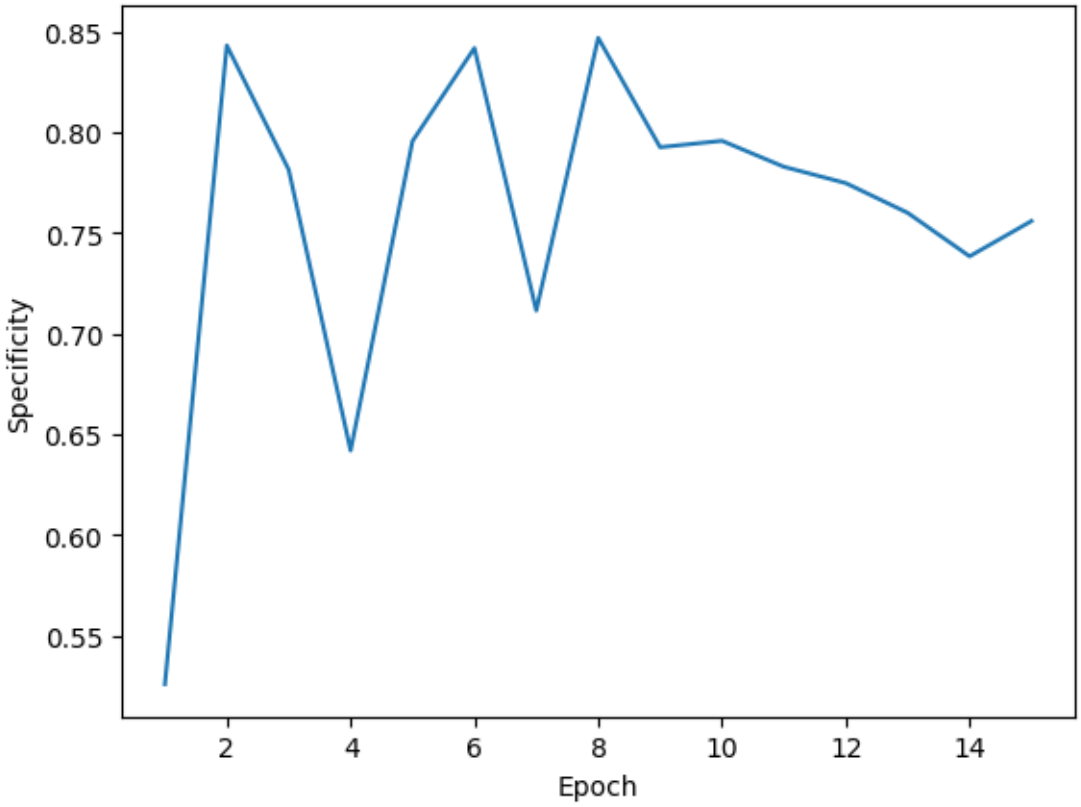

Рисунок 31. График Specificity GRU

10.2. BiGRU

Параметры запуска обучения

parameters

{'epochs': 100, 'patience_limit': 15, 'batch_size': 256, 'learning_rate': 0.01, 'l2_decay': 0.0, 'optimizer': 'adam', 'device': 'cuda'}

model_parameters

{'input_size': 3, 'hidden_size': 128, 'num_layers': 2, 'dropout': 0.3, 'device': 'cuda'}

Таблица 13. Сводка по эпохам двунаправленного GRU

	Training-loss	Validation-loss	Sensitivity	Specificity
1	1.1707	1.0972	0.7018	0.6390
2	0.8903	0.9219	0.8061	0.7467
3	0.7907	0.8268	0.8274	0.7348
4	0.7607	0.8363	0.8209	0.7621
5	0.8055	0.8493	0.8706	0.6687
6	0.7740	0.8463	0.7733	0.7932
7	0.7643	0.9546	0.6434	0.8884
8	0.7579	0.8450	0.8017	0.7557
9	0.7418	0.8244	0.7848	0.8186
10	0.7602	0.9322	0.8012	0.7008
11	0.8802	1.0004	0.6472	0.8484
12	0.9619	0.9966	0.7264	0.7111
13	0.9735	1.0701	0.8826	0.4549
14	0.9725	0.9928	0.6963	0.7877
15	0.9367	0.9640	0.7908	0.6790
16	0.9046	1.0323	0.6253	0.8244

	Training-loss	Validation-loss	Sensitivity	Specificity
17	0.8938	0.9897	0.8465	0.5886
18	0.8925	1.0632	0.8979	0.4514
19	0.9015	0.9177	0.7848	0.7195
20	0.8680	1.0075	0.6319	0.8363
21	0.8743	0.9550	0.8525	0.6250
22	0.8673	0.9209	0.7193	0.7705
23	0.8550	0.9101	0.8454	0.6547
24	0.8580	0.8932	0.8269	0.6844

Таблица 14. Тестирование двунаправленного GRU

	TP	TN	FP	FN	Sensitivity	Specificity	G-mean
MI	29.17%	25.39%	19.7%	34.02%	78.91%	74.88%	76.87%
STTC	30.85%	24.91%	22.17%	18.18%	88.1%	72.21%	79.76%
CD	25.94%	26.33%	20.08%	32.26%	77.82%	75.21%	76.5%
HYP	14.05%	23.37%	38.06%	15.54%	79.77%	58.68%	68.42%
all	100.0%	100.0%	100.0%	100.0%	81.36%	69.81%	75.36%

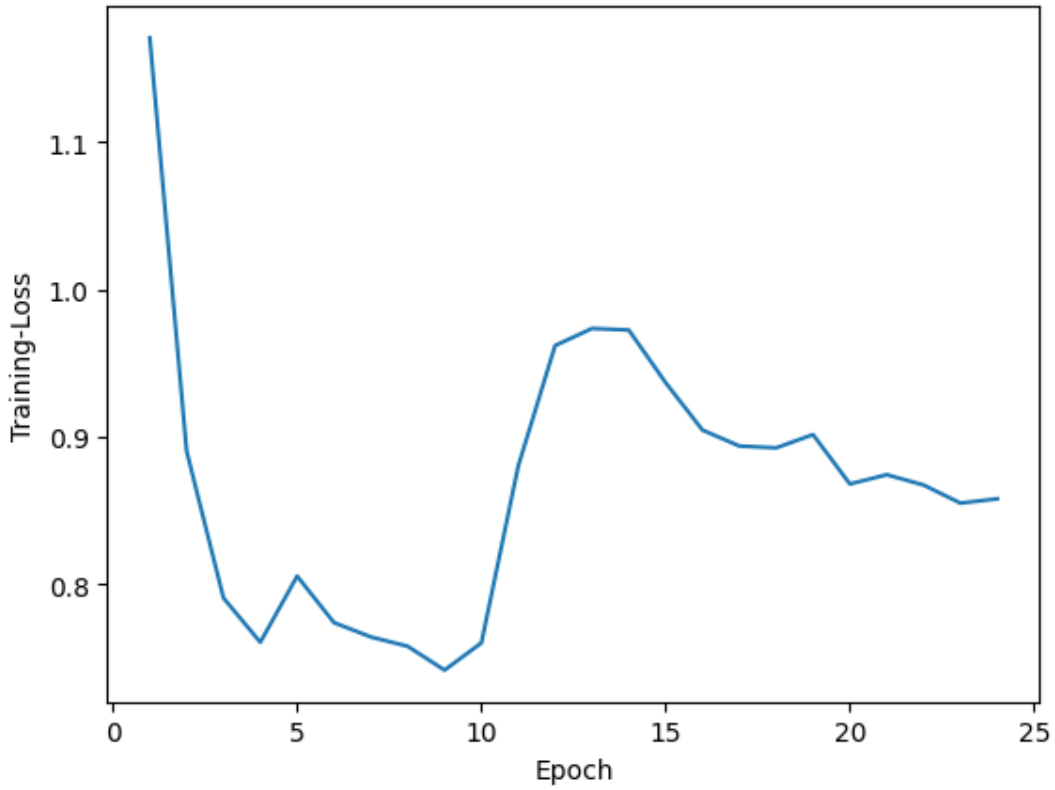

Рисунок 32. График функций потерь на тренировочном наборе двунаправленного GRU

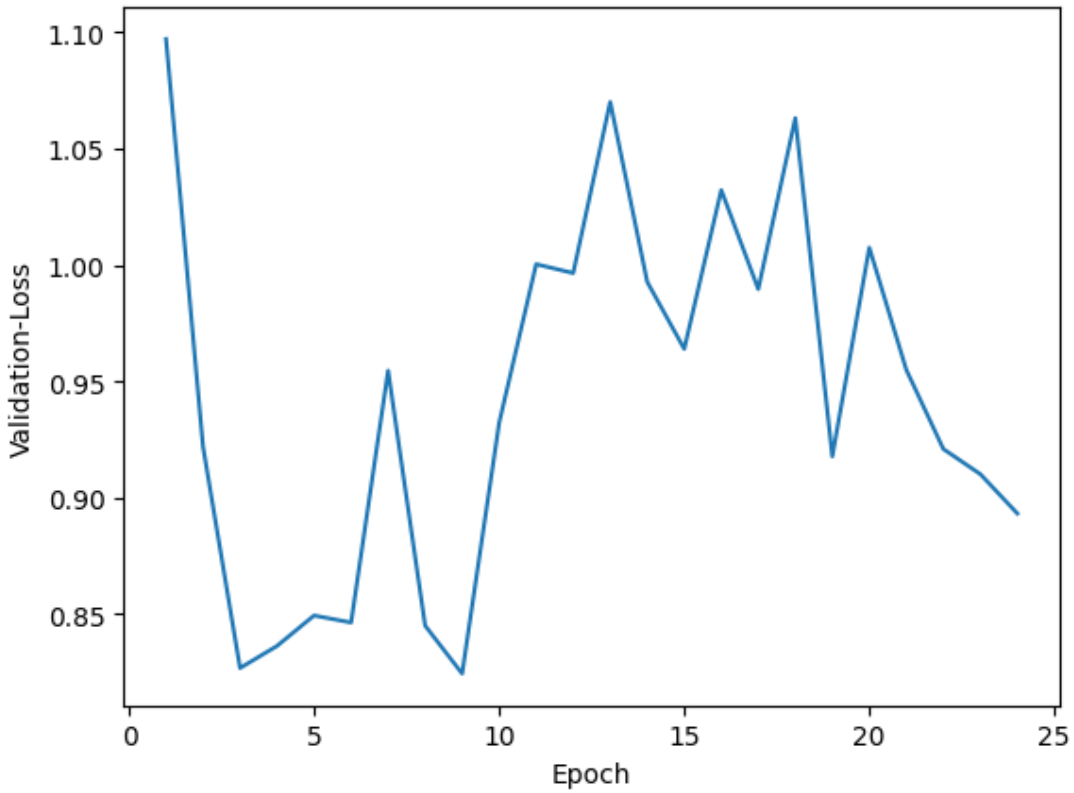

Рисунок 33. График функции потерь на валидационном наборе двунаправленного GRU

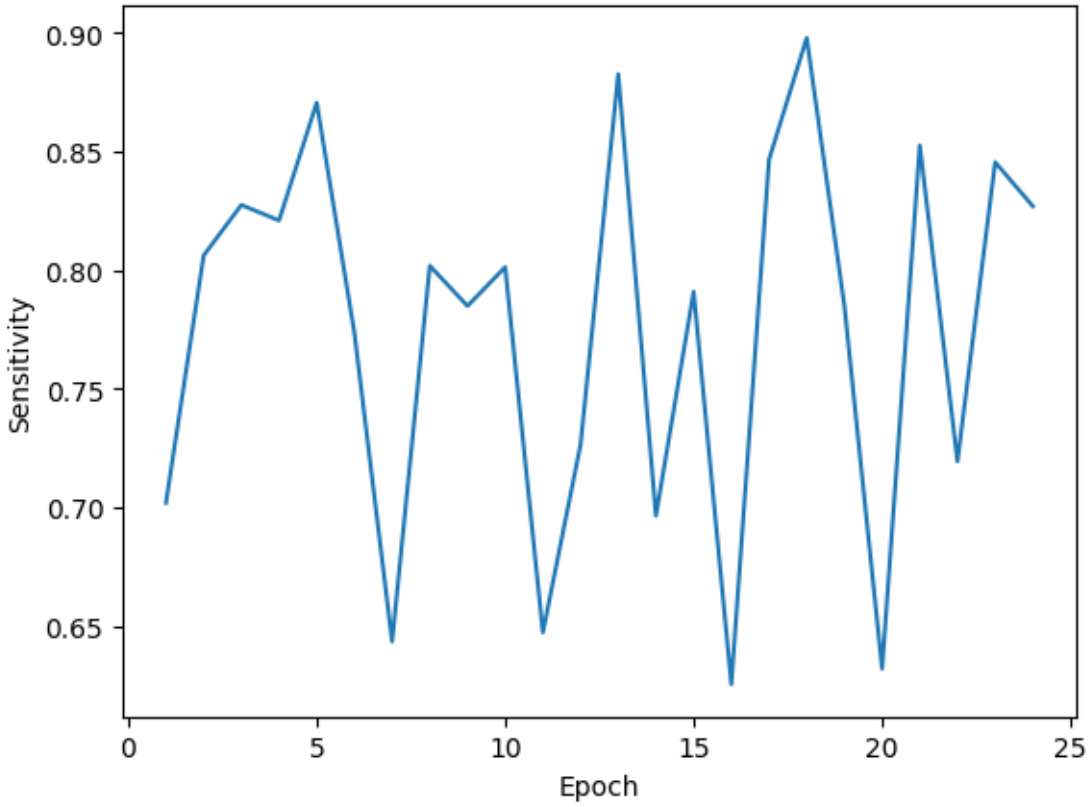

Рисунок 34. График Sensitivity двунаправленного GRU

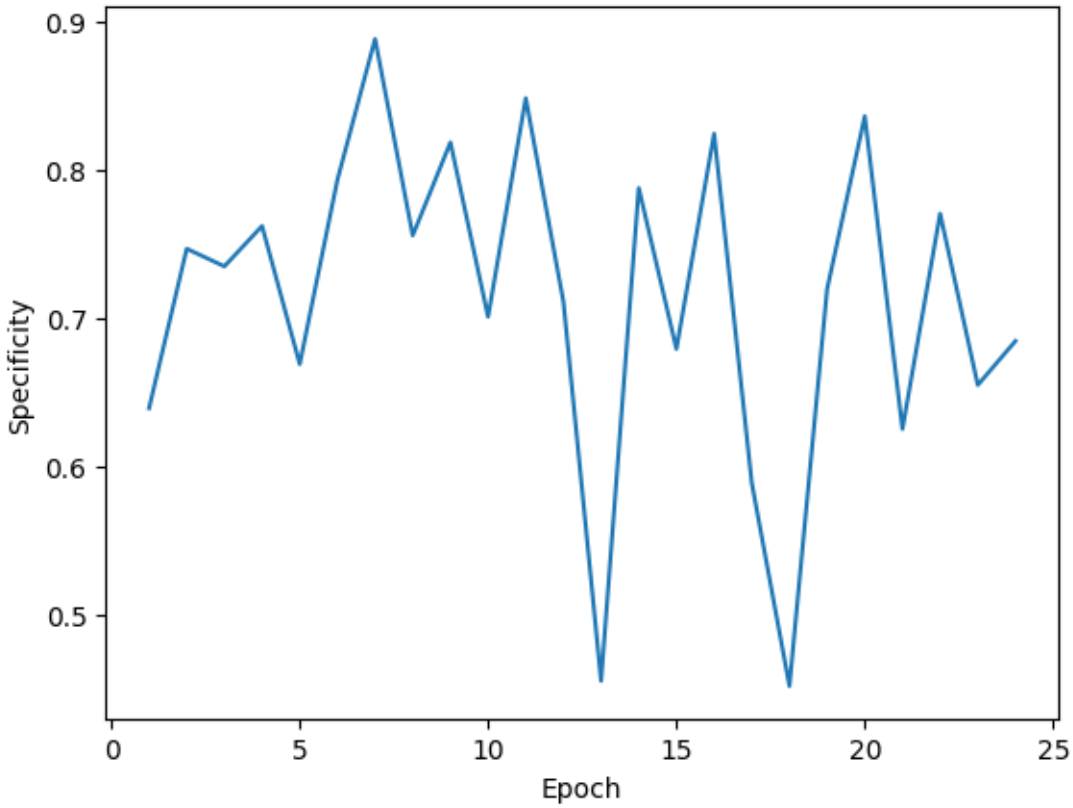

Рисунок 35. График Specificity двунаправленного GRU

10.3. AlexNet

Таблица 15. Сводка по эпохам AlexNet

	Training-loss	Validation-loss	Sensitivity	Specificity
1	1.1730	1.1582	0.4894	0.8274
2	1.0941	1.0908	0.6019	0.7957
3	1.0451	1.0914	0.8061	0.5438
4	1.0150	1.1476	0.6800	0.6496
5	0.9898	1.0406	0.8094	0.5675
6	0.9658	0.9982	0.8738	0.5157
7	0.9470	0.9722	0.7865	0.6827
8	0.9382	1.0151	0.7548	0.6699
9	0.9203	1.0289	0.7662	0.6686
10	0.9087	0.9812	0.8214	0.6403
11	0.8998	0.9458	0.7422	0.7526
12	0.8851	0.9549	0.7619	0.6856
13	0.8696	1.0019	0.7198	0.7722
14	0.8617	0.9532	0.8028	0.6963
15	0.8476	0.9784	0.7717	0.6586
16	0.8370	0.9741	0.8345	0.6238
17	0.8246	0.9690	0.7821	0.7073
18	0.8117	0.9825	0.7193	0.7593
19	0.7964	1.0158	0.7460	0.7341

Таблица 16. Тестирование AlexNet

	TP	TN	FP	FN	Sensitivity	Specificity	G-mean
MI	32.84%	20.11%	34.07%	22.43%	80.18%	63.29%	71.24%
STTC	28.97%	26.28%	17.68%	27.16%	74.66%	81.28%	77.9%
CD	28.74%	22.65%	29.67%	22.63%	77.82%	69.04%	73.3%
HYP	9.46%	30.96%	18.58%	27.78%	48.47%	82.95%	63.41%
all	100.0%	100.0%	100.0%	100.0%	73.43%	74.49%	73.96%

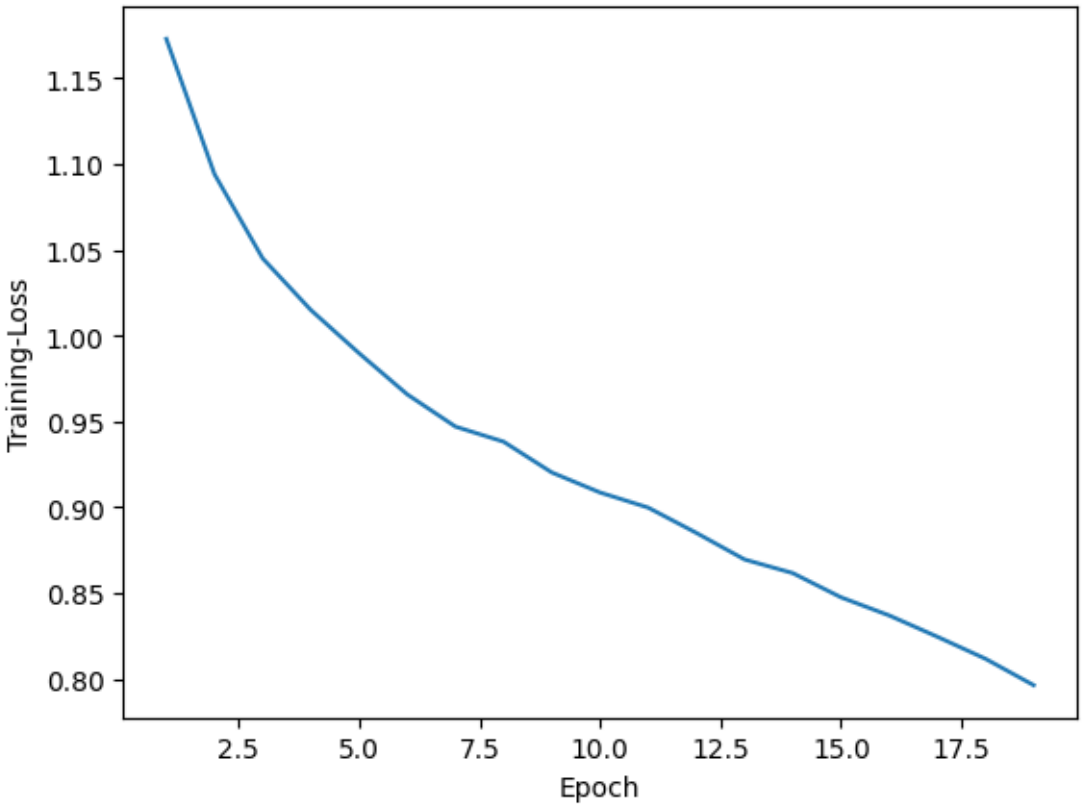

Рисунок 36. График функций потерь на тренировочном наборе AlexNet

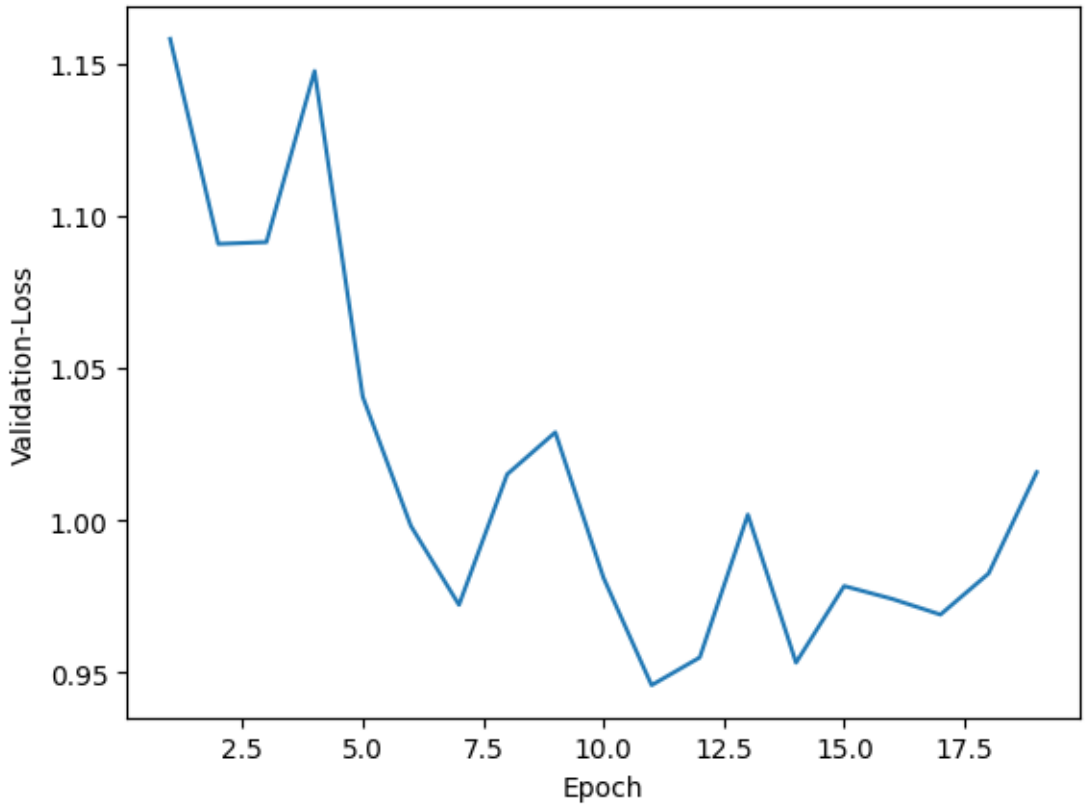

Рисунок 37. График функции потерь на валидационном наборе AlexNet

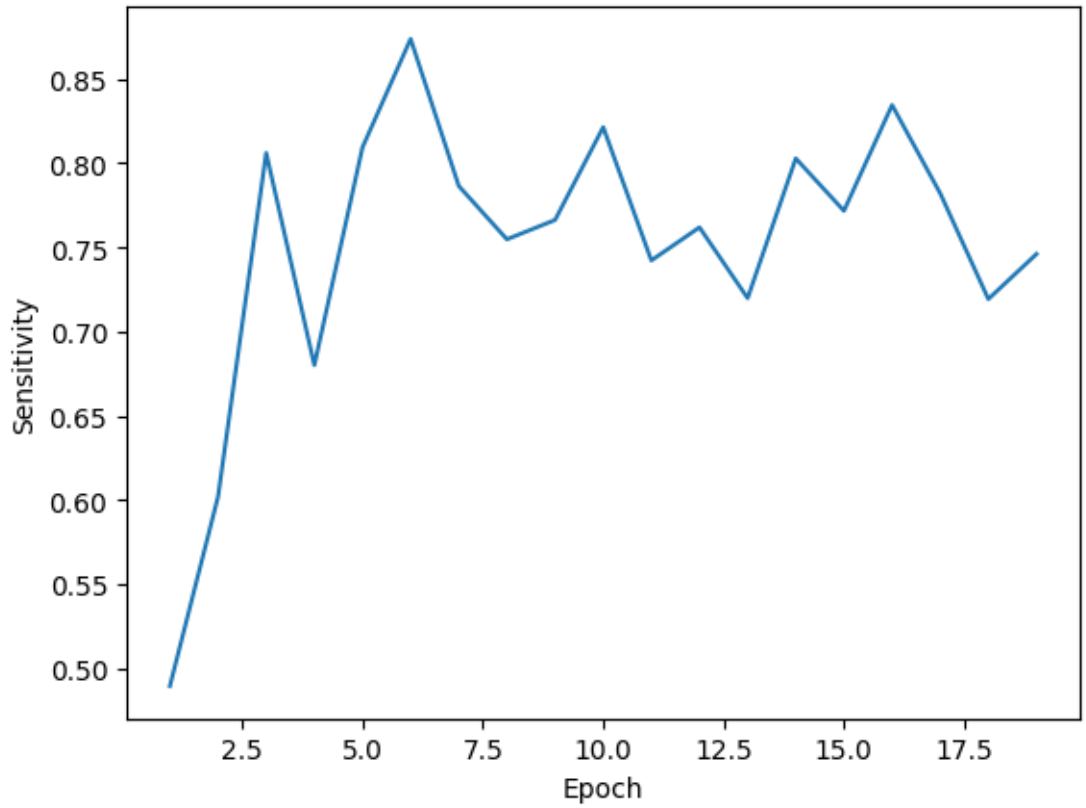

Рисунок 38. График Sensitivity AlexNet

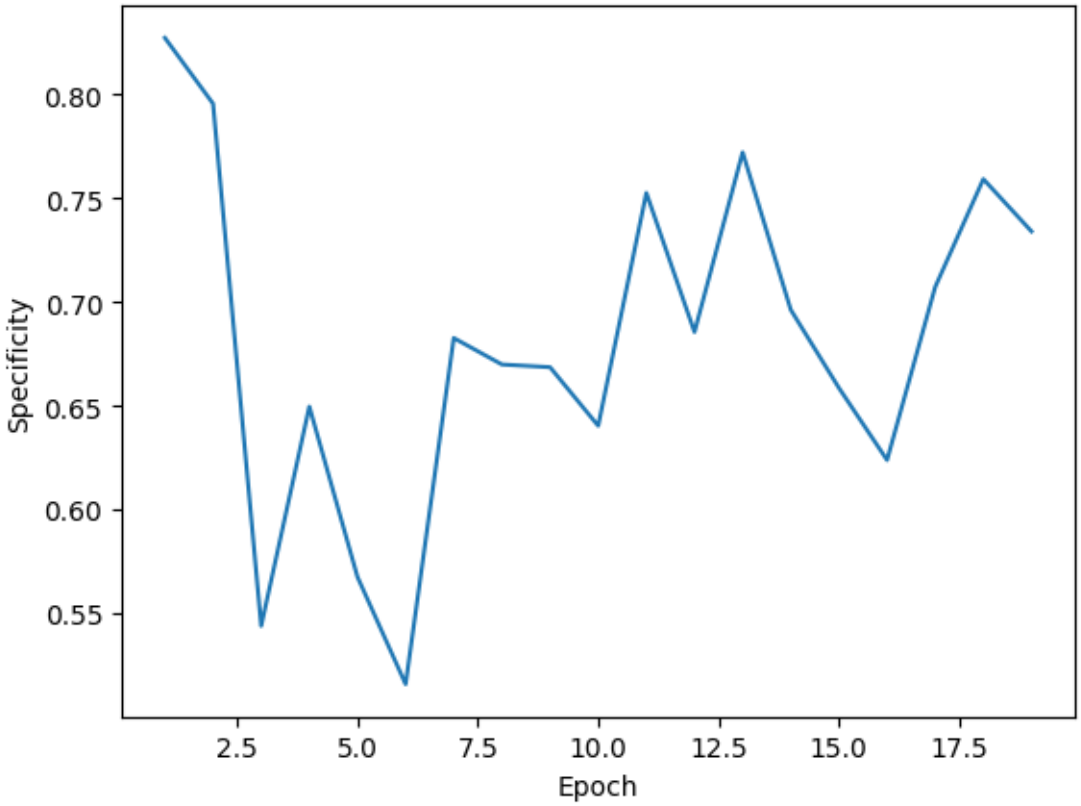

Рисунок 39. График Specificity AlexNet

Заключение

В ходе выполнения дипломной работы была разработана нейронная сеть для классификации данных электрокардиограммы по пяти классам:

- нормальный,
- инфаркт миокарда,
- изменение сегмента ST/T,
- нарушение проводимости,
- гипертрофия –

со следующими показателями качества:

- Sensitivity: 82.45%,
- Specificity: 77.22%,
- G-mean: 79.79%.

Такие показатели получила RNN с ячейкой памяти GRU.

Чтобы достичь цели были проделаны следующие шаги:

1) разработано графическое приложение для удобной настройки параметров обучения нейронной сети;

2) проведена предобработка данных, заключающаяся в фильтрации и нормализации сигналов ЭКГ в целях уменьшения влияния выбросов; а также в преобразовании сигналов в изображения GAF, RP, MTF в целях обучения на них сверточных нейронных сетей;

3) реализованы и обучены три архитектуры нейронных сетей: рекуррентные GRU и двунаправленный GRU и сверточный AlexNet;

Сравнив результаты тестирования моделей наилучшим из них оказалось GRU.

Основные причины, по которым показатели не превзошли полученных, могу быть следующими:

- несбалансированность классов и недостаточное количество меток некоторых из них. Например, можно заметить, что образцов с меткой HYP

меньше всего. Логичным следствием этого является наихудшие предсказательные способности моделей по отношению к этому классу;

- изображения, полученные в ходе GAF, RP и MTF трансформаций в последствии усекаются из размеров 1000×1000 до 224×224, из-за чего происходит потеря информации. Усечение происходит, во-первых, по причине недостаточных вычислительных возможностей ЭВМ, на котором происходило обучения, во-вторых, архитектура AlexNet разработана под изображения 224×224;
- не исключено, что гиперпараметры обучения подобраны не самые подходящие для получения более надежного результата.

Устранив данные недостатки, можно улучшить качество моделей.

Список литературы

1. 10 ведущих причин смерти в мире [Электронный ресурс]: Всемирная организация здравоохранения. URL: https://www.who.int/ru/news-room/fact-sheets/detail/the-top-10-causes-of-death.
2. Хэмптон Дж. Р. Основы ЭКГ: пер. с англ. – М.: Мед. лит., 2006 – 224 с., ил.
3. E. Pasolli and F. Melgani, "Active learning methods for electrocardiographic signal classification", IEEE Trans. Inf. Technol. Biomed., vol. 14, no. 6, pp. 1405-1416, Nov. 2010, https://ieeexplore.ieee.org/abstract/document/5610575.
4. Y. H. Hu, S. Palreddy and W. J. Tompkins, "A patient-adaptable ECG beat classifier using a mixture of experts approach", IEEE Trans. Biomed. Eng., vol. 44, no. 9, pp. 891-900, Sep. 1997, https://ieeexplore.ieee.org/document/623058.
5. V. Chouhan and S. Mehta, "Threshold-based detection of P and T-wave in ECG using new feature signal", Int. J. Comput. Sci. Netw. Secur., vol. 8, no. 2, pp. 144-153, 2008, https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1cc1ef734898e340126bdb0eec5fbadb538de08b.
6. N. A. Bhaskar, "Performance analysis of support vector machine and neural networks in detection of myocardial infarction", Procedia Comput. Sci., vol. 46, pp. 20-30, Jan. 2015, https://www.sciencedirect.com/science/article/pii/S1877050915000447?via%3Dihub.
7. K. Minami, H. Nakajima and T. Toyoshima, "Real-time discrimination of ventricular tachyarrhythmia with Fourier-transform neural network", IEEE Trans. Biomed. Eng., vol. 46, no. 2, pp. 179-185, Feb. 1999, https://ieeexplore.ieee.org/document/740880.
8. H. Khorrami and M. Moavenian, "A comparative study of DWT CWT and DCT transformations in ECG arrhythmias classification", Expert Syst. Appl., vol. 37, no. 8, pp. 5751-5757, Aug. 2010,

https://www.sciencedirect.com/science/article/abs/pii/S0957417410000722?via%3Dihub.

9. L. N. Sharma, R. K. Tripathy and S. Dandapat, "Multiscale energy and eigenspace approach to detection and localization of myocardial infarction", IEEE Trans. Biomed. Eng., vol. 62, no. 7, pp. 1827-1837, Jul. 2015, https://ieeexplore.ieee.org/document/7047810.

10. P.-C. Chang, J.-J. Lin, J.-C. Hsieh and J. Weng, "Myocardial infarction classification with multi-lead ECG using hidden Markov models and Gaussian mixture models", Appl. Soft Comput., vol. 12, no. 10, pp. 3165-3175, Oct. 2012, https://www.sciencedirect.com/science/article/abs/pii/S156849461200275X?via%3Dihub.

11. H. L. Lu, K. Ong and P. Chia, "An automated ECG classification system based on a neuro-fuzzy system", Proc. Comput. Cardiol., vol. 27, pp. 387-390, 2000, https://ieeexplore.ieee.org/document/898538.

12. K. A. Sidek, I. Khalil and H. F. Jelinek, "ECG biometric with abnormal cardiac conditions in remote monitoring system", IEEE Trans. Syst. Man Cybern. Syst., vol. 44, no. 11, pp. 1498-1509, Nov. 2014, https://ieeexplore.ieee.org/document/6867363.

13. Hemaxi Narotamo, Mariana Dias, Ricardo Santos, André V. Carreiro, Hugo Gamboa, Margarida Silveira, Deep learning for ECG classification: A comparative study of 1D and 2D representations and multimodal fusion approaches, Biomedical Signal Processing and Control. 93 (2024) 106141, https://www.sciencedirect.com/science/article/pii/S174680942400199X.

14. ECG Interpretation for Everyone: An On-The-Spot Guide, First Edition. Fred Kusumoto and Pam Bernath. 2012 John Wiley & Sons, Ltd. Published 2012 by John Wiley & Sons, Ltd. Перевод – Абашин А.А., 2014.

15. Техника снятия ЭКГ [Электронный ресурс]: МТ диагностика. URL: https://mtdiagnostica.ru/readpage_technika-sniatiia-iekg.html.

16. PTB-XL, a large publicly available electrocardiography dataset [Электронный ресурс]: PhysioNet. URL: https://physionet.org/content/ptb-xl/1.0.3/.

17. P. Wagner, N. Strodthoff, R.-D. Bousseljot, D. Kreiseler, F.I. Lunze, W. Samek, T. Schaeffter, PTB-XL, A large publicly available electrocardiography dataset, Sci. Data 7 (154) (2020) https://www.nature.com/articles/s41597-020-0495-6.

18. N.T. Bui, G.S. Byun, The comparison features of ECG signal with different sampling frequencies and filter methods for real-time measurement, Symmetry (ISSN: 20738994) 13 (2021) https://www.mdpi.com/2073-8994/13/8/1461.

19. H. Zhu, Y. Pan, K.T. Cheng, R. Huan, A lightweight piecewise linear synthesis method for standard 12-lead ECG signals based on adaptive region segmentation, PLoS One (ISSN: 1932-6203) 13 (2018) e0206170 https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0206170.

20. H.Z. Li, P. Boulanger, A Survey of Heart Anomaly Detection Using Ambulatory Electrocardiogram (ECG), Sensors (Basel, Switzerland) (ISSN: 14248220) 20 (2020) https://www.mdpi.com/1424-8220/20/5/1461.

21. K.N. Rajesh, R. Dhuli, Classification of ECG heartbeats using nonlinear decomposition methods and support vector machine, Comput. Biol. Med. (ISSN: 1879-0534) 87 (2017) 271–284 https://www.sciencedirect.com/science/article/abs/pii/S0010482517301701?via%3Dihub.

22. H. Sivaraks, C.A. Ratanamahatana, Robust and accurate anomaly detection in ECG artifacts using time series motif discovery, Comput. Math. Methods Med. (ISSN: 17486718) 2015 (2015) https://www.hindawi.com/journals/cmmm/2015/453214/.

23. Z. Ahmad, A. Tabassum, L. Guan, N.M. Khan, ECG heartbeat classification using multimodal fusion, IEEE Access 9 (2021), https://ieeexplore.ieee.org/document/9486862.

24. Convert a Time Series Into an Image with Gramian Angular Fields and Markov Transition Fields [Электронный ресурс]: Lazy Programmer. URL: https://lazyprogrammer.me/convert-a-time-series-into-an-image/.

25. Жерон, Орельен. Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем, 2-е изд.: Пер. с англ. – СПБ.: ООО "Диалектика" 2020. – 1040 с.: ил. Парал. тит. англ.

26. RNN, LSTM, GRU и другие рекуррентные нейронные сети [Электронный ресурс]. URL: http://vbystricky.ru/2021/05/rnn_lstm_gru_etc.html.

27. AlexNet – сверточная нейронная сеть для классификации изображений [Электронный ресурс]: Neurohive. URL: https://neurohive.io/ru/vidy-nejrostej/alexnet-svjortochnaya-nejronnaja-set-dlja-raspoznavanija-izobrazhenij.

Приложение А

DataProcessingPage.py

```
class DataProcessingPage(ctk.CTkFrame):
    def __init__(self, controller):
        # 8 row 4 col
        super().__init__(controller)
        self.controller = controller

        self.grid_columnconfigure((0, 1, 2, 3), weight=1)
        self.grid_rowconfigure((0, 1, 2, 3, 4, 5, 6, 7, 8), weight=1)

        self.label = ctk.CTkLabel(
            self, text="Обработка данных", fg_color="lightblue", corner_radius=6
        )
        self.label.grid(row=0, column=0, padx=20, pady=20, sticky="ew", columnspan=4)

        # self.button_back = ctk.CTkButton(
        #     self, text="Назад", command=lambda: controller.show_frame("MainMenu")
        # )
        # self.button_back.grid(row=1, column=0, padx=20, pady=10, sticky="ew")

        self.window_download = None
        self.button_download = ctk.CTkButton(
            self, text="Скачать PTB-XL", command=self.download_data
        )
        self.button_download.grid(
            row=2, column=0, padx=20, pady=10, sticky="ew", columnspan=4
        )

        self.frame_frequency = FrequencyFrame(self)
        self.frame_frequency.grid(row=3, column=3, padx=20, pady=10, sticky="ew")

        self.frame_leads = LeadsFrame(self)
        self.frame_leads.grid(
            row=4, column=0, padx=20, pady=10, sticky="ew", columnspan=4
        )

        self.frame_processing_1d = ProcessingFrame(self, "1D обработка", 1)
        self.frame_processing_1d.grid(
            row=5, column=0, padx=20, pady=10, sticky="ew", columnspan=2
        )

        self.frame_processing_2d = ProcessingFrame(self, "2D обработка", 2)
        self.frame_processing_2d.grid(
            row=5, column=2, padx=20, pady=10, sticky="ew", columnspan=2
        )

        self.button_next_page = ctk.CTkButton(
            self,
            text="Выбрать нейронную сеть",
            command=lambda: controller.show_frame("NNSelectionPage"),
        )
        self.button_next_page.grid(row=6, column=3, padx=20, pady=(10, 20), sticky="ew")

    def download_data(self):
        if self.window_download is None or not self.window_download.winfo_exists():
            self.window_download = DownloadWindow(self)
        else:
            self.window_download.focus()

class DatasetFrame(ctk.CTkFrame):
    def __init__(self, parent):
        super().__init__(parent)
        self.path_data_raw = Path(PROJECT_ROOT, "data", "raw")
        self.raw_datasets = tuple(self.path_data_raw.iterdir())

        self.grid_columnconfigure(tuple(range(len(self.raw_datasets))), weight=1)

        self.title = ctk.CTkLabel(
            self, text="Датасет", fg_color="lightblue", corner_radius=6
        )
        self.title.grid(
            row=0,
            column=0,
            columnspan=len(self.raw_datasets),
            padx=10,
            pady=10,
            sticky="ew",
        )
        self.variable = ctk.StringVar(value="")

        self.radiobuttons = []
        for i, dataset in enumerate(self.raw_datasets):
            radiobutton = ctk.CTkRadioButton(
```

```
            self, text=dataset.name, variable=self.variable, value=dataset.name
        )
        radiobutton.grid(row=1, column=i, padx=5, pady=5, sticky="ew")
        self.radiobuttons.append(radiobutton)

    def get(self):
        return self.variable.get()

    def set(self, value):
        self.variable.set(value)
```

NNSelectionPage.py

```
PROJECT_ROOT = os.getenv("project_root")

class NNSelectionPage(ctk.CTkFrame):
    def __init__(self, controller):
        super().__init__(controller)
        self.controller = controller
        self.grid_rowconfigure((0, 1, 2, 3), weight=1)

        self.label = ctk.CTkLabel(
            self, text="Нейронные сети", fg_color="lightblue", corner_radius=6
        )
        self.label.grid(row=0, column=0, padx=20, pady=20, sticky="ew", columnspan=4)

        self.button_back = ctk.CTkButton(
            self,
            text="Назад",
            command=lambda: controller.show_frame("DataProcessingPage"),
        )
        self.button_back.grid(row=1, column=0, padx=20, pady=10, sticky="ew")

        self.var_nn = ctk.StringVar()

        self.frame_nn1d = NNFrame(self, 1)
        self.frame_nn1d.grid(
            row=2, column=0, padx=20, pady=10, sticky="ew", columnspan=2
        )

        self.frame_nn2d = NNFrame(self, 2)
        self.frame_nn2d.grid(
            row=2, column=2, padx=20, pady=10, sticky="ew", columnspan=2
        )

        self.button_next_page = ctk.CTkButton(
            self,
            text="Настройка гиперпараметров",
            command=self.next_page,
        )
        self.button_next_page.grid(row=3, column=3, padx=20, pady=(10, 20), sticky="ew")

    def next_page(self):
        path_nn = self.var_nn.get()
        if path_nn == "":
            return

        dimension = 1 if "nn1d" in path_nn else 2
        page = self.controller.frames["ParametersPage"]
        page.set_args(path_nn=path_nn, dimension=dimension)
        self.controller.show_frame(page.__class__.__name__)


class NNFrame(ctk.CTkFrame):
    def __init__(self, parent, dimension):
        super().__init__(parent)
        self.grid_columnconfigure(0, weight=1)
        self.label = ctk.CTkLabel(
            self,
            text=f"{dimension}D Нейронные Сети",
            fg_color="lightblue",
            corner_radius=6,
        )
        self.label.grid(row=0, column=0, padx=10, pady=20, sticky="ew")

        path_nns = Path(PROJECT_ROOT, "src", f"nn{dimension}d")
        self.radiobuttons = []
        for i, path_nn in enumerate(path_nns.iterdir(), start=1):
            if not path_nn.is_file() or path_nn.name == "__init__.py":
                continue

            radiobutton = ctk.CTkRadioButton(
                self,
                text=path_nn.stem,
                variable=parent.var_nn,
                value=str(path_nn),
            )
            radiobutton.grid(row=i, column=0, padx=10, pady=10, sticky="ew")
            self.radiobuttons.append(radiobutton)
```

ParametersPage.py

```
PROJECT_ROOT = os.getenv("project_root")
```

```
class ParametersPage(ctk.CTkFrame):
    def __init__(self, controller):
        super().__init__(controller)
        self.controller = controller
        self.grid_rowconfigure((0, 1, 2, 3), weight=1)

        self.label = ctk.CTkLabel(
            self, text="Гиперпараметры", fg_color="lightblue", corner_radius=6
        )
        self.label.grid(row=0, column=0, padx=20, pady=20, sticky="ew", columnspan=4)

        self.button_back = ctk.CTkButton(
            self,
            text="Назад",
            command=self.prev_page,
        )
        self.button_back.grid(row=1, column=0, padx=20, pady=10, sticky="ew")

        self.frame_parameters = ParametersFrame(self)
        self.frame_parameters.grid(
            row=2, column=0, padx=20, pady=20, sticky="ew", columnspan=2
        )

        self.frame_nn = NNFrame(self)
        self.frame_nn.grid(
            row=2, column=2, padx=20, pady=20, sticky="nsew", columnspan=2
        )

        self.frame_dataset = DatasetFrame(self)  # см. в init_hyperparameters
        self.frame_dataset.grid(
            row=3, column=0, padx=20, pady=20, sticky="ew", columnspan=4
        )

        self.frame_model_naming = ModelNamingFrame(self)
        self.frame_model_naming.grid(
            row=4, column=2, padx=20, pady=20, sticky="ew", columnspan=2
        )

        self.button_next_page = ctk.CTkButton(
            self,
            text="Обучить модель",
            command=self.next_page,
        )
        self.button_next_page.grid(row=5, column=3, padx=20, pady=(10, 20), sticky="ew")

    def set_args(self, **kwargs):
        """Получение параметров из предыдущей страницы

        kwargs:
            path_nn (str): путь к нейронной сети
            dimension (int): 1 или 2 (2D или 1D нейросеть будет обучаться)
        """
        self.kwargs = kwargs
        self.frame_nn.init(kwargs["path_nn"])
        self.frame_dataset.init(kwargs["dimension"])

    def prev_page(self):
        for widget in self.frame_dataset.winfo_children():
            widget.destroy()
        self.controller.show_frame("NNSelectionPage")

    def next_page(self):
        self.frame_parameters.reset_color()
        self.frame_model_naming.reset_color()

        if not self.frame_parameters.check_parameters():
            return
        if not self.frame_model_naming.check_name():
            return

        parameters = self.frame_parameters.get()
        model_name = self.frame_model_naming.get()

        dataset_name = self.frame_dataset.get()
        if dataset_name == "":
            return

        path_nn = self.frame_nn.get()
        # src.nn1(2)d
        module = ".".join(path_nn.parts[-3:-1])
        model_class = getattr(importlib.import_module(module), path_nn.stem)

        self.controller.frames["TrainingPage"].set_args(
            model_class=model_class,
            dimension=self.kwargs["dimension"],
            model_name=model_name,
            dataset_name=dataset_name,
            parameters=parameters,
        )
        self.controller.show_frame("TrainingPage")
```

TrainingPage.py

```
PROJECT_ROOT = os.getenv("project_root")

class RedirectText:
    def __init__(self, text_widget):
        self.text_widget = text_widget

    def write(self, string):
        self.text_widget.insert(ctk.END, string)

    def flush(self):
        pass  # Метод flush нужен для совместимости, но мы его не используем



class TrainingPage(ctk.CTkFrame):
    def __init__(self, controller):
        super().__init__(controller)
        self.controller = controller
        self.grid_rowconfigure((0, 1, 2, 3), weight=1)

        self.training_thread = None

        self.label = ctk.CTkLabel(
            self, text="Обучение модели", fg_color="lightblue", corner_radius=6
        )
        self.label.grid(row=0, column=0, padx=20, pady=20, sticky="ew", columnspan=4)

        self.button_back = ctk.CTkButton(
            self,
            text="Назад",
            command=self.prev_page,
        )
        self.button_back.grid(row=1, column=0, padx=20, pady=10, sticky="ew")

        self.button_start = ctk.CTkButton(
            self, text="Обучить модель", command=self.start
        )
        self.button_start.grid(
            row=2, column=0, padx=20, pady=20, sticky="ew", columnspan=4
        )

        self.label_status = ctk.CTkTextbox(self, width=800, height=400)
        self.label_status.grid(
            row=3, column=0, padx=20, pady=20, sticky="ew", columnspan=4
        )
        # self.label_status.configure(state="disabled")

    def set_args(self, **kwargs):
        """Принимает с предыдущей страницы данные

        kwargs:
            model_class (type[Module]): класс нейронной сети
            dimension (int): 1 или 2 (2D или 1D нейросеть будет обучаться)
            model_name (str): имя, которая под которой будет сохранена модель
            dataset_name (str): название датасета, на которой будет обучаться модель
            parameters (dict):
                (epochs, batch_size, learning_rate, l2_decay, optimizer: str = ["adam", "sgd"], device: str = ["cuda", "cpu", "mps"])
            model_parameters (dict): Параметры для конструктора модели. Default {}
        """
        self.kwargs = kwargs
        self.keep_stdout = sys.stdout
        self.keep_stderr = sys.stderr
        sys.stdout = RedirectText(self.label_status)
        sys.stderr = RedirectText(self.label_status)

    def start(self):
        for frame_name, frame in self.controller.frames.items():
            if frame_name != "TrainingPage" and frame_name != "ResultsPage":
                frame.destroy()
                self.controller.frames[frame_name] = None

        self.button_back.destroy()
        self.button_start.destroy()
        if self.training_thread is None or not self.training_thread.is_alive():
            print("Запуск обучения...\n")
            self.training_thread = threading.Thread(
                target=start_training, kwargs=self.kwargs
            )
            self.training_thread.daemon = True
            self.training_thread.start()
            self.check_training_thread()

    def check_training_thread(self):
        if self.training_thread.is_alive():
            # Если поток все еще активен, проверяем его снова через 100 мс
            self.after(100, self.check_training_thread)
        else:
            # Если поток завершился, обновляем статус
            print("Обучение завершено!\n")
            sys.stdout = self.keep_stdout
            sys.stderr = self.keep_stderr
```

```
        self.button_next = ctk.CTkButton(
            self,
            text="Результаты",
            command=self.next_page,
        )
        self.button_next.grid(row=4, column=3, padx=20, pady=20,
sticky="ew")

    def prev_page(self):
        sys.stdout = self.keep_stdout
        sys.stderr = self.keep_stderr
        self.controller.show_frame("ParametersPage")

    def next_page(self):
        self.controller.frames["ResultsPage"].set_args(
            model_name=self.kwargs["model_name"]
        )
        self.controller.show_frame("ResultsPage")
```

ResultsPage.py

```
PROJECT_ROOT = os.getenv("project_root")


class ResultsPage(ctk.CTkFrame):
    def __init__(self, controller):
        super().__init__(controller)
        self.controller = controller
        self.grid_rowconfigure((0, 1, 2, 3), weight=1)

        self.label = ctk.CTkLabel(
            self, text="Результаты обучения", fg_color="lightblue",
corner_radius=6
        )
        self.label.grid(row=0, column=0, padx=20, pady=20,
sticky="ew", columnspan=4)

        self.button_back = ctk.CTkButton(
            self,
            text="Назад",
            command=self.prev_page,
        )
        self.button_back.grid(row=1, column=0, padx=20, pady=10,
sticky="ew")

        self.frame_files = FilesFrame(self)
        self.frame_files.grid(
            row=2, column=0, padx=20, pady=20, sticky="ew",
columnspan=4
        )

        self.button_finish = ctk.CTkButton(
            self,
            text="Закрыть",
            command=self.controller.destroy,
        )
        self.button_finish.grid(row=3, column=1, padx=20, pady=10,
sticky="ew")

    def set_args(self, **kwargs):
        """Принимает с предыдущей страницы данные

        kwargs:
        model_name (str): имя, под которой сохранена модель
        """
        self.model_name = kwargs["model_name"]
        self.frame_files.init(self.model_name)

    def prev_page(self):
        for widget in self.frame_files.winfo_children():
            widget.destroy()
        self.controller.show_frame("TrainingPage")


class FilesFrame(ctk.CTkFrame):
    def __init__(self, parent):
        super().__init__(parent)

    def init(self, model_name: str):
        self.model_name = model_name
        path_reports = Path(PROJECT_ROOT, "reports",
self.model_name)
        self.labels_files = []
        self.buttons_open = []
        for i, path in enumerate(path_reports.iterdir()):
            label = ctk.CTkLabel(self, text=path.stem)
            label.grid(row=i, column=0, padx=20, pady=10, sticky="ew")
            button = ctk.CTkButton(
                self, text="Открыть", command=lambda path=path:
os.startfile(path)
            )
            button.grid(row=i, column=1, padx=20, pady=10,
sticky="ew")
            self.labels_files.append(label)
            self.buttons_open.append(button)
```

Приложение Б

make_dataset.py

```
def make_dataset(
    raw_dataset_dir_name: str,
    processed_dataset_dir_name: str,
    dimension: int,
    sampling_rate: int,
    leads: list[str],
    bar: Any = None,
):
    PROJECT_ROOT = Path(os.getenv("project_root"))
    PATH_RAW_DATA = Path(
        PROJECT_ROOT, "data", "raw", raw_dataset_dir_name
    ) # путь к каталогу с датасетом
    PATH_PROCESSED_DATA = Path(
        PROJECT_ROOT, "data", "processed", f"{dimension}D",
processed_dataset_dir_name
    ) # путь к каталогу с обработанным датасетом
    for part in ("train", "dev", "test"):
        Path(PATH_PROCESSED_DATA,
f"X_{part}").mkdir(parents=True, exist_ok=True)
        Path(PATH_PROCESSED_DATA,
f"labels_{part}").mkdir(parents=True, exist_ok=True)

    DEV_FOLD = 9
    TEST_FOLD = 10

    # load and convert annotation data (см. project/notebooks/1.0-vmr-
ptbxl-data-review.ipynb)
    Y = pd.read_csv(Path(PATH_RAW_DATA, "ptbxl_database.csv"),
index_col="ecg_id")
    Y.scp_codes = Y.scp_codes.apply(lambda x: ast.literal_eval(x))

    # Load scp_statements.csv for diagnostic aggregation
    scp_statements = pd.read_csv(Path(PATH_RAW_DATA,
"scp_statements.csv"), index_col=0)
    scp_statements = scp_statements[scp_statements.diagnostic == 1]

    # Добавляем поле diagnostic_class к каждой записи из БД.
diagnostic_class - это класс, обозначающий группу болезней.
    Y["diagnostic_class"] = Y.scp_codes.apply(
        lambda scp_codes: list(
            {
                scp_statements.loc[diagnostic].diagnostic_class
                for diagnostic in scp_codes
                if diagnostic in scp_statements.index
            }
        )
    )
    DIAGNOSTIC_CLASSES = list(
        set(scp_statements.diagnostic_class.values)
    ) # метки (классы болезней)
    LABELS_COUNT = (
        len(DIAGNOSTIC_CLASSES) - 1
    ) # количество меток (классов болезней) . Norm - все 0, поэтому
-1

    i_train, i_dev, i_test = 0, 0, 0
    n_MI, n_STTC, n_CD, n_HYP = (
        0,
        0,
        0,
        0,
    )
    total = len(Y) + 20
    current = 0
    filename = "filename_hr" if sampling_rate == 500 else
"filename_lr"
    for y in Y.itertuples():
        signal, meta = wfdb.rdsamp(Path(PATH_RAW_DATA, getattr(y,
filename)))
        signal = signal[:, [meta["sig_name"].index(lead) for lead in
leads]]

        if dimension == 1:
            signal = process1d(signal)
        elif dimension == 2:
            signal = process2d(signal)

        # Представим метки в виде вектора 0 и 1
        labels = np.zeros(LABELS_COUNT)
        for label in y.diagnostic_class:
            # labels[diagnostic_classes.index(label)] = 1  # one-hot
encoding
            if "MI" in label:
                labels[0] = 1
                n_MI += 1
            if "STTC" in label:
                labels[1] = 1
                n_STTC += 1
            if "CD" in label:
                labels[2] = 1
                n_CD += 1
            if "HYP" in label:
                labels[3] = 1
                n_HYP += 1
```

```
        # Разбиение на тренировочную, валидационную и тестовую выборки
        if y.strat_fold == DEV_FOLD:
            np.save(Path(PATH_PROCESSED_DATA, "X_dev", f"{i_dev}.npy"), signal)
            np.save(Path(PATH_PROCESSED_DATA, "labels_dev", f"{i_dev}.npy"), labels)
            i_dev += 1
        elif y.strat_fold == TEST_FOLD:
            np.save(Path(PATH_PROCESSED_DATA, "X_test", f"{i_test}.npy"), signal)
            np.save(Path(PATH_PROCESSED_DATA, "labels_test", f"{i_test}.npy"), labels)
            i_test += 1
        else:
            np.save(Path(PATH_PROCESSED_DATA, "X_train", f"{i_train}.npy"), signal)
            np.save(Path(PATH_PROCESSED_DATA, "labels_train", f"{i_train}.npy"), labels)
            i_train += 1

        current += 1
        if bar is not None:
            bar(current, total)

    # некоторые необходимые дополнительные данные
    save_meta = {
        "n_sig": 3,
        "fs": sampling_rate,
        "n_classes": LABELS_COUNT,
        "labels": {0: "MI", 1: "STTC", 2: "CD", 3: "HYP"},
        "n_MI": n_MI,
        "n_STTC": n_STTC,
        "n_CD": n_CD,
        "n_HYP": n_HYP,
        "n_train_dev_test": i_train + i_dev + i_test,
        "n_train": i_train,
        "n_dev": i_dev,
        "n_test": i_test,
    }

    with open(Path(PATH_PROCESSED_DATA, "meta.pkl"), "wb") as outfile:
        pickle.dump(save_meta, outfile)

    if bar is not None:
        bar(current + 20, total)
```

processing1d.py

```
def process1d(signal: np.ndarray) -> np.ndarray:
    """На одно отведение один столбец в массиве."""
    signal = butterworth_filter(signal)
    signal = znormolization(signal)

    return signal


band_pass_filter = butter(2, [1, 45], "bandpass", fs=100, output="sos")


def butterworth_filter(signal):
    """Band pass filter. Полосовой фильтр Баттерворта 2-ого порядка"""
    return sosfilt(band_pass_filter, signal, axis=0)


def znormolization(signal):
    """Z-score normalization"""
    return zscore(signal, axis=0)
```

processing2d.py

```
def process2d(signal: np.ndarray) -> np.ndarray:
    signal = butterworth_filter(signal)

    signal = minmax_normalization(signal)
    imgs = transform_to_image(signal)
    imgs = resize(imgs)

    return imgs


band_pass_filter = butter(2, [1, 45], "bandpass", fs=100, output="sos")


def butterworth_filter(signal):
    """Band pass filter. Полосовой фильтр Баттерворта 2-ого порядка"""
    return sosfilt(band_pass_filter, signal, axis=0)


def minmax_normalization(signal: np.ndarray) -> np.ndarray:
    return (signal - np.min(signal, axis=0)) / (
        np.max(signal, axis=0) - np.min(signal, axis=0)
    )


GAF = GramianAngularField(image_size=1000, method="summation")
RP = RecurrencePlot(threshold="distance", percentage=20)
MTF = MarkovTransitionField(image_size=1000, n_bins=20)
```

```
def transform_to_image(signal: np.ndarray) -> np.ndarray:
    # signal.shape() = (длина отведений = 1000 или 5000, кол-во отведений)
    # 3 изображения GAF, RP, MTF
    imgs = np.zeros((signal.shape[1] * 3, signal.shape[0], signal.shape[0]))

    i = 0
    for sig in signal.T:
        sig = sig.reshape(1, -1)
        imgs[i] = GAF.transform(sig)
        imgs[i + 1] = RP.transform(sig)
        imgs[i + 2] = MTF.transform(sig)
        i += 3
    return imgs


transform_resize = AlexNet_Weights.IMAGENET1K_V1.transforms()


def resize(imgs: np.ndarray) -> np.ndarray:
    return (
        transform_resize(torch.from_numpy(imgs.reshape(3, 3, 1000, 1000)))
        .numpy()
        .reshape(9, 224, 224)
    )
```

Приложение В

GRU.py

```
#
https://github.com/HemaxiN/DL_ECG_Classification/blob/main/gru.
py

class GRU(nn.Module):
  def __init__(
    self,
    input_size: int = 3,
    hidden_size: int = 200,
    num_layers: int = 2,
    dropout: float = 0.3,
    device="cuda",
  ):
    """
    Define the layers of the model
    Args:
      input_size (int): Кол-во входных признаков (1 на кол-во
отведений, участвующих в обучение)
      hidden_size (int): Кол-во скрытых нейронов
      num_layers (int): Кол-во слоев
      dropout (float): Вероятность отключения нейронов
    """
    super(GRU, self).__init__()
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.num_layers = num_layers
    self.dropout = dropout
    self.device = device
    self.bidirectional = 1  # 1 - не двунаправленный, 2 -
двунаправленный

    self.gru = nn.GRU(
      input_size,
      hidden_size,
      num_layers,
      dropout=dropout,
      batch_first=True,
      bidirectional=(self.bidirectional == 2),
    )

    # 4 - на число классов
    self.fc = nn.Linear(hidden_size * self.bidirectional, 4)

  def forward(self, X: torch.Tensor) -> torch.Tensor:
    """
    Forward Propagation

    Args:
      X: dimension (batch_size, 1000, 3)
    """
    # начальные состояние:
    h_0 = torch.zeros(
      self.num_layers * self.bidirectional,
      X.size(0),
      self.hidden_size,
    ).to(self.device)
    out_gru, _ = self.gru(X, h_0)
    # out_rnn shape: (batch_size, seq_length,
hidden_size*bidirectional) = (batch_size, 1000,
hidden_size*bidirectional)

    # last timestep
    out_gru = out_gru[:, -1, :]

    # out_rnn shape: (batch_size, hidden_size*bidirectional) - ready
to enter the fc layer
    out_fc = self.fc(out_gru)
    # out_fc shape: (batch_size, num_classes)

    return out_fc
```

BiGRU.py

```
#
https://github.com/HemaxiN/DL_ECG_Classification/blob/main/gru.
py

class BiGRU(nn.Module):
  def __init__(
    self,
    input_size: int = 3,
    hidden_size: int = 200,
    num_layers: int = 2,
    dropout: float = 0.3,
    device="cuda",
  ):
    """
    Args:
      input_size (int): Кол-во входных признаков (1 на кол-во
отведений, участвующих в обучение)
      hidden_size (int): Кол-во скрытых нейронов
      num_layers (int): Кол-во слоев
      dropout (float): Вероятность отключения нейронов
    """
    super(BiGRU, self).__init__()
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.num_layers = num_layers
    self.dropout = dropout
```

```
        self.device = device
        self.bidirectional = 2    # 1 - не двунаправленный, 2 - двунаправленный

        self.gru = nn.GRU(
            input_size,
            hidden_size,
            num_layers,
            dropout=dropout,
            batch_first=True,
            bidirectional=(self.bidirectional == 2),
        )

        # 4 - на число классов
        self.fc = nn.Linear(hidden_size * self.bidirectional, 4)

    def forward(self, X: torch.Tensor) -> torch.Tensor:
        """
        Args:
            X: размерность (batch_size, 1000, 3)
        """
        # начальные состояние:
        h_0 = torch.zeros(
            self.num_layers * self.bidirectional,
            X.size(0),
            self.hidden_size,
        ).to(self.device)
        out_gru, _ = self.gru(X, h_0)
        # out_rnn shape: (batch_size, seq_length, hidden_size*bidirectional) = (batch_size, 1000, hidden_size*bidirectional)

        # конкатенация выходов последнего временного шага из слоя "слева направо" и первого временного ряда слоя "справа налево"
        out_gru = torch.cat(
            (out_gru[:, -1, : self.hidden_size], out_gru[:, 0, self.hidden_size :]),
            dim=1,
        )

        # out_rnn shape: (batch_size, hidden_size*bidirectional) - ready to enter the fc layer
        out_fc = self.fc(out_gru)
        # out_fc shape: (batch_size, num_classes)
        return out_fc
```

AlexNet.py

```
# https://pytorch.org/vision/main/_modules/torchvision/models/alexnet.html#AlexNet_Weights

class AlexNet(nn.Module):
    def __init__(
        self, input_size: int = 9, num_classes: int = 4, dropout: float = 0.5
    ) -> None:
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(input_size, 96, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(96, 288, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(288, 576, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(576, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(384 * 6 * 6, 6144),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(6144, 6144),
            nn.ReLU(inplace=True),
            nn.Linear(6144, num_classes),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

Приложение Г

main_training.py

```
project_path = os.getenv("project_root")

def start_training(
    model_class: type[nn.Module],
    dimension: int,
    model_name: str,
    dataset_name: str,
    parameters: dict,
    model_parameters: dict = {},
):
    """Основная функция запуска обучения и тестирования модели.
    Включает в себя:

    Args:
        model_class (type[nn.Module]): класс модели
        dimension (int): размерность входных данных
        model_name (str): имя, под которым будет сохранена обученная модель
        dataset_name (str): имя датасета, из которого будут загружены данные
        parameters (dict): гиперпараметры обучения в виде словаря (epochs, batch_size, learning_rate, l2_decay, early_stop, optimizer: str = ["adam", "sgd"], device: str = ["cuda", "cpu", "mps"])
        model_parameters (dict): Параметры класса модели
    """

    # meta.keys() = (n_sig, fs, n_classes, labels, n_MI, n_STTC, n_CD, n_HYP, n_train_dev_test, n_train, n_dev, n_test)
    device = parameters["device"]
    meta = Path(
        project_path, "data", "processed", f"{dimension}D", dataset_name, "meta.pkl"
    )
    with open(meta, "rb") as f:
        meta = pickle.load(f)

    # установка устройства, на котором запускается обучение
    set_device(device)
    # установка seed на все модули рандома
    set_seed(2024)

    model = model_class(**model_parameters).to(device)
    dataset_train = MyDataset(dataset_name, dimension, "train", meta["n_train"])
    dataset_dev = MyDataset(dataset_name, dimension, "dev", meta["n_dev"])
    dataset_test = MyDataset(dataset_name, dimension, "test", meta["n_test"])

    dataloader_train = DataLoader(
        dataset_train, batch_size=parameters["batch_size"], shuffle=True
    )
    dataloader_dev = DataLoader(dataset_dev, batch_size=1, shuffle=False)
    dataloader_test = DataLoader(dataset_test, batch_size=1, shuffle=False)

    optimizer = (
        torch.optim.Adam if parameters["optimizer"] == "adam" else torch.optim.SGD
    )
    optimizer = optimizer(
        model.parameters(),
        lr=parameters["learning_rate"],
        weight_decay=parameters["l2_decay"],
    )

    # https://discuss.pytorch.org/t/weighted-binary-cross-entropy/51156/6
    # https://discuss.pytorch.org/t/multi-label-multi-class-class-imbalance/37573/2
    labels_weights = torch.tensor(
        [
            meta["n_train_dev_test"] / meta["n_MI"],
            meta["n_train_dev_test"] / meta["n_STTC"],
            meta["n_train_dev_test"] / meta["n_CD"],
            meta["n_train_dev_test"] / meta["n_HYP"],
        ],
        dtype=torch.float,
    )
    loss_function = nn.BCEWithLogitsLoss(pos_weight=labels_weights).to(device)
    # https://learnopencv.com/multi-label-image-classification-with-pytorch-image-tagging/
    # https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

    # эпохи в виде массива, для дальнейшего plot
    epochs = np.arange(1, parameters["epochs"] + 1)
    statistics = pd.DataFrame(
        data=np.full((parameters["epochs"], 4), -1.0),
        index=epochs,
```

```
        columns=("Training-loss", "Validation-loss", "Sensitivity",
"Specificity"),
    )
    # потери за каждый батч
    train_losses = np.zeros(int(np.ceil(meta["n_train"] /
dataloader_train.batch_size)))

    best_model_sensitivity = (model, 0)
    best_model_dev_loss = (model, float("inf"))
    patience_i = 0
    patience_limit = parameters["patience_limit"]
    for epoch in epochs:
        print(f"Epoch {epoch}")

        train_losses[:] = 0
        for i, (X, label) in enumerate(dataloader_train):
            loss = train_batch(X, label, model, optimizer, loss_function,
device)

            del X, label
            torch.cuda.empty_cache()

            train_losses[i] = loss

        train_mean_loss = train_losses.mean().item()
        dev_mean_loss = compute_dev_loss(
            model, dataloader_dev, loss_function, meta["n_dev"], device
        )
        statistics.at[epoch, "Training-loss"] = train_mean_loss
        statistics.at[epoch, "Validation-loss"] = dev_mean_loss
        quality_metrics = evaluate(
            model, dataloader_dev, meta["labels"].values(), device
        )
        statistics.at[epoch, "Sensitivity"] = quality_metrics.at["all",
"Sensitivity"]
        statistics.at[epoch, "Specificity"] = quality_metrics.at["all",
"Specificity"]

        print(f"Training-Loss: {train_mean_loss}")
        print(f"Validation-Loss: {dev_mean_loss}")
        print(f"Sensitivity: {quality_metrics.at["all","Sensitivity"]:.4f}")
        print(f"Specificity: {quality_metrics.at["all",
"Specificity"]:.4f}")

        if quality_metrics.at["all", "Specificity"] >
best_model_sensitivity[1]:
            best_model_sensitivity = (model, quality_metrics.at["all",
"Specificity"])
        if dev_mean_loss < best_model_dev_loss[1]:
            best_model_dev_loss = (model, dev_mean_loss)
            patience_i = 0
        else:
            patience_i += 1

        if patience_i == patience_limit:
            print("Early stopping")
            break

# Тестирование на тестовой выборке
test_quality_metrics = evaluate(
    best_model_dev_loss[0],
    dataloader_test,
    meta["labels"].values(),
    device,
)

# Сохраняем модель
path_save_model = Path(project_path, "models", model_name)
path_save_model.mkdir(parents=True, exist_ok=True)
torch.save(
    best_model_sensitivity[0].state_dict(),
    path_save_model / f"{model_name}_max_sens.pth",
)
torch.save(
    best_model_dev_loss[0].state_dict(),
    path_save_model / f"{model_name}_min_loss.pth",
)

# Сохраняем результаты
path_reports = Path(project_path, "reports", model_name)
path_reports.mkdir(parents=True, exist_ok=True)

statistics.to_html(
    path_reports / "training_report.html",
    index=True,
    col_space=100,
    float_format=lambda x: f"{x:.4f}",
    justify="left",
)

test_quality_metrics = percentage(test_quality_metrics)
test_quality_metrics.to_html(
    path_reports / "test_report.html",
    index=True,
    col_space=100,
    float_format=lambda x: f"{x}%",
    justify="left",
)
save_plot(
    statistics["Validation-loss"][statistics["Validation-loss"] > -0.9],
    "Validation-Loss",
    path_reports,
    "validation_loss",
)
```

```
    save_plot(
        statistics["Training-loss"][statistics["Training-loss"] > -0.9],
        "Training-Loss",
        path_reports,
        "training_loss",
    )
    save_plot(
        statistics["Sensitivity"][statistics["Sensitivity"] > -0.9],
        "Sensitivity",
        path_reports,
        "sensitivity",
    )
    save_plot(
        statistics["Specificity"][statistics["Specificity"] > -0.9],
        "Specificity",
        path_reports,
        "specificity",
    )

    # Сохраняем параметры запуска модели
    with open(path_reports / "parameters.txt", "w") as f:
        if not model_parameters:
            model_parameters = {
                k: value.default
                for k, value in
inspect.signature(model_class).parameters.items()
            }
        print(
            f"parameters\n{parameters}",
            f"model_parameters\n{model_parameters}",
            model_parameters,
            sep="\n\n",
            file=f,
        )


def train_batch(
    X: torch.Tensor,
    label: torch.Tensor,
    model: nn.Module,
    optimizer: torch.optim.Optimizer,
    loss_function: nn.BCEWithLogitsLoss,
    device: str,
) -> float:
    X, label = X.to(device), label.to(device)
    # Обнуляем градиенты всех параметров
    optimizer.zero_grad()
    # Вызов метода forward(). Прямой проход по сети
    out = model(X)
    # Вычисление функции потерь. criterion - функция потерь
    loss = loss_function(out, label)
    # Обратное распространение функции потерь. Вычисление
градиентов функции потерь
    loss.backward()
    # Обновление параметров оптимизатором на основе
вычисленных ранее градиентов
    optimizer.step()
    # Возвращаем значение функции потерь
    return loss.item()


def compute_dev_loss(
    model: nn.Module,
    dataloader: DataLoader,
    loss_function: nn.BCEWithLogitsLoss,
    n_dev: int,
    device: str,
) -> float:
    """Вычисление потерь на валидационном датасете

    Args:
        model (nn.Module): модель
        dataloader (DataLoader): датасет
        loss_function (nn.BCEWithLogitsLoss): функция потерь
        n_int (int): количество образцов в валидационном датасете
        device (str): устройство "cpu" или "cuda" или "mps"
    Returns:
        float: потери
    """
    model.eval()
    with torch.no_grad():
        dev_losses = np.zeros(int(np.ceil(n_dev /
dataloader.batch_size)))
        for i, (X, label) in enumerate(dataloader):
            X, label = X.to(device), label.to(device)
            y_pred = model(X)
            loss = loss_function(y_pred, label)
            dev_losses[i] = loss.item()

            del X, label
            torch.cuda.empty_cache()

    model.train()

    return np.mean(dev_losses)


def predict(model: nn.Module, X: torch.Tensor) -> np.ndarray:
    """Предсказание модели

    Args:
        model (nn.Module): модель
        X (torch.Tensor): (batch_size, ...)
```

```
    Returns:
        np.ndarray: (batch_size, n_classes)
    """
    # logits_ - логиты (ненормализованные вероятности) для каждого класса для каждого примера в пакете данных.
    logits = model(X)  # (batch_size, n_classes)
    # sigmoid - преобразование логитов в вероятности, т.е. в числа в диапазоне [0,1]
    probabilities = torch.sigmoid(logits).cpu()

    return (probabilities > 0.5).numpy()


def evaluate(
    model: nn.Module,
    dataloader: DataLoader,
    diagnostic_classes: tuple[str],
    device: str,
) -> pd.DataFrame:
    """Оценка качества модели

    Args:
        model (nn.Module): модель
        dataloader (DataLoader): dev_dataloader or test_dataloader
        diagnostic_classes (tuple[str]): классы диагнозов meta["labels"] = {0: "MI", 1: "STTC", 2: "CD", 3: "HYP"}
        device (str): "cpu" or "cuda" or "mps"

    Returns:
        pd.DataFrame:
        |     | TP | TN | FP | FN | Sensitivity | Specificity | G-mean |
        |------|----|----|----|----|-------------|-------------|--------|
        | MI  |  |  |  |  |  |  |  |
        | STTC |  |  |  |  |  |  |  |
        | CD  |  |  |  |  |  |  |  |
        | HYP |  |  |  |  |  |  |  |
        | all |  |  |  |  |  |  |  |
    """
    # Перевод модели в режим оценки
    model.eval()
    # Отключение вычисления градиентов
    with torch.no_grad():
        diagnostic_classes = tuple(diagnostic_classes) + ("all",)
        quality_metrics = pd.DataFrame(
            data=np.zeros((len(diagnostic_classes), 7)),
            index=diagnostic_classes,
            columns=("TP", "TN", "FP", "FN", "Sensitivity", "Specificity", "G-mean"),
        )
        for i, (X, label) in enumerate(dataloader):
            X, label = X.to(device), label.to(device)
            # прогноз модели
            y_predict = predict(model, X)
            # метки
            y_true = label.cpu().numpy()

            quality_metrics = compute_tptnfpfn(y_predict, y_true, quality_metrics)

            # Очистка памяти
            del X, label
            torch.cuda.empty_cache()
    quality_metrics = compute_all_tptnfpfn(quality_metrics)
    quality_metrics = compute_metrics(quality_metrics)
    # Возврат модели в режим обучения
    model.train()

    return quality_metrics
```

config_run.py

```
class DeviceError(Exception):
    pass

def set_seed(seed: int):
    """
    Конфигурация случайного генератора для воспроизводимости результатов обучения
    Устанавливает seed для всех используемых модулей:
    - random
    - numpy
    - torch
    - torch.cuda

    Также устанавливает флаг torch.backends.cudnn.deterministic = True также для воспроизводимости результатов обучения. В обычном режиме (при False) работы cuDNN использует оптимизированные алгоритмы, которые могут включать в себя некоторую степень неопределенности в вычислениях.
    """
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        torch.backends.cudnn.deterministic = True


def set_device(device: str):
    if device == "cpu":
        torch.device("cpu")
        return

    if device == "cuda" and torch.cuda.is_available():
```

```
    torch.device("cuda")
    return

  if device == "mps" and torch.backends.mps.is_available():
    torch.device("mps")
    return

  raise DeviceError(
    "The specified device is not available. Available devices: 'cpu', 'cuda', 'mps'"
  )
```

dataset.py

```
PROJECT_ROOT = os.getenv("project_root")


class MyDataset(Dataset):
  def __init__(self, dataset_name: str, dimension: int, part: str, n_samples: int):
    """Интерфейс датасета

    Args:
      dataset_name (str): название датасета
      dimension (int): размерность входных данных
      part (str): часть датасета (train, dev, test)
      n_samples (int): количество образцов
    """
    self.dataset_name = dataset_name
    self.dimension = dimension
    self.part = part
    self.n_samples = n_samples

    self.path_data_X_part = Path(
      PROJECT_ROOT,
      "data",
      "processed",
      f"{dimension}D",
      dataset_name,
      f"X_{part}",
    )
    self.path_data_labels_part = Path(
      PROJECT_ROOT,
      "data",
      "processed",
      f"{dimension}D",
      dataset_name,
      f"labels_{part}",
    )

  def __len__(self):
    return self.n_samples

  def __getitem__(self, idx):
    X = np.load(Path(self.path_data_X_part, f"{idx}.npy"))
    label = np.load(Path(self.path_data_labels_part, f"{idx}.npy"))

    return torch.tensor(X).float(), torch.tensor(label).float()
```

metrics.py

```
def compute_tptnfpfn(
  y_predict: np.ndarray, y_true: np.ndarray, quality_metrics: pd.DataFrame
):
  """Вычисление метрик TP | TN | FP | FN для каждого класса в метке.

  Args:
    y_predict (np.ndarray): прогноз модели (batch_size, num_classes)
    y_true (np.ndarray): метка (batch_size, num_classes)
    quality_metrics (pd.DataFrame): Таблица вида

    |     | TP | TN | FP | FN | Sensitivity | Specificity | G-mean |
    |------|----|----|----|----|-------------|-------------|--------|
    | MI  |  |  |  |  |          |          |      |
    | STTC |  |  |  |  |          |          |      |
    | CD  |  |  |  |  |          |          |      |
    | HYP |  |  |  |  |          |          |      |
    | all |  |  |  |  |          |          |      |
  Returns:
    pd.DataFrame: обновленный quality_metrics
  """
  for prediction, true in zip(y_predict, y_true):
    for i in range(len(prediction)):
      if prediction[i] == 1 and true[i] == 1:
        quality_metrics.iat[i, 0] += 1
      elif prediction[i] == 0 and true[i] == 0:
        quality_metrics.iat[i, 1] += 1
      elif prediction[i] == 1 and true[i] == 0:
        quality_metrics.iat[i, 2] += 1
      elif prediction[i] == 0 and true[i] == 1:
        quality_metrics.iat[i, 3] += 1

  return quality_metrics


def compute_all_tptnfpfn(quality_metrics: pd.DataFrame):
  """Вычисление метрик общего кол-ва TP | TN | FP | FN (строка 'all').

  Args:
    quality_metrics (pd.DataFrame): Таблица
  Returns:
    pd.DataFrame: обновленный quality_metrics
  """
  for column in quality_metrics.columns:
```

```
    quality_metrics.at["all", column] = quality_metrics[column].sum()

  return quality_metrics


def compute_metrics(quality_metrics: pd.DataFrame):
  """Вычисление метрик Sensitivity | Specificity | G-mean для каждого класса в метке.

  Args:
    quality_metrics (pd.DataFrame): Таблица
  Returns:
    pd.DataFrame: обновленный quality_metrics
  """
  quality_metrics["Sensitivity"] = quality_metrics["TP"] / (
    quality_metrics["TP"] + quality_metrics["FN"] + 0.0001
  )
  quality_metrics["Specificity"] = quality_metrics["TN"] / (
    quality_metrics["TN"] + quality_metrics["FP"] + 0.0001
  )
  quality_metrics["G-mean"] = np.sqrt(
    (quality_metrics["Sensitivity"] * quality_metrics["Specificity"])
  )

  return quality_metrics


def percentage(quality_metrics: pd.DataFrame):
  """Представление метрик в процентах

  Args:
    quality_metrics (pd.DataFrame):
  Args:
    quality_metrics (pd.DataFrame): Таблица
  Returns:
    pd.DataFrame: обновленный quality_metrics
  """
  quality_metrics["TP"] = np.round(
    quality_metrics["TP"] / quality_metrics.loc["all", "TP"] * 100, 2
  )
  quality_metrics["TN"] = np.round(
    quality_metrics["TN"] / quality_metrics.loc["all", "TN"] * 100, 2
  )
  quality_metrics["FP"] = np.round(
    quality_metrics["FP"] / quality_metrics.loc["all", "FP"] * 100, 2
  )
  quality_metrics["FN"] = np.round(
    quality_metrics["FN"] / quality_metrics.loc["all", "FN"] * 100, 2
  )
  quality_metrics["Sensitivity"] = np.round(quality_metrics["Sensitivity"] * 100, 2)
  quality_metrics["Specificity"] = np.round(quality_metrics["Specificity"] * 100, 2)
  quality_metrics["G-mean"] = np.round(quality_metrics["G-mean"] * 100, 2)

  return quality_metrics
```

plotting.py

```
def save_plot(x_y, ylabel, path, name):
  """сохранение графика

  Args:
    x_y: двумерный итерабельный объект
    ylabel (str): имя оси ордина
    path (str | Path): путь к папке для сохранения
    name (str): имя файла сохранения
  """
  plt.clf()
  plt.xlabel("Epoch")
  plt.ylabel(ylabel)
  plt.plot(x_y)
  plt.savefig(Path(path, f"{name}.png"), bbox_inches="tight")
```