МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Чорноморський національний університет
імені Петра Могили
Факультет комп'ютерних наук
Кафедра Інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ
Завідувач кафедри інтелектуальних інформаційних систем, д.т.н., проф.,
_____Ю.П. Кондратенко
«____»_____2022 року

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

ВИЯВЛЕННЯ СУПУТНІХ АРИТМІЙ НА ОСНОВІ ЕКГ У ХВОРИХ НА COVID – 19 З ВИКОРИСТАННЯМ МЕТОДІВ ІНТЕЛЕКТУАЛЬНОГО АНАЛІЗУ

Спеціальність 124 – «Системний аналіз»

124-МКР-607.21610407

Студент _____ С. О. Єгоров
«__» _____ 2022 р.

Консультант _____ А. І. Воробйова
канд. фіз.– мат. наук, доцент
«__» _____ 2022 р.

Миколаїв – 2022

Чорноморський національний університет ім. Петра Могили
Факультет комп'ютерних наук
Кафедра інтелектуальних інформаційних систем

Освітньо-кваліфікаційний рівень **магістр**
Галузь знань **12 «Інформаційні технології»**
(шифр і назва)
Спеціальність **124 «Системний аналіз»**
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри інтелектуальних інформаційних систем, д-р техн. наук, проф.
_____ Ю. П. Кондратенко
«____» _____20__р.

ЗАВДАННЯ
на магістерську кваліфікаційну роботу
ЄГОРОВУ Сергію Олександровичу

1. Тема магістерської кваліфікаційної роботи «Виявлення супутніх аритмій на основі ЕКГ у хворих на Covid – 19 з використанням методів інтелектуального аналізу.

Керівник роботи Воробйова Алла Іванівна, канд. фіз. – мат. наук, доцент.

Затв. наказом Ректора ЧНУ ім. Петра Могили від «20» _10_2022р. № _288__

2. Строк подання студентом роботи «___» ___ 2022р.

3. Вхідні (початкові) дані до роботи: для виконання роботи не потрібні початкові дані.

Очікуваний результат роботи: Проектування та розробка інформаційної системи виявлення супутніх аритмій на основі ЕКГ.

4. Зміст пояснювальної записки (перелік питань, які потрібно розглянути):

a) *теоретико-методологічна частина: провести аналіз нейронних мереж та сфер їх використання, також провести огляд парадигм навчання нейронних мереж*;

b) *дослідницько-аналітична частина: Провести огляд інструментальних рішень для створення власної програмної реалізації*;

c) *проектна частина*: Провести проектування і розробку власної програмної реалізації з використанням методології UML і обраної мови програмування;

5. Перелік графічного матеріалу

1) таблиці;

2) рисунки.

6. Завдання до спеціальної частини: Основні вимоги техніки безпеки і промислової санітарії.

7. Консультанти:

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис**
Спеціальна частина з охорони праці	Григор'єва Л. І., д. біологічних наук, професор	
Методична частина	Воробйова А. І., канд. фіз. – мат. наук	

Керівник роботи ___канд. фіз. – мат. наук, доцент, Воробйова А. І. __
(наук. ступінь, вчене звання, прізвище та ініціали)

_____
(підпис)

Завдання прийнято до виконання ____Єгоров С. О. _____
(прізвище та ініціали)

_____*(підпис)*

Дата видачі завдання «____» _____ 2021 р.

КАЛЕНДАРНИЙ ПЛАН
Виконання магістерської кваліфікаційної роботи

Тема: _Виявлення супутніх аритмій на основі ЕКГ у хворих на Covid – 19 з використанням методів інтелектуального аналізу.

№	Найменування роботи	Початок	Закінчення	Примітки
1	Визначення керівника і теми МКР. Подання заяви на затвердження теми МКР	01.09.2021	10.10.2021	
2	Отримання завдання на виконання МКР	19.10.2021	22.10.2021	
3	Складання календарного плану на період виконання МКР	23.10.2021	26.10.2021	
4	Огляд літератури за темою дослідження	27.10.2021	10.11.2021	
5	Проходження переддипломної практики, збір та аналіз матеріалів до МКР	29.11.2021	18.12.2021	
6	Аналіз предметної області та розробка технічного завдання. Моделювання результатів	23.12.2021	02.01.2022	
7	Опис фахової частини МКР	03.01.2022	25.01.2022	
8	Розробка спеціальної частини з охорони праці та методичної частини	26.01.2022	30.01.2022	
9	Попередній захист МКР на засіданні комісії кафедри	31.01.2022	31.01.2022	
10	Корегування роботи за результатами попереднього захисту	01.02.2022	03.02.2022	
11	Остаточне оформлення пояснювальної записки та слайдів доповіді для захисту	04.02.2022	08.02.2022	
12	Подання МКР рецензенту	11.02.2022	12.02.2022	
13	Рецензування МКР	13.02.2022	14.02.2022	
14	Подання МКР, її електронної копії та інших документів (відгуку, рецензії) до захисту	16.02.2022	17.02.2022	
15	Захист МКР перед екзаменаційною комісією (ЕК)	23.02.2022	24.02.2022	

Розробив студент _____Єгоров С. О._____ _____

(прізвище та ініціали) (*підпис*)

Керівник роботи _канд. фіз. – мат. наук, доцент, Воробйова А.І._____

(наук. ступінь, вчене звання, прізвище та ініціали) *(підпис)*

«____» _____ 2022р.

АНОТАЦІЯ

до магістерської кваліфікаційної роботи
студента групи 607 ЧНУ ім. Петра Могили

Єгоров Сергій Олександрович

на тему: **«Виявлення супутніх аритмій на основі ЕКГ у хворих на Covid – 19 з використанням методів інтелектуального аналізу»**

Актуальність. В сучасному світі будь-яка людина, яка зіштовхувалася з необхідністю викликати швидку допомогу, знає про необхідність зняття кардіограми для виключення проблем з сердцем. Але, при цьому, основна проблема залишається в тому, що не всі лікарі здатні "на око" виявити аритмію того, чи іншого виду, вірно її інтерпретувати та назначити відповідне лікування.

Саме з цього випливає необхідність створення програмного продукту, який буде на базі результатів кардіограми пацієнта давати точний результат стосовно наявності аритмії та її виду, для постановки більш точного діагнозу.

Об’єктом дослідження є методи і технології побудови нейронних мереж.

Предметом дослідження є обробка результатів ЕКГ методами інтелектуального аналізу з метою виявлення аритмій.

Мета магістерської роботи полягає в розробленні нейронної мережі для виявлення можливих аритмій і їх типів на базі аналізу результатів електрокардіограми.

У першому розділі було проведено аналіз предметної області і виявлено основні теоретичні аспекти, які в подальшому стануть основою для розуміння процесів, що протікають в середині предметної області, яке, в свою чергу, допоможе при проектуванні і розробці програмного забезпечення в межах предметної області. У другому розділі автором представлено огляд інструментів розробки інформаційної системи виявлення супутніх аритмій на основі ЕКГ у хворих на Covid-19. У третьому розділі було проведено аналіз варіантів використання системи, спроектовано внутрішню структуру системи, проведено проектування будови нейронної мережі, яка є основною складовою частиною системи. Окрім цього написано інструкцію користувача.

Сторінок – 91 , рисунків – 11 , посилань – 45 , додатків – 1.

Ключові слова: *нейронна мережа, аритмія, мова програмування python, діаграма класів, електрокардіограма*.

ABSTRACT

Master's scientific work
Student of group 607 Petro Mohyla Black Sea National University

Yehorov Serhii Alexandrovich

«Detection of concomitant arrhythmias according to ECG data in patients with Covid-19 using methods of intellectual analysis»

Relevance. In today's world, anyone who has had to call an ambulance knows about the need to take a cardiogram to rule out heart problems. However, the main problem remains that not all doctors are able to "eye" to detect arrhythmia of one kind or another, correctly interpret it and prescribe appropriate treatment.

This implies the need to create a software product that will be based on the results of the patient's cardiogram to give an accurate result regarding the presence of arrhythmia and its type, to make a more accurate diagnosis.

The object of research of this work are methods and technologies for building neural networks.

The subject of research is the processing of ECG results by methods of intellectual analysis to detect arrhythmias.

The purpose of the thesis is to develop a neural network to identify possible arrhythmias and their types based on the analysis of electrocardiogram results.

The first section analyzes the subject area and identifies the main theoretical aspects that will later be the basis for understanding the processes taking place within the subject area, which, in turn, will help in the design and development of software within the subject area. In the second section, the author presents an overview of tools for developing an information system for detecting concomitant arrhythmias based on ECG in patients with Covid-19. The third section analyzed the use of the system, designed the internal structure of the system, designed the structure of the neural network, which is the main component of the system. In addition, the user manual is written.

The work consists of an introduction, 3 chapters, conclusion, list of sources used (45 items). The work contains 36 drawings. The total volume is 91 pages, the main text of the work is set out on 44 pages.

Key words: *neural network, arrhythmia, Python programming language, class diagram, electrocardiogram*.

ЗМІСТ

Пояснювальна записка

до магістерської наукової роботи

на тему:

«Виявлення супутніх аритмій на основі ЕКГ у хворих на Covid – 19 з використанням методів інтелектуального аналізу»

Спеціальність 124 – Системний аналіз

124 – МКР – 607.21610407

Студент _____ Єгоров С. О.
«__» _____ 2022 р.

Керівник _____ Воробйова А.І
к. фіз-мат. н., доцент
«__» _____ 2022 р.

Миколаїв – 2022

ВСТУП

В сучасному світі, де панують тенденції на діджиталізацію та автоматизацію усіх процесів, явище нейронних мереж, які замінюють людину в деяких аспектах при прийнятті тих чи інших рішень — не новина.

Кожного дня створюються сотні нових нейронних мереж, кожна з яких виконує свою задачу, розпізнавання мови і жестів, розпізнавання образів, класифікація предметів, тощо.

Подібне явище не оминуло і одну з найважливіших галузей людського життя – медицину. Вже достатньо довгий час в медицині використовують різного роду додатки з використання нейронних мереж і штучного інтелекту, які допомагають медперсоналу виконувати ті чи інші дії, виключаючи так званий «людський фактор», тобто фактори, які можуть спричини помилку, притаманні виключно людині, такі як стрес, втома, різного роду недуги, емоційний стан, які впливають на концентрацію, увагу та можливість адекватно оцінювати ситуацію і приймати рішення.

Виходячи з описаної вище актуальності явища нейронних мереж в медицині, об’єктом дослідження було обрано використання нейронних мереж в сфері медицини, а саме кардіології.

В свою чергу, предметом дослідження став аналіз методів і засобів інтелектуального аналізу результатів електрокардіограми.

Мета роботи полягає в розробленні нейронної мережі для виявлення можливих аритмій і їх типів на базі аналізу результатів електрокардіограми.

Методи дослідження — аналіз літературних та інших джерел, комп’ютерне моделювання та проектування.

Для досягнення поставленої мети слід виконати наступні завдання:

1. Провести аналіз предметної області
   - Проаналізувати поняття нейронної мережі

- Проаналізувати поняття штучних нейронів
- Провести огляд парадигм навчання нейронних мереж
- Проаналізувати поняття моделі нейронної мережі
- Оглянути сфери використання нейронних мереж
- Розглянути поняття аритмії

2. Описати обрані засоби розробки
   - Провести огляд мови програмування Python
   - Провести огляд бібліотеки sckit-image
3. Реалізувати програмний засіб
   - Створити діаграму варіантів використання
   - Створити діаграму класів
   - Створити графічний інтерфейс користувача
   - Написати інструкцію користувача

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття нейронної мережі

Штучна нейронна мережа (ANN), яку зазвичай називають нейронною мережею (NN), є обчислювальною системою, натхненною біологічними нейронними мережами, які утворюють мозок тварин. [1]

ШНМ заснована на сукупності пов’язаних одиниць або вузлів, які називаються штучними нейронами, які можуть вільно моделювати нейрони біологічного мозку. Кожна сполука, як синапс у біологічному мозку, може передавати сигнали іншим нейронам. Штучний нейрон, який отримує сигнал, потім обробляє його і може посилати сигнали до підключених до нього нейронів. «Сигналом» у з’єднанні є дійсне число, а вихід кожного нейрона обчислюється деякою нелінійною функцією суми його вхідних даних. З’єднання називаються ребрами. Нейрони та ребра часто мають ваги, які модулюються під час навчання. Під час підключення вага збільшує або зменшує силу сигналу. Нейрон може мати поріг, тобто посилати сигнал лише тоді, коли агрегований сигнал перевищує цей поріг. Як правило, нейрони групуються в шари. Різні шари можуть виконувати різні перетворення на своїх входах. Сигнал переміщається від першого шару (вхідного шару) до останнього шару (вихідного шару), можливо, через кілька раундів шарів.

Уоррен Маккаллок і Уолтер Піттс (2) (1943) почали цю тему зі створення обчислювальних моделей нейронних мереж. [3] Наприкінці 1940-х років Д. О. Хебб [4] запропонував гіпотезу навчання, засновану на механізмі нейропластичності, пізніше відомому як навчання Хебба. Фарлі та Уеслі А. Кларк [5] (1954) вперше використали комп’ютер (пізніше названий «калькулятором») для моделювання мережі Хеббі. Розенблат [6] (1958) створив персептрон. [7] Першу багатошарову функціональну мережу опублікували Івахненко і Лапа в 1965 році як груповий метод обробки даних. [8] [9] [10] Основа безперервного

зворотного поширення [8] [11] [12] [13] була отримана в контексті теорії керування Келлі [14] у 1960 році та Брайсона [15] у 1961 році, використовуючи принципи динамічного програмування.

У 1970 р. Сеппо Ліннайнмаа опублікував загальний метод автоматичного диференціювання (AD) дискретних підключених мереж вкладених диференційованих функцій [16] [17]. У 1973 році Дрейфус використовував зворотне розповсюдження для адаптації параметрів контролерів пропорційно градієнтам помилок. [18] Алгоритм зворотного розповсюдження Вербоса (1975) дозволив практичне навчання багаторівневих мереж. У 1982 р. Він застосував метод AD Ліннайнмаа до нейронних мереж способом, який став широко застосовуватися [11] [19]. Потім дослідження застоювались після Мінські та Паперта (1969) [20], який виявив, що основні персептрони не здатні обробляти ексклюзив або схему і що комп'ютерам не вистачає потужності для обробки корисних нейронних мереж.

Розвиток дуже масштабної інтеграції (ОМС) метал-оксид-напівпровідник (МОП) у формі додаткової технології МОП (КМОП) дозволило збільшити кількість транзисторів МОП в цифровій електроніці. Це забезпечило більшу обробну потужність для розвитку практичних штучних нейронних мереж у 1980-х рр. [21]

У 1992 році було введено макс-пулінг, щоб допомогти з мінімальним зсувом незмінності та толерантності до деформації, щоб допомогти 3D-розпізнаванню об'єктів. [22] [23] [24] Шмідхубер прийняв багаторівневу ієрархію мереж (1992), попередньо підготовлену по одному рівню за допомогою безконтрольного навчання та доопрацьованої шляхом зворотного поширення.

Джеффрі Хінтон (2006) запропонував моделювати кожен шар за допомогою кінцевої машини Больцмана [26], використовуючи послідовні шари двійкових або реальних прихованих змінних для вивчення представлень високого рівня. У 2012 році була створена мережа, яка вчиться розпізнавати поняття вищого рівня, такі як кішки, просто дивлячись на зображення без міток [27]. Неконтрольована

попередня підготовка та збільшення обчислювальної потужності графічних процесорів і розподілених обчислень призвели до використання більших мереж, особливо для розпізнавання зображень та візуального, що стало відомим як «глибоке навчання». [28]

Ciresan та його колеги (2010) [29] показали, що, незважаючи на проблему зникаючих градієнтів, графічні процесори дозволяють зворотне поширення багатошарових нейронних мереж, що безпосередньо спілкуються. [30] З 2009 по 2012 роки ANN почали вигравати нагороди в конкурсах ANN, підходячи до різних завдань на рівні людини, спочатку в розпізнаванні образів і машинному навчанні. [31] [32] Наприклад, Грейвс та ін., Двонаправлена та багатовимірна довготривала короткочасна пам’ять (LSTM) [33] [34] [35] [36]. У 2009 році переміг у трьох суміжних конкурсах на розпізнавання рукописного тексту без будь-яких попередніх знань з трьох мов, які вивчаються [35] [34].

Ciresan та його колеги створили перший розпізнавач шаблонів, щоб досягти конкурентоспроможності людини/надлюдини на таких еталонах, як розпізнавання дорожніх знаків (IJCNN 2012) [37].

Нейронні мережі навчаються, обробляючи приклади, кожен з яких містить відомі «вхідні дані» та «результати», утворюючи між ними зважені за ймовірністю асоціації, які зберігаються у власних структурах даних мережі. Навчання нейронної мережі на даному прикладі зазвичай здійснюється шляхом визначення різниці між обробленим результатом мережі (зазвичай очікуваним) і цільовим результатом. Це помилка. Потім мережа коригує свої зважені асоціації відповідно до вивченого правила та з використанням цього значення помилки. Послідовні коригування змушують нейронну мережу давати результати, які все більше і більше схожі на цільові результати. Після достатньої кількості цих налаштувань навчання може бути припинено за певними критеріями. Це називається контрольованим навчанням.

Такі системи «вчаться» виконувати завдання, розглядаючи приклади, часто без необхідності писати певні правила для конкретних завдань. У розпізнаванні

зображень, наприклад, вони можуть навчитися ідентифікувати зображення, які містять кішок, аналізуючи приклади зображень, які вручну позначено як «кіт» або «без кішки», і використовуючи результати для ідентифікації котів на інших зображеннях. Вони роблять це без будь-яких попередніх знань про кішок, таких як їх шерсть, хвіст, вуса та обличчя кішки. Натомість вони автоматично генерують ідентифікаційні ознаки на основі прикладів, які вони обробляють.

1.2 Штучні нейрони як основна складова мережі

Штучні нейронні мережі складаються з штучних нейронів, концептуально похідних від біологічних нейронів. Кожен штучний нейрон має вхід і виробляє вихід, який можна надіслати кільком іншим нейронам. Вхідними даними можуть бути значення ознак зовнішніх зразків даних, таких як зображення чи документи, або вихідні дані інших нейронів. Кінцевий вихід нейронної мережі Виходи нейронів виконують такі завдання, як розпізнавання об'єктів у зображеннях.

Щоб знайти вихід нейрона, ми спочатку беремо зважену суму всіх вхідних даних, зважену вагами зв’язків входів з нейроном. До цієї суми ми додамо термін упередження. Цю зважену суму іноді називають активацією. Це зважування потім передається через (зазвичай нелінійну) функцію активації для отримання вихідних даних. Вихідними даними є зовнішні дані, такі як зображення та документи. Кінцевим результатом є такі завдання, як розпізнавання об’єктів на зображеннях. [40]

1.3 Парадигми навчання нейронних мереж

Існує дві парадигми навчання нейронних мереж – з учителем і без вчителя. У першому випадку, на вхідний вектор є готова відповідь, у другому випадку нейронна мережа самонавчається. У кожного виду навчання є своя ніша завдань і за великим рахунком вони не перетинаються. На даний момент придумано і запатентовано велику кількість архітектур нейронних мереж і методів їх навчання. Але основними (вихідними) є – навчання з учителем це «алгоритм зворотного поширення помилки», а навчання без учителя це алгоритми Хебба і Кохонена. Ці парадигми сильно перетинаються з біологічної дійсністю, наприклад - дитина навчається з учителем або без.

Також, останнім часом, сформувалася нова, третя парадигма - навчання з підкріпленням.

Навчання з вчителем

Ця категорія навчання має справу в парах X '; і Y ';, і завдання - відобразити їх у функції f: X → Y. Тут Y дана - вчитель, планові бажані виходи, і X дані - дані, які генерують Y дані. Це аналогічно вчителю, який навчає всіх виконувати певне завдання.

Одна виняткова особливість цієї парадигми навчання - пряме посилання на помилку, яка просто порівнює між плановим і поточним результатом. Параметри мережі йдуть на вхід вартісної функції, яка визначає кількість невідповідність між бажаним і поточним висновком.

Навчання з учителем - дуже придатна для задач, які заздалегідь надає патерн, мету досягнення. Приклади: класифікація картинок, розпізнавання голосу, функція апроксимації, прогнозування. Зверніть увагу, що нейромережу слід забезпечити попередніми знаннями парою вводити-незалежних значень (X) і висновком Классифікаці-залежних значень (Y). Присутність залежних висновків - обов'язкова умова для навчання з учителем.

Навчання без учителя

В навчанні без учителя ми маємо справу лише з даними без міток або класифікації; замість цього нейронна структура намагається намалювати логічні висновки і витягти знання з цього використовуючи тільки вхідні дані X.

Це аналогічно самонавчанню коли хтось навчає його / її завдяки його / її досвіду і встановлюючи побічні критерії. в навчанні без учителя ми не визначаємо бажаний патерн для кожного спостереження але нейронна структура може навчатися без будь-якого вчителя.

Тут, функція вартості (cost function) грає важливу роль. Вона сильно впливає на значення нейрона так само добре як і зв'язок між вхідними значеннями.

Приклади завдань, де використовується навчання без учителя: кластеризація, компресія даних, статистичні моделі і мовні моделі.

1.4 Модель нейронної мережі

Модель нейронної мережі описує її архітектуру і конфігурацію, а також використовувані алгоритми навчання.

Архітектура нейронної мережі визначає загальні принципи її побудови (плоскослоїста, полнозв'язна, слабозв'язаних, прямого поширення, реккурентна і т.д.).

Конфігурація конкретизує структуру мережі в рамках заданої архітектури: число нейронів, число входів і виходів мережі, які використовуються активаційні функції.

Розрізняють такі базові архітектури:

1. мережі прямого поширення - всі зв'язки направлені строго від вхідних нейронів до вихідних. До таких мереж відносяться, наприклад, персептрон Розенблатта і багатошаровий персептрон;
2. рекурентні нейронні мережі - сигнал з вихідних нейронів або нейронів прихованого шару частково передається назад на нейрони вхідного шару мережі;
3. мережі радіально-базисних функцій - мережі, що містять єдиний прихований шар, нейрони якого використовують радіально-симетричну активаційну функцію, застосовуються для вирішення задач класифікації та прогнозування;
4. мережі Кохонена - клас мереж, що використовують навчання без учителя і призначених для вирішення завдань кластеризації. Вони містять всього два прошарки: вхідний (розподільний) і вихідний (кластеризуються);
5. карти Кохонена або карти ознак, що самоорганізуються - різновид мереж Кохонена, в яких число вихідних нейронів вибирається багато більше числа формованих кластерів. Використовуються для візуалізації результатів кластеризації багатовимірних даних;

6. повнозв'язні мережі - нейронні мережі, в яких кожен нейрон пов'язаний з усіма іншими нейронами. Такі мережі мають найвищу щільність зв'язків;
7. слабозв'язані мережі - в них нейрони з'єднані тільки зі своїми найближчими сусідами;
8. нейронні мережі з плоскими шарами - в них нейрони утворюють каскади, звані шарами, при цьому нейрони кожного шару пов'язані з усіма нейронами наступного і попереднього шарів, а всередині шару зв'язків немає. Мережі з плоскими шарами можуть бути одношарові (містити один прихований шар) і багатошаровими (містити кілька прихованих шарів).

Кожна архітектура мережі призначення для вирішення певного класу задач аналізу даних (регресії, класифікації, кластеризації, прогнозування) і використовує спеціальні алгоритми навчання.

1.5 Використання нейронних мереж

До теперішнього часу найбільшого поширення набули такі види нейромереж:

- згорткові нейронні мережі (CNN) імітують роботу зорової кори головного мозку і частково виконують функцію абстрактного мислення. Вони прекрасно справляються із завданням розпізнавання зображень, а їх обчислення легко розпаралелити на графічних процесорах, що дозволяє створювати відносно дешеві апаратні платформи з елементами ШІ.

CNN застосовуються в системах машинного зору безпілотних автомобілів, комерційних дронів, роботів, а також в охоронному відеоспостереженні. Ви теж кожен день використовуєте CNN, якщо включили в налаштуваннях смартфона розблокування за допомогою розпізнавання особи.

- рекурентні нейронні мережі (RNN) володіють короткочасною пам'яттю, за рахунок чого легко аналізують послідовності довільної довжини. RNN розбивають потік даних на елементарні частини і оцінюють взаємозв'язки між ними. Ці алгоритми знайшли основне застосування в розпізнаванні рукописного тексту й мови. Коли ви шукаєте мелодію на слух в Shazam, розмовляєте з Siri, Google Now або Алісою від «Яндекса», залишаєте замітки від руки для Cortana - на хмарних платформах беруться за справу рекурентні нейромережі.
- мережі з довготривалою і короткочасною пам'яттю (LSTM) стали подальшим розвитком RNN. Вони гарні для прогнозування змін будь-якої величини (наприклад, біржових курсів або купівельного попиту) шляхом екстраполяції. Також їх застосовують для глибокого аналізу природної мови. Наприклад, Google використовує LSTM в персональному помічника і системі машинного перекладу Google Translate. Без LSTM якість перекладів так і залишалося б на рівні програм з дев'яностих, з якими часом було простіше перевести текст самому, ніж виправляти численні помилки.

- керовані рекурентні блоки (GRU) - порівняно недавня модифікація RNN, що з'явилася тільки в 2014 році. Їх використовують для синтезу мови, яка володіє емоційним забарвленням і звучить як справжня. Наприклад, в тестах сервісів Google Duplex і Microsoft XiaoIce люди не змогли відрізнити мовлення ботів від живих співрозмовників. Примітно, що XiaoIce дозволила Microsoft зміцнитися на азіатському ринку, де розвиток компанії завжди стримувалося мовним бар'єром.
- глибокі нейронні мережі (DNN) - будь-яка мережа більш ніж з трьома шарами. Вони лежать в основі механізмів глибокого машинного навчання, знаходячи неявні взаємозв'язку між різнорідними даними. Яскравий приклад - пошук кореляцій між розвитком захворювань і різними потенційними факторами у величезних масивах наукових статей за допомогою IBM Watson.
- генеративно-змагальні мережі (GAN). Це комбінація нейромереж, одна з яких генерує варіанти, а інша відсіває їх (виступає в якості арбітра). Таке поєднання дозволяє реалізувати машинне навчання без учителя, що підвищує автономність. Наприклад, PixelDTGAN генерує окремі зображення одягу, взуття та аксесуарів для каталогів онлайн-магазинів. В якості вхідних даних використовуються фотографії, на яких ці предмети гардероба демонструють фотомоделі. Поки зйомка для каталогів одягу вважається досить витратною частиною електронної комерції, але цілком можливо, що в найближчі роки нейромережі дозволять швидко проілюструвати каталог, навіть не залучаючи фотографа. Обробка фотографій теж забирає багато часу.

1.6 Класифікація аритмії для моделювання системи

Аритмії, також відомі як аритмії або аритмії, — це група станів, при яких серцебиття є нерегулярним, занадто швидким або надто повільним. [2] Прискорений пульс (понад 100 ударів на хвилину у дорослих) називається тахікардією, а повільний (менше 60 ударів на хвилину) називається брадикардією. [2] Деякі типи серцевих аритмій не мають симптомів. [1] Коли виникають симптоми, вони можуть включати серцебиття або паузи між серцебиттям. [1] У більш важких випадках може виникнути запаморочення, втрата свідомості, задишка або біль у грудях. [1] Хоча більшість типів аритмій не є серйозними, деякі з них схильні до ускладнень, таких як інсульт або серцева недостатність. [2] [3] Інші можуть викликати раптову смерть. [3] Існує чотири основні групи аритмії: зайві удари, надшлуночкові тахікардії, шлуночкові аритмії та брадиаритмії. [3] До додаткових ударів відносяться передчасне скорочення передсердь, передчасне скорочення шлуночків та передчасне стикання. [3] До суправентрикулярних тахікардій належать фібриляція передсердь, тремтіння передсердь та пароксизмальна суправентрикулярна тахікардія. [3] Шлуночкові аритмії включають фібриляцію шлуночків та шлуночкову тахікардію. [3] [7] Аритмії виникають через проблеми з системою електропровідності серця. [2] У дітей також можуть виникати аритмії; однак нормальний діапазон частоти серцевих скорочень різний і залежить від віку. [3] Ряд тестів може допомогти в діагностиці, включаючи електрокардіограму (ЕКГ) та монітор Холтера. [5]

Більшість аритмій можна ефективно лікувати. [2] Лікування може включати ліки, медичні процедури, такі як введення кардіостимулятора та хірургічне втручання [6]. Ліки для прискореного серцебиття можуть включати бета-блокатори або антиаритмічні засоби, такі як прокаїнамід, які намагаються відновити нормальний серцевий ритм. [6] Ця остання група може мати більш значні побічні ефекти, особливо якщо приймати їх протягом тривалого періоду часу [6]. Електрокардіостимулятори часто використовують для повільного

серцевого ритму. [6] Хворих з нерегулярним серцебиттям часто лікують розріджувачами крові, щоб зменшити ризик ускладнень. [6] Ті, хто має важкі симптоми аритмії, можуть отримати термінове лікування з контрольованим ураженням електричним струмом у вигляді кардіоверсії або дефібриляції. [6]

Аритмії вражають мільйони людей. [4] У Європі та Північній Америці станом на 2014 рік фібриляція передсердь вражає приблизно 2-3% населення. [8] Миготлива аритмія та фібриляція передсердь спричинили 112 000 смертей у 2013 році, порівняно з 29 000 у 1990 році. [9] Раптова серцева смерть є причиною приблизно половини смертей від серцево-судинних захворювань і приблизно 15% усіх смертей у всьому світі. [10] Приблизно 80% раптових серцевих смертей спричинені шлуночковими аритміями. [10] Аритмії можуть виникати в будь-якому віці, але частіше зустрічаються у літніх людей. [4]

Висновки до розділу 1

В ході написання першого розділу було проведено аналіз предметної області і виявлено основні теоретичні аспекти, які в подальшому стануть основою для розуміння процесів, що протікають в середині предметної області, яке, в свою чергу, допоможе при проектуванні і розробці програмного забезпечення в межах предметної області.

2 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Мова програмування Python

Python — це інтерпретована мова програмування загального рівня високого рівня. Філософія дизайну Python підкреслює читабельність коду завдяки використанню значних відступів. Його мовні конструкції та об'єктно-орієнтований підхід розроблені, щоб допомогти програмістам писати чистий логічний код для великих і великих проектів.

Python динамічно збирає сміття. Підтримуються декілька парадигм програмування, включаючи структурне (включаючи процедурне), об'єктно-орієнтоване та функціональне програмування. Оскільки Python має повну стандартну бібліотеку.

Наслідник мови програмування ABC, Гвідо ван Россум почав працювати над Python наприкінці 1980-х і вперше випустив мову Python 0.9.0 у 1991 році. [32] Python 2.0, випущений у 2000 році, представив нові функції, такі як розуміння списку та збір сміття за допомогою підрахованого ланцюжка, і був припинений у 2.7.18 у 2020 році. [33] Випущена в 2008 році, Python 3.0 була основною версією мови. Мова не є повністю зворотною сумісністю, і більшість коду Python 2 не працюватиме належним чином на Python 3.

Python завжди був однією з найпопулярніших мов програмування.

Python був задуманий наприкінці 1980-х років Гвідо ван Россумом з Centrum Wiskunde & Informatica (CWI) у Нідерландах. Він є наступником мови програмування ABC, натхненної ABC, яка обробляє винятки та взаємодіє з операційною системою Amoeba. Його реалізація почалася в грудні 1989 року. Як провідний розробник проекту, Ван Россум несе повну відповідальність за проект до 12 липня 2018 року. Це знайшло відображення в його довгій історії спільноти Python. Зобов'язання виконувати функції генерального менеджера проекту. Зараз він обіймає керівну посаду як член Наглядової ради з п'яти членів. Вибрано

активними розробниками ядра Python у січні 2019 року — Бретта Кеннона, Ніка Коглана, Баррі Варшаву, Керол Віллінг та Ван Россума членами комітету з п'яти осіб. З тих пір Гвідо ван Россум відкликав свою кандидатуру до ради 2020 року.

Python 2.0 був випущений 16 жовтня 2000 року з багатьма основними новими функціями, включаючи збірник сміття для виявлення циклів і підтримку Unicode.

Python 3.0 був випущений 3 грудня 2008 року. Це серйозна мовна цензура і не повністю сумісна з іншою стороною. Багато його основних функцій було перенесено на версії Python 2.6.x та 2.7.x. Дистрибутив Python 3 містить утиліту 2to3, яка автоматично (принаймні частково) перетворює код Python 2 на Python 3.

Python 2.7 спочатку був встановлений на 2015 рік, а потім перенесений на 2020 рік через занепокоєння, що більшу частину існуючого коду неможливо було легко перенести на Python 3. Більше жодних виправлень безпеки чи інших покращень не буде. Наприкінці свого життєвого циклу Python 2 підтримуватиме лише Python 3.6.x і пізніших версій.

Швидкість Python 3.9.2 і 3.8.8 була прискорена через проблеми безпеки з усіма версіями Python, що призвело до можливого віддаленого виконання коду та отруєння веб-кешу.

Python — це мова програмування з багатьма парадигмами. Об’єктно-орієнтоване програмування та структурне програмування повністю підтримуються, і багато його функцій підтримують функціональне та аспектно-орієнтоване програмування (включаючи метапрограмування та метаоб’єкти). Розширення підтримує багато інших парадигм, включаючи дизайн контрактів і логічне програмування.

Python використовує динамічне введення тексту та комбінацію підрахунку посилань і збирачів сміття. Збірник сміття визначає цикл управління пам'яттю. Він також має динамічне розділення імен (пізнє прив’язування) для зв’язування імен методів і змінних під час виконання програми.

Python був розроблений для забезпечення певної підтримки функціонального програмування на основі Lisp. Має можливості фільтрації, відображення та скорочення; перелічує розуміння генераторів, словників, наборів і виразів. Стандартна бібліотека має два модулі (itertools та functools) для реалізації функціональних інструментів, запозичених у Haskell та Standard ML.

Основні принципи мови викладені в документі Zen of Python (PEP 20), який включає такі сентенції:

- Красиво-краще, ніж потворно;
- Явне краще, ніж неявне;
- Просте - краще, ніж складне;
- Комплекс краще, ніж комплекс;
- Розрахунок читабельності.

Python не має всіх функцій, вбудованих у своє ядро, але розроблено, щоб бути дуже розширюваними (з модулями). Ця компактна модульність робить мову особливо популярною як спосіб додавання програмованих інтерфейсів до існуючих програм. Бачення Ван Россума щодо невеликої базової мови, великої стандартної бібліотеки та легко розширюваного перекладача випливало з його невдоволення азбукою, яка схиляла до протилежного підходу. Python прагне спростити та зменшити синтаксис і синтаксичну плутанину, дозволяючи розробникам вибирати свій метод кодування. На відміну від девізу Perl «Є кілька способів зробити це», Python наголошує на філософії дизайну «повинен бути один – бажано лише один очевидний спосіб». [69] Алекс Мартеллі, член Python Software Foundation і автор книги Python, пише: «Зображення чогось як «розумного» не вважається компліментом культурі Python».

Розробники Python намагаються уникнути передчасної оптимізації та відмовляються виправляти некритичні частини довідкової реалізації CPython. Ці виправлення пов’язані з різкістю, що призводить до незначного збільшення швидкості. Коли швидкість має значення, програмісти Python можуть перемістити

важливі за часом функції в розширення, написані на таких мовах, як C, або використовувати компілятор PyPy. Також доступний Cython, який перетворює скрипти Python на C і здійснює виклики API рівня C безпосередньо до інтерпретатора Python.

Важлива мета для розробників Python — продовжувати отримувати задоволення. Це знайшло відображення в назві мови (кивок британській комедійній компанії Monty Python), а іноді і в кумедних підручниках і довідкових методах, таких як стандартні foo і bar.

Користувачів і шанувальників Python, особливо тих, які вважаються обізнаними або досвідченими, часто називають Pythonists. Python має бути легкою для читання мовою. Його формат візуально не захаращений, і він часто використовує англійські ключові слова, тоді як інші мови використовують розділові знаки. На відміну від багатьох інших мов, він не використовує фігурні дужки для розділення блоків і дозволяє крапку з комою після операторів, але крапка з комою використовується рідко. Мова має менше синтаксичних винятків і особливих випадків, ніж C або Pascal.

Python використовує пробіли замість фігурних дужок або ключових слів для розділення блоків. Збільшення відступу відбувається після певних операторів; зменшення відступу означає кінець поточного блоку. Тому візуальна структура програми точно відображає семантичну структуру програми. Цю функцію іноді називають «зовнішнім» правилом, і в деяких інших мовах вона є, але в більшості мов відступ не має семантики. Рекомендований розмір відступу – чотири пробіли.

Оператори Python включають (серед іншого):

- Знак рівності = оператор присвоєння.
- Оператор if, що використовується разом із else та elif (скорочення від else-if) для умовного виконання блоку коду.
- for оператор, який фіксує кожен елемент у локальній змінній для використання вкладеними блоками для перегляду об’єкта, який можна повторювати.

- Оператор while виконує блок коду, поки умова істинна.
- оператор try, який дозволяє захоплювати та обробляти уривки, що містяться у вкладених блоках коду (крім речень); він також гарантує, що незалежно від того, як блок виходить, чистий код у блоці завжди буде виконуватися до кінця.
- Оператор boost використовується, щоб отримати вказаний виняток або повторно підняти перехоплений виняток.
- Оператор класу, який виконує блок коду та приєднує свій локальний простір імен до класу для об’єктно-орієнтованого програмування.
- Оператор Def, який визначає функцію або метод.
- Оператори Python 2.5, випущений у вересні 2006 р. [82], який охоплює фрагмент коду в Context Manager (наприклад, отримати блокування перед запуском блоку коду, потім зняти блокування або відкрити файл. Потім закрити), що дозволяє вам ініціалізуйте поведінку цього класу ресурсу. (RAII) і замініть звичайний try/final.
- Оператор break для виходу з циклу.
- Оператор continue пропускає цю ітерацію та переходить до наступного елемента.
- Оператор del видаляє змінну, тобто посилання на ім’я значення видаляється, і спроба використати змінну призведе до помилки. Ви можете перепризначити видалені змінні.
- Функція NOP виконується оператором. Синтаксично це потрібно для створення порожнього блоку коду.
- Заява про схвалення для перевірки умов, які застосовуються під час налагодження.
- Оператор Yield, який повертає значення з функції генератора. У Python 2.5 yield також є оператором. Ця форма використовується для реалізації спільного плану.

- Оператор return використовується для повернення значень із функцій.
- Оператор import використовується для імпорту модулів, функції чи змінні яких можна використовувати в поточній програмі. Існує три способи використання імпорту: імпорт <ім'я модуля> [як <псевдонім>] або <ім'я модуля> імпорт * або <ім'я модуля> для імпорту <визначення 1> [як <псевдонім 1>], <визначення 2 > [Як <псевдонім 2>], ...

Оператор присвоєння прив’язує ім’я як посилання на один динамічно розподілений об’єкт. Тоді ви можете відхилити змінну будь-якого об’єкта в будь-який час. У Python ім'я змінної є спільним власником посилання і не пов'язане з фіксованим типом даних. Однак на цьому етапі змінна посилається на об’єкт з типом. Це називається динамічним типом, на відміну від статичних мов програмування, де кожна змінна може містити значення лише певного типу.

Python не підтримує оптимізацію хвостових викликів або першокласні розширення, і, за словами Гвідо ван Россума, ніколи не буде. Однак у версії 2.5 була надана краща підтримка функціональності, подібної до корутина, розширюючи генератор Python. Ліниві ітератори мають максимум 2,5 генератора. Інформація передається від генератора в одному напрямку. З Python 2.5 ви можете передавати інформацію назад до функції генератора, а з Python 3.3 ви можете передавати інформацію на кілька рівнів стека.

Деякі вирази Python нагадують вирази на таких мовах, як C і Java, тоді як інші ні:

1. Додавання, віднімання і множення однакові, але ділення поводиться по-різному. У Python є два типи розділів. Вони є основним поділом (або цілочисельним поділом) // і з плаваючою комою/ поділом. Python також використовує оператор ** для просування.
2. У Python 3.5 було представлено новий оператор @infix. Бібліотеки, такі як NumPy, використовують його для множення матриць.

3. У Python 3.8 введено синтаксис: =, який називається «оператор моржа». Він призначає значення змінній як частину більшого виразу. [91]
4. == в Python порівнює за значенням з Java, Java порівнює значення за значенням і порівнює об'єкти за посиланням. (Порівняння значень об’єктів у Java можна використовувати метод equals().) Оператор is можна використовувати для порівняння ідентифікаторів об’єктів (ланцюжкові порівняння). У Python ви можете порівнювати такі порівняння, як <= b <= c.Python використовує слово і, або не для своїх булевих операторів, а не для символів &&, ||! Використовується в Java та C.Python має тип виразу, що називається розумінням списку, а також більш загальний вираз, який називається генераторським виразом.
5. Анонімні функції реалізовані з використанням лямбда-виразів, однак вони обмежені тим фактом, що тіло може бути лише виразом.
6. Якщо c інакше y, умовний вираз у Python буде записаний як x (відмінний від порядку операнда оператора c? X: y), що є однаковим у багатьох інших мовах.
7. Python розрізняє списки та кортежі. Список записаний як [1, 2, 3], він був змінений і не може використовуватися як ключ словника (ключ словника повинен бути постійним у Python). Кортеж записується як (1, 2, 3) і є незмінним, тому, поки всі елементи кортежу є незмінними, він може використовуватися як ключ словника. Оператор + може використовуватися для об'єднання двох кортежів, що безпосередньо не змінює їх вміст, але створює новий кортеж, що містить елементи двох передбачених кортежів. Отже, враховуючи, що змінна t спочатку дорівнює (1,2,3), коли виконується t = t + (4, 5), спочатку обчисліть t + (4, 5), щоб отримати (1,2, 3) . 3, 4, 5), а потім призначити його назад до t, тим самим ефективно "змінюючи вміст t", одночасно відповідаючи інваріантній природі об'єкта кортежу. У чіткому контексті кортежі не потребують дужок.

8. Python має функцію декомпресії послідовності, в якій кілька виразів (кожний вираз обчислює все, що можна призначити) (змінні, атрибути запису тощо) мають ту ж форму, що й текст, що складає кортеж, зазвичай розміщується зліва . Позови про рівність володіння. Оператор очікує ітерації праворуч від знака рівності. Під час сортування об’єкт виводить ту саму кількість значень, що й даний вираз, і фільтрує їх, призначаючи кожне створене значення лівій частині відповідного виразу.
9. Python має оператор % «формат рядка». Це схоже на рядок printf в C, наприклад, "спам = % s egg = % d" % ("blah", 2) оцінюється як "спам = blah-eggs = 2". У Python 3 та 2.6+ формат методу str() () доповнює це, наприклад, "spam={0}egg={1}". format("дурниця", 2). Python 3.6 додав "f-рядки": blah = "blah"; eggs=2; f'spam = {blah} eggs = {eggs}.Рядки в Python можна поєднувати шляхом "додавання" (те саме твердження, що і додавання цілих чи значень з плаваючою комою). Наприклад, "спам" + "яйце" повертає "спамера". Навіть якщо ваш рядок містить цифри, вони все одно будуть додані як рядки замість цілих чисел. Наприклад, "2" + "2" повертає "22".
10. Python має різні типи рядкових літералів:
    - Рядки, узяті в одинарні або подвійні лапки. На відміну від оболонки Unix, мова Perl і одинарні та подвійні лапки працюють однаково. Обидва типи рядків використовують зворотну косу риску (\) як вихідний символ. Інтерполяція рядка стала «відформатованим рядковим літералом» у Python 3.6. Рядок у подвійних лапках починається і закінчується серією з трьох одинарних або подвійних лапок. Вони можуть охоплювати кілька рядків і функціонувати подібно до документації оболонки, Perl і Ruby.
    - Початковий рядок позначається префіксом рядкового літералу, що передує r. Escape-послідовності не інтерпретуються, тому необроблені рядки корисні там, де поширені зворотні косі риски, наприклад регулярні вирази та шляхи в стилі Windows. Порівняйте "@-citation" в C#.

11. Python має індекси масивів та вирази масивів у списках, позначених як [ключ], [старт: зупинка] або [старт: зупинка: крок]. Індекс базується на нулі та є від'ємним відносно кінця. Нарізання переміщує елементи від початкового індексу до (але не враховуючи) індексу зупинки. Третій варіант зрізу називається кроком або висотою, що дозволяє пропускати та обертати елементи. Індекс зрізу можна опустити, наприклад [:] повертає копію всього списку. Кожен елемент зрізу є неглибокою копією.

Python чітко розрізняє вирази та вирази, які відрізняються від мов Common Lisp, Scheme або Ruby. Це призводить до дублювання певних функцій.

Python зазвичай використовується в проектах штучного інтелекту та машинного навчання з такими бібліотеками, як TensorFlow, Keras, Pytorch і Scikit-learn. Як мова сценаріїв із модульною архітектурою, простим синтаксисом і розширеними інструментами обробки текстів, Python часто використовується для обробки природних мов. Python успішно впроваджено в програмні продукти багатьма мовами сценаріїв, включаючи програмне забезпечення кінцевих елементів, таке як Abaqus, 3D-параметричні моделювання, такі як FreeCAD, 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, MotionBuilder, Softimage, Nuke visual ефекти Composer, програми для двовимірної графіки (такі як GIMP, Inkscape, Scribus і Paint Shop Pro), а також програми для створення нотаток (наприклад, Author and Group). Налагоджувач GNU використовує Python як хороший принтер для відображення складних структур, таких як контейнери C ++.

Esri вважає Python найкращим вибором для написання сценаріїв в ArcGIS. Він також використовується в кількох відеоіграх і є першою з трьох доступних мов програмування, які використовує Google App Engine.

Багато операційних систем використовують Python як стандартний компонент. Він постачається з більшістю дистрибутивів Linux, AmigaOS 4 (з використанням Python 2.7), FreeBSD (як пакет), NetBSD, OpenBSD (як пакет) і

macOS, і може використовуватися з командного рядка (термінал). У багатьох дистрибутивах Linux використовуються інсталятори, написані на Python: Ubuntu використовує інсталятор Ubiquity, а Red Hat Linux і Fedora — інсталятор Anaconda. Gentoo Linux використовує Python у своїй системі керування пакетами Portage.

Python широко використовується для захисту інформації, включаючи розробку експлойтів.

Більшість програм Sugar Laptop XO, які зараз розробляє Sugar Labs, написані на Python. Проект одноплатного комп'ютера Raspberry Pi використовує Python як свою основну мову програмування.

LibreOffice включає Python і має намір замінити Java на Python. Його постачальник сценаріїв Python був основною функцією у версії 4.0 7 лютого 2013 року.

Виходячи з діапазону та потужних можливостей мови Python, описаних вище, можна зробити висновок, що він є універсальним. Тому для виконання завдання була обрана ця мова програмування.

2.2 Sckit-image

Scikit-image (раніше scikits.image) - це бібліотека зображень з відкритим кодом для мови програмування Python. [2] Вона включає алгоритми сегментації, геометричного перетворення, маніпулювання кольоровим простором, аналізу, фільтрації, морфології, виявлення особливостей тощо. Вона призначена для взаємодії з числовою та науковою бібліотеками Python NumPy та SciPy.

Проект scikit-image розпочався на scikits.image, а його автором був Стефан ван дер Вальт. Його назва походить від ідеї, що це "SciKit" (набір інструментів SciPy), розроблений та розповсюджений окремо стороннім розширенням SciPy. [4] Оригінальна база коду пізніше була широко переписана іншими розробниками. У листопаді 2012 року в різних наукових роботах наукові знання та наукові знання були описані як «ретельно модифіковані та популярні» [5]. Scikit-image також дуже активний у Google Summer of Code. [6]

Scikit-image в основному написаний на Python, а деякі основні алгоритми написані на Cython для підвищення продуктивності.

Висновки до розділу 2

Другий розділ роботи присвячено огляду інструментів розробки інформаційної системи виявлення супутніх аритмій на основі ЕКГ у хворих на Covid-19.

Результатом написання даного розділу є розуміння основних механік взаємодії з обраною мовою програмування і супутніми інструментами.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Діаграма варіантів використання

Найпростіша діаграма використання — це уявлення про взаємодію користувача з системою, яка показує взаємозв’язок між користувачем і різними видами використання, в яких бере участь користувач. Діаграми варіантів використання визначають різні типи користувачів системи та різні варіанти використання, і часто супроводжуються діаграмами інших типів. Мета зображується колом або овалом.

У той час як самі варіанти використання можуть детально досліджувати кожну можливість, діаграми варіантів використання можуть допомогти надати огляд системи більш високого рівня.

Завдяки своїй спрощеній природі сценарії використання можуть бути хорошим інструментом комунікації для зацікавлених сторін. Ці малюнки намагаються імітувати реальний світ, даючи зацікавленим сторонам уявлення про те, як буде розвиватися система. Було проведено дослідження, щоб визначити, чи є реальний випадок використання програми. Було виявлено, що діаграми варіантів використання більш спрощено повідомляють про наміри системи зацікавленим сторонам, і вони були «пояснені більш повно, ніж діаграми класів».

Метою використання діаграм є показати динамічні аспекти системи. Для повного функціонального та технічного представлення системи доступні додаткові схеми та документація. Вони забезпечують спрощене та графічне представлення того, що насправді має робити система.

проект:

- рамки системи (англ. system border) - прямокутник із назвою у верхніх частинах та еліпсами (прецедентами) всередині,
- актор (англ. актор) - стилізований людський персонаж, обозначає набір ролей користувача (розуміється в широкому змісті: людина, зовнішня

сутність, клас, інша система), взаємодіючого з деякою сутністю (системною, підсистемою, класом). Актори не можуть бути пов'язані між собою з іншим (за вимкнення відносин щодо обробки / дослідження),

– прецедент - еліпс із надписом, що обозначає виконувану систематичну дію (може включати можливі варіанти), що призводить до спостерігаємих акторами результатів. Надпис може бути ім'ям або описом (з точки зору актора) того, "що" робить система (а не "як"). Ім'я прецедента зв’язано з неперервним (атомарним) сценарієм - конкретною послідовністю дій. Під час сценарію актори обмінюються із систематичними повідомленнями. Сценарій може бути приведений на діаграмі прецедентів у відео UML-коментарі. З одним прецедентом може бути пов'язано кілька різних сценаріїв

На рисунку 3.1 зображено діаграму варіантів використання, яка описує можливі дії користувача в системі.

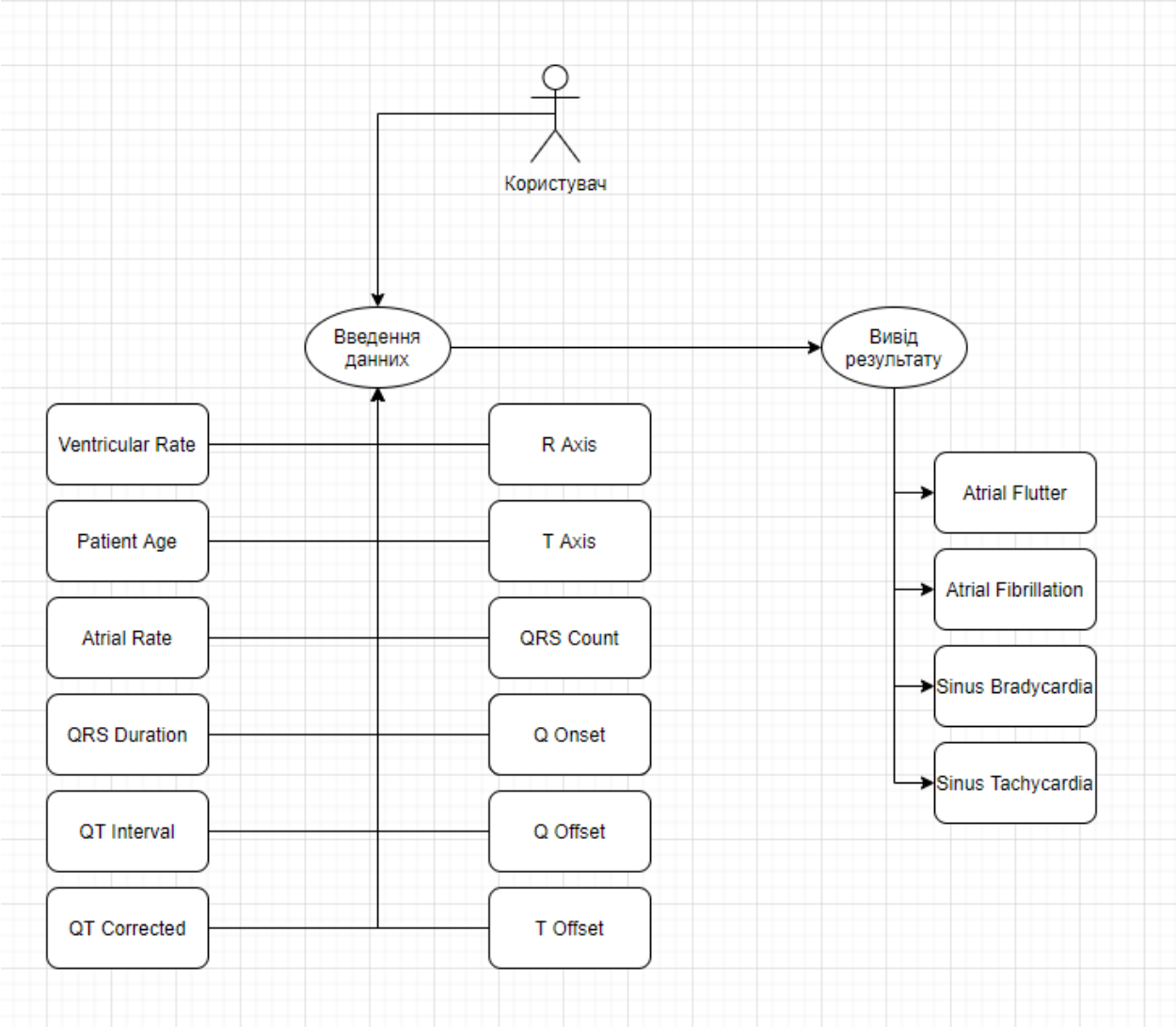

Рис. 3.1. Діаграма варіантів використання

3.2 Діаграма класів програмного продукту

У програмній інженерії діаграма класів уніфікованої мови моделювання (UML) — це статична структурна діаграма, яка описує структуру системи та показує системні класи, їх властивості, операції (або методи) і зв’язки між об’єктами.

Діаграми класів є основними будівельними блоками об'єктно-орієнтованого моделювання. Вони використовуються для загального концептуального моделювання структури програми та детального моделювання для перетворення моделей у програмний код. Діаграми класів також можна використовувати для моделювання даних. Класи на діаграмі класів представляють основні елементи, взаємодію та класи програмування в програмі.

На схемі ці класи представлені вікнами, що містять три відділення:

- У верхньому відділенні міститься назва класу. Надруковано жирним шрифтом і відцентровано, а перша літера написана великими літерами.
- Середній відсік містить атрибути класу. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні містяться операції, які може виконувати клас. Вони також вирівняні за лівим краєм, а перша буква - мала.

При проектуванні системи багато класів ідентифікуються і групуються в схему класів, що допомагає визначити статичні відносини між ними. При детальному моделюванні класи концептуального проектування часто поділяють на підкласи.

Залежність — це семантичний зв’язок між залежними елементами моделі та незалежними елементами. Він існує між двома елементами, якщо зміна, визначена одним елементом (сервером або цільовим), може спричинити зміну іншого елемента (клієнта чи джерела). Це об’єднання одностороннє. Залежності показані

пунктирними лініями з порожнистими стрілками, що вказують від клієнтів до постачальників.

Для подальшого опису поведінки системи ці діаграми класів можуть бути доповнені діаграмами станів або автоматами станів UML.

Асоціація являє собою серію посилань. Бінарні асоціації (з двома кінцями) зазвичай представляють у вигляді рядків. Асоціація може пов'язувати будь-яку кількість класів. Асоціація з трьома ланками називається потрійною асоціацією. Асоціації можна назвати, а кінці асоціації можна прикрасити іменами ролей, індикаторами власності, множинністю, видимістю та іншими атрибутами.

Є чотири різні типи асоціацій: двонаправлені, односпрямовані, агрегатні (включаючи складені агрегати) і рефлексивні. Найбільш поширеними є двонаправлені та односпрямовані асоціації.

Наприклад, категорія польоту пов’язана з категорією літака з двостороннім рухом. Асоціація являє собою статичні відносини, спільні між об'єктами двох класів.

Агрегації є різновидом відношення «has»; агрегації є більш конкретними, ніж асоціації. Це асоціація, яка представляє частину мети або частину відносин. Як показано, у професора «має» клас викладати. Як тип асоціації, агрегати можуть бути названі й мати те саме прикраси, що й асоціації. Однак агрегат не може включати більше двох класів; це має бути бінарна асоціація. Крім того, у процесі впровадження існує невелика різниця між агрегацією та асоціацією, і графік може повністю опустити зв’язок агрегації. [7]

Агрегація може відбуватися, коли клас є колекцією або контейнером інших класів, але міститься клас не має сильної залежності від часу життя контейнера. Коли контейнер знищено, вміст контейнера все ще існує.

В UML він представлений графічно у вигляді порожнистого ромба на класі хоста, з'єднаного з класом хоста лінією. Агрегат є семантично розширеним об’єктом і вважається одиницею в багатьох операціях, навіть якщо він фізично складається з кількох менших об’єктів.

Приклад: бібліотека та студенти. Тут студенти можуть існувати без бібліотеки, а зв’язок між студентами та бібліотекою є сукупністю.

Це показує, що один із двох споріднених класів (підкласів) вважається особливою формою іншого (суперкласу), а суперклас є узагальненням підкласу. На практиці це означає, що будь-який екземпляр підтипу також є екземпляром суперкласу. Типове дерево узагальнень цієї форми можна знайти в біологічних класифікаціях: людина — підклас людиноподібних мавп, людиноподібні мавпи — підклас ссавців тощо. Найкраще цей зв’язок можна зрозуміти за допомогою фрази «А є Б» (люди — ссавці, ссавці — тварини).

Графічне представлення узагальнення UML — це форма порожнистого трикутника в кінці суперкласу рядка (або дерева рядків), який з’єднує його з одним або кількома підтипами.

Відносини узагальнення також відомі як відносини успадкування або відносини «є».

Суперклас (базовий клас) у відносинах узагальнення також називають «батьківським класом», суперкласом, базовим класом або базовим класом.

Підтип у відносинах спеціалізації також називається "дочірнім" підкласом, похідним класом, похідним типом, успадкованим класом або успадкованим типом.

Зауважте, що ці відносини зовсім не схожі на біологічні відносини між батьком і дитиною: використання цих термінів дуже поширене, але воно може ввести в оману.

А є типом B

Наприклад, «дуб — це дерево», «автомобіль — транспортний засіб»

Узагальнення можуть бути показані лише в діаграмах класів і діаграмах використання.

У моделюванні UML відносини реалізації — це відносини між двома елементами моделі, де один елемент моделі (клієнт) реалізує (реалізує або виконує) поведінку, задану іншим елементом моделі (постачальником).

Графічне представлення реалізації UML — це порожнистий трикутник на кінці штрихового інтерфейсу (або дерева ліній), який з'єднує його з одним або кількома реалізаторами. Проста стрілка використовується в кінці інтерфейсу з пунктирами, щоб підключити його до користувача. У діаграмах компонентів використовується графічна умова «м'яч і розетка» (реалізатор розміщує кульку або льодяник, а користувач відображає розетку).

Реалізації можна показати лише на діаграмах класів або компонентів. Реалізації – це зв'язки між класами, інтерфейсами, компонентами та пакетами, які з'єднують елементи замовника з елементами постачальника. Відношення реалізації між класом/компонентом та інтерфейсом вказує на те, що клас/компонент реалізує операції, запропоновані інтерфейсом.

Залежність — це слабкіша форма спілкування, яка вказує на те, що один клас залежить від іншого класу, оскільки він використовує його в певний момент часу. Клас залежить від іншого класу, якщо незалежний клас є змінною параметра або локальною змінною методу залежного класу. Це відрізняється від асоціації властивостей залежних класів, які є екземплярами незалежних класів. Іноді відносини між двома класами слабкі. Вони взагалі не реалізуються зі змінними-членами. Натомість вони можуть бути реалізовані як аргументи для функцій-членів.

Ці відносини часто описують як «А має Б» (мама з кошенятами, кошенята з мамою).

Представлення асоціації UML - це лінія, що з'єднує два пов'язані класи.

Кожен кінець рядка має інші символи. Наприклад, ми можемо використовувати загострені стрілки для вказівки. Ми можемо представити право власності, помістивши кулю, роль, яку відіграє елемент на цьому кінці, вказавши ім'я ролі та кілька екземплярів цієї сутності (з іншого боку, пов'язуючи область задіяних об'єктів).

Класи сутностей моделюють довгострокову інформацію, яку обробляє система, а іноді і поведінку, пов’язану з цією інформацією. Їх не слід ідентифікувати як таблиці бази даних або інші сховища даних.

Вони намальовані у вигляді кіл з короткою лінією, прикріпленою до нижньої частини кола. Крім того, їх можна намалювати як звичайні класи з «основним» стереотипом у назві класу.

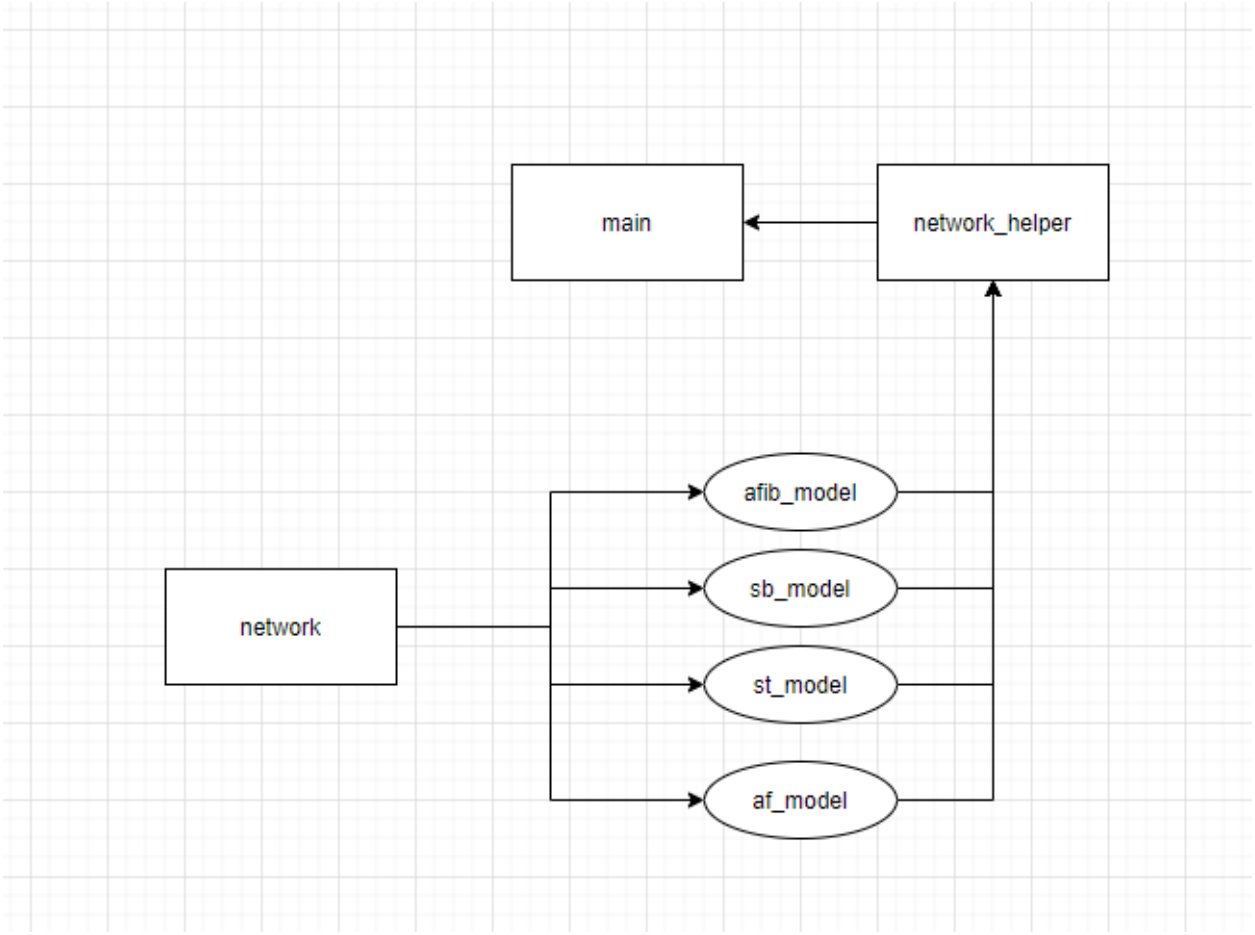

Рис. 3.2. Діаграма класів

3.3 Структура нейронної мережі

Розробка нейронної мережі, фактично, представляє собою реалізацію бібліотечного класу DescisionTreeClasifier, тобто дерева прийняття рішень, яке класифікує отримані дані завдяки створеному під час навчання алгоритму.

Нейронна мережа навчалася на датасеті даних щодо кардіограм, а саме наборах критеріїв результатів кардіограм.

Нейронна мережа складається з 4-х окремих моделей, кожна з яких навчалася на окремому датасеті. Кожна з моделей націлена на визначення вірогідності наявності одного з чотирьох типів аритмії у людини, якій належить кардіограма.

Моделі навчалися на датасетах с 4-ма видами кардіограм, а саме:

- Миготлива аритмія (рис. 3.3)
- Тріпотіння передсердь (рис. 3.4)
- Синусова брадикардія (рис. 3.5)
- Синусова тахікардія (рис. 3.6)

Рис. 3.3. Миготлива аритмія на ЕКГ

Рис. 3.4. Фібриляція (тріпотіння) передсердь на ЕКГ

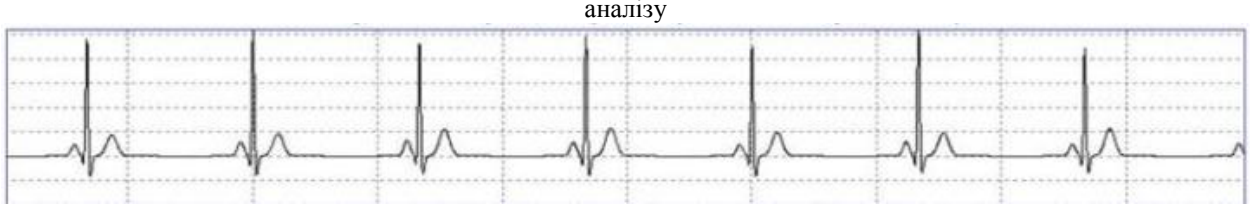

Рис. 3.5. Синусова брадикардія на ЕКГ

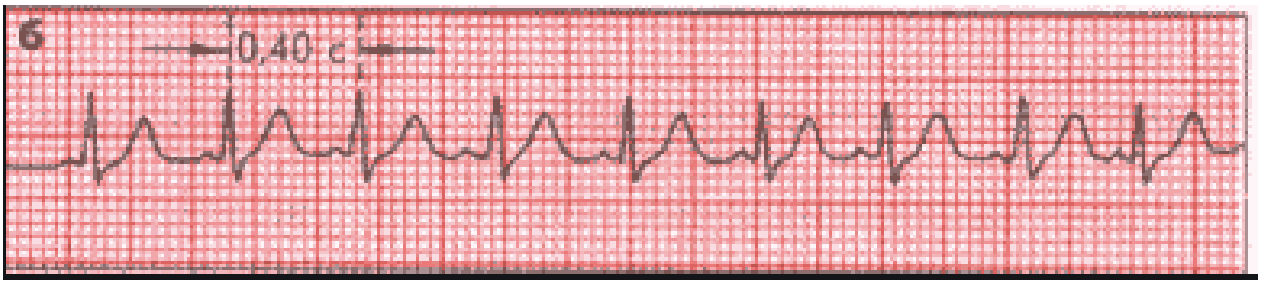

Рис. 3.6. Синусова тахікардія на ЕКГ

Кожний тип аритмії має свої унікальні характеристики, наприклад, кількість QRS-комплексів. Графік розподілу кількості QRS-комплексів зображено на рисунках 3.7 — 3.10.

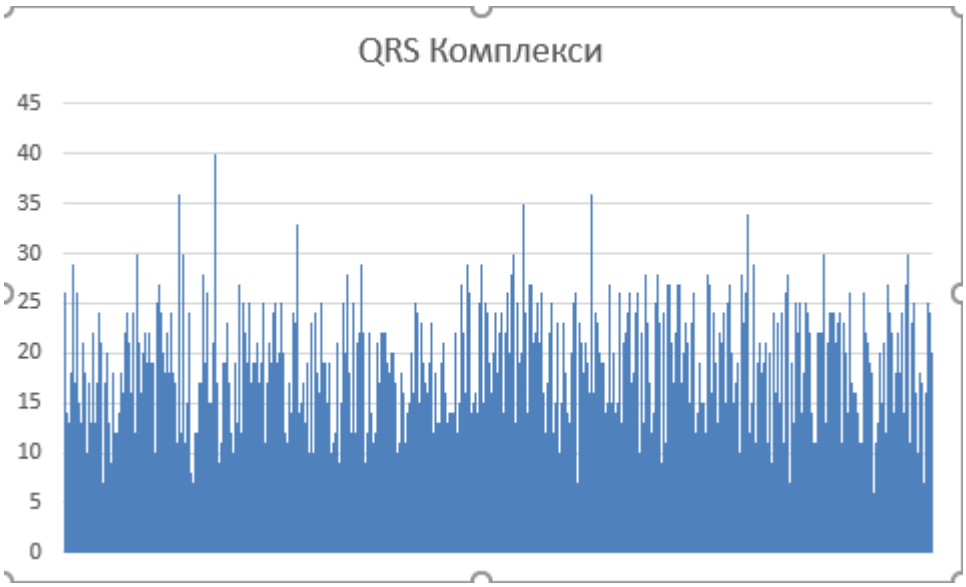

Рис. 3.7. Графік розподілу кількості QRS-комплексів для фібриляції передсердь

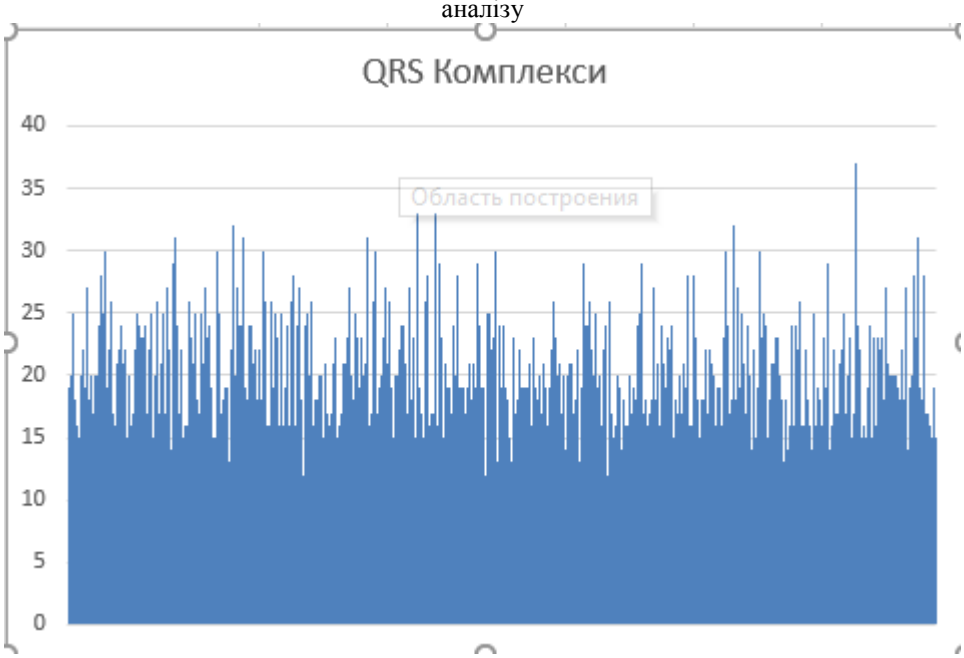

Рис. 3.8. Графік розподілу кількості QRS-комплексів для миготливої аритмії

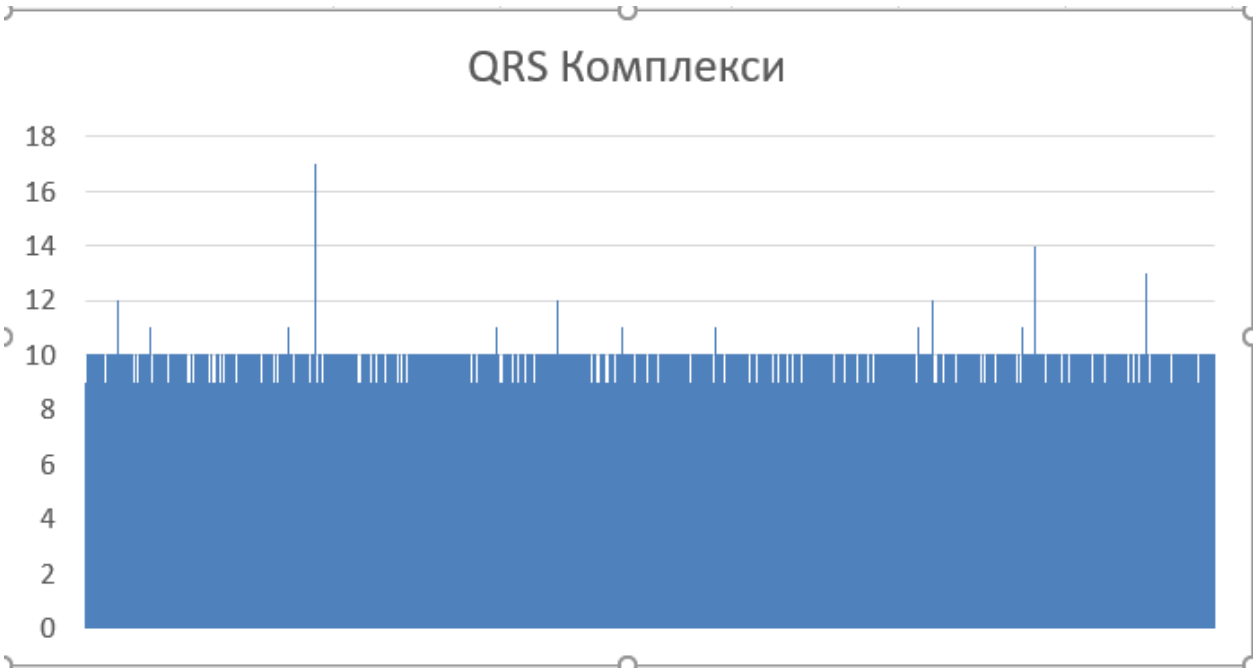

Рис. 3.9. Графік розподілу кількості QRS-комплексів для синусової брадикардії

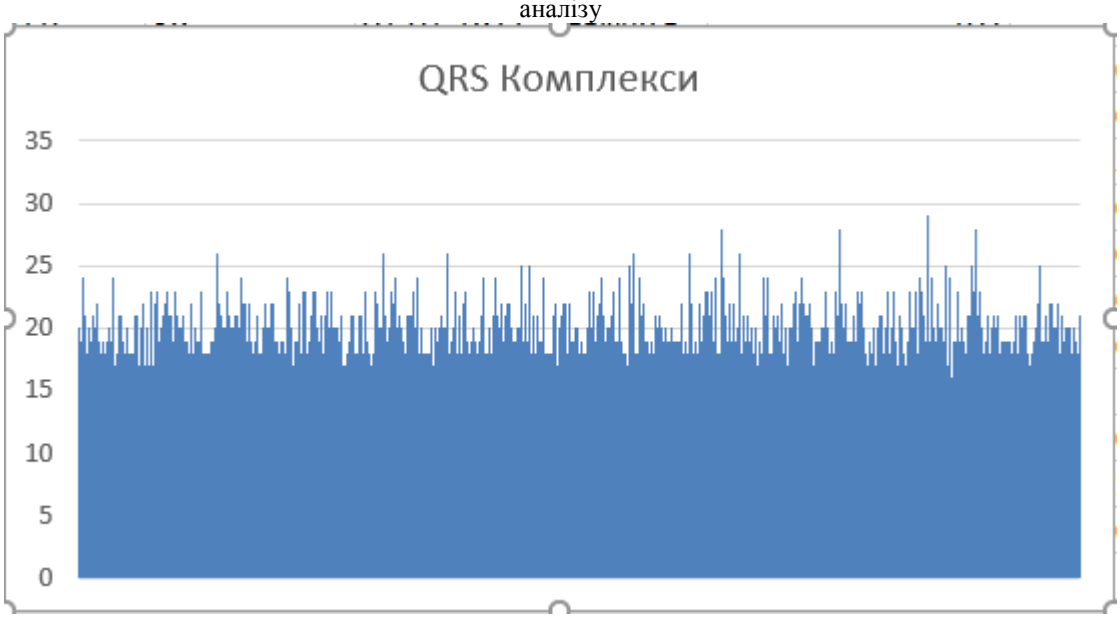

Рис. 3.10. Графік розподілу кількості QRS-комплексів для синусової тахікардії

Після навчання моделі були збережені у файли за допомогою модулю pickle, як файли з розширенням .pkl.

Після навчання нейронна мережа готова до використання в реальних умовах. Принцип роботи такий: нейронна мережа отримує на вхід набір із 12 параметрів кардіограми, починаючи з віку пацієнта, який передається у вигляді масиву у допоміжну функцію.

Допоміжна функція по черзі вивантажує моделі нейронної мережі із файлів .pkl за допомогою модулю pickle та викликає у них функцію predict, в яку і подається отриманий масив параметрів.

Кожна модель на основі даних, отриманих під час навчання, визначає вірогідність наявності у людини того виду аритмії, на виявлення якого вона націлена.

Допоміжна функція повертає 4 значення з плаваючою комою, кожне з яких є вірогідністю того чи іншого виду аритмії.

Оскільки при навчанні нейронної мережі для кожної моделі був створений датасет на тисячу і більше семплів, зазвичай результат є однозначними і в процесі

тестування не було виявлено набору даних, в якому 2 моделі дали б суперечливі результати.

3.4 Графічний інтерфейс користувача

Графічний інтерфейс користувача представлено інтерфейсом командного рядку, який почергово виводить запити на необхідну інформацію і видає результат роботи мережі в кінці. Зовнішній вигляд інтерфейсу представлено на рисунку 3.11.

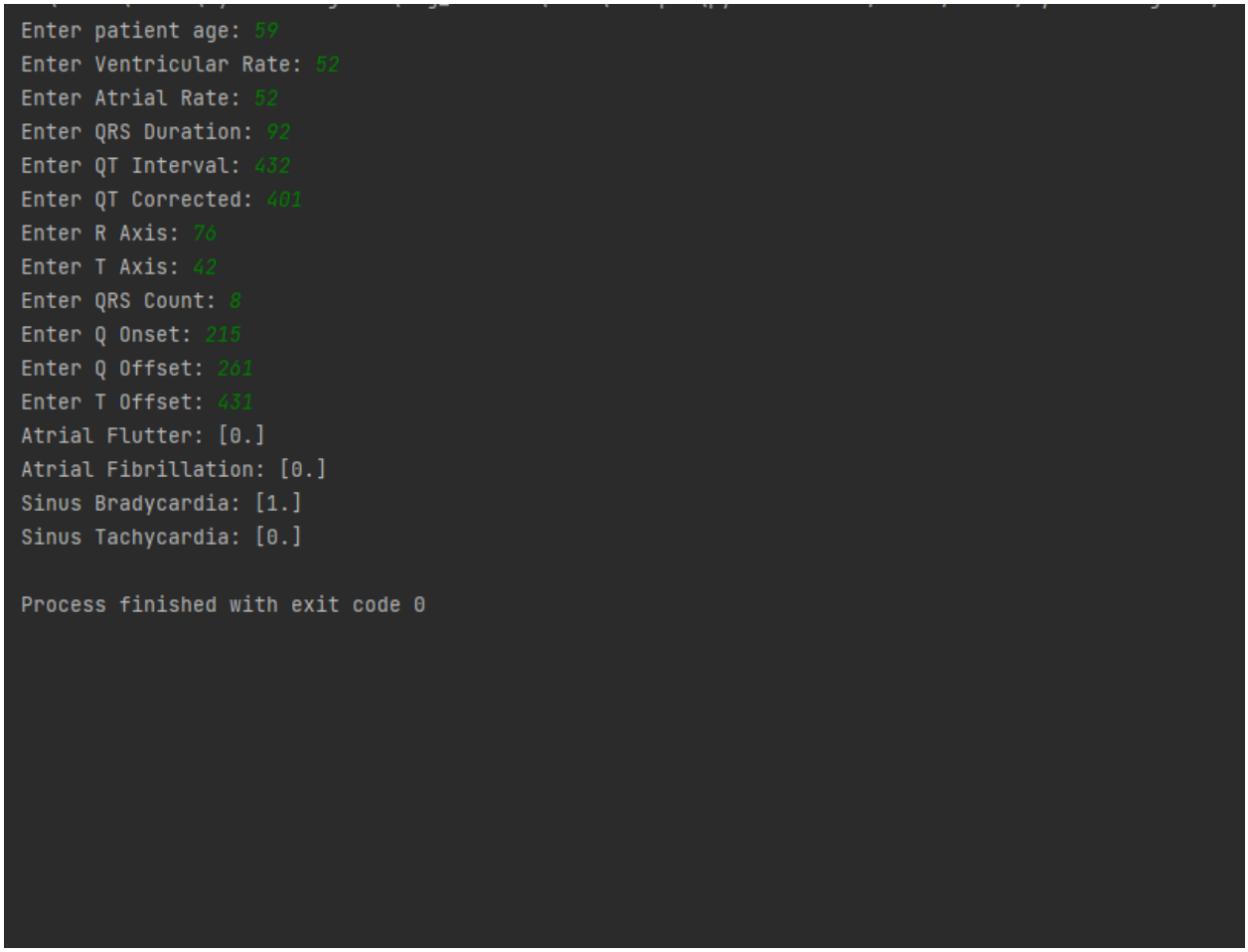

Рис. 3.11. Інтерфейс користувача

3.5 Інструкція користувача

Для роботи з розробленою системою користувачеві слід запустити програму. Для цього є два шляхи.

Запуск програми через IDE:

1. Запустити IDE PyCharm
2. Натиснути кнопку «Файл» в меню вгорі вікна
3. Натиснути кнопку «Відкрити проект»
4. Обрати папку проекту
5. Вгорі вікна, зліва від зеленої кнопки «Пуск» перевірити, що конфігурація встановлена у значенні «main», якщо це не так — змінити
6. Натиснути зелену кнопку «Пуск»
7. Слідувати рекомендаціям щодо роботи з програмою
8. Запуск програми через командний рядок:
9. Запустити командний рядок із директорії проекту
10. В командному рядку ввести наступну команду
    - python main.py
11. Слідувати рекомендаціям щодо роботи з програмою
12. Для роботи з програмою:
13. Запустити програму одним з описаних вище способів
14. Слідуючи підказкам інтерфейсу ввести всі необхідні данні
15. Після кожного вводу натискати клавішу «Enter»
16. Вводити лише цифри
17. Після вводу останнього значення отримати результати

Висновки до розділу 3

Третій розділ присвячений проектуванню і розробці інформаційної системи виявлення супутніх аритмій на основі ЕКГ у хворих на Covid – 19.

В ході розробки системи було проведено аналіз варіантів використання системи, спроектовано внутрішню структуру системи, проведено проектування будови нейронної мережі, яка є основною складовою частиною системи.

Окрім цього розроблено графічний інтерфейс користувача та написано інструкцію користувача, яка покликана допомогти користувачеві в найкоротший час зрозуміти принципи роботи з системою.

ВИСНОВКИ

Метою даної роботи було розроблення нейронної мережі для виявлення можливих аритмій і їх типів на базі аналізу результатів електрокардіограми.

Для виконання поставленої мети було проведено аналіз предметної області. Проаналізовано поняття нейронної мережі, штучного нейрона, моделі нейронної мережі. Проведено огляд парадигм навчання нейронних мереж та сфери використання нейронних мереж. Також розглянуто поняття аритмії.

Були оглянуті та описані обрані засоби розробки: мову програмування Python та бібліотеку sckit-image.

Було реалізовано програмний засіб, а саме, було створено діаграму варіантів використання та діаграму класів. Було створено графічний інтерфейс користувача та написано інструкцію користувача.

Завдяки чіткому виконанню завдань, поставлених на початку роботи, в результаті виконання роботи було отримано повноцінну систему, що здатна виконувати закладений в неї функціонал і готова до використання в реальних умовах.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bhadeshia H. K. D. H. (1999). "Neural Networks in Materials Science" (PDF). ISIJ International. 39 (10): 966–979. doi:10.2355/isijinternational.39.966.
2. Bishop, Christopher M. (1995). Neural networks for pattern recognition. Clarendon Press. ISBN 978-0-19-853849-3. OCLC 33101074.
3. Borgelt, Christian (2003). Neuro-Fuzzy-Systeme : von den Grundlagen künstlicher Neuronaler Netze zur Kopplung mit Fuzzy-Systemen. Vieweg. ISBN 978-3-528-25265-6. OCLC 76538146.
4. Cybenko, G.V. (2006). "Approximation by Superpositions of a Sigmoidal function". In van Schuppen, Jan H. (ed.). Mathematics of Control, Signals, and Systems. Springer International. pp. 303–314. PDF
5. Dewdney, A. K. (1997). Yes, we have no neutrons : an eye-opening tour through the twists and turns of bad science. New York: Wiley. ISBN 978-0-471-10806-1. OCLC 35558945.
6. Duda, Richard O.; Hart, Peter Elliot; Stork, David G. (2001). Pattern classification (2 ed.). Wiley. ISBN 978-0-471-05669-0. OCLC 41347061.
7. Egmont-Petersen, M.; de Ridder, D.; Handels, H. (2002). "Image processing with neural networks – a review". Pattern Recognition. 35 (10): 2279–2301. CiteSeerX 10.1.1.21.5444. doi:10.1016/S0031-3203(01)00178-9.
8. Fahlman, S.; Lebiere, C (1991). "The Cascade-Correlation Learning Architecture" (PDF).
9. created for National Science Foundation, Contract Number EET-8716324, and Defense Advanced Research Projects Agency (DOD), ARPA Order No. 4976 under Contract F33615-87-C-1499.
10. Gurney, Kevin (1997). An introduction to neural networks. UCL Press. ISBN 978-1-85728-673-1. OCLC 37875698.
11. Haykin, Simon S. (1999). Neural networks : a comprehensive foundation. Prentice Hall. ISBN 978-0-13-273350-2. OCLC 38908586.

12. Hertz, J.; Palmer, Richard G.; Krogh, Anders S. (1991). Introduction to the theory of neural computation. Addison-Wesley. ISBN 978-0-201-51560-2. OCLC 21522159.
13. Information theory, inference, and learning algorithms. Cambridge University Press. 25 September 2003. Bibcode:2003itil.book.....M. ISBN 978-0-521-64298-9. OCLC 52377690.
14. Kruse, Rudolf; Borgelt, Christian; Klawonn, F.; Moewes, Christian; Steinbrecher, Matthias; Held, Pascal (2013). Computational intelligence : a methodological introduction. Springer. ISBN 978-1-4471-5012-1. OCLC 837524179.
15. Lawrence, Jeanette (1994). Introduction to neural networks : design, theory and applications. California Scientific Software. ISBN 978-1-883157-00-5. OCLC 32179420.
16. MacKay, David, J.C. (2003). Information Theory, Inference, and Learning Algorithms (PDF). Cambridge University Press. ISBN 978-0-521-64298-9.
17. Masters, Timothy (1994). Signal and image processing with neural networks : a C++ sourcebook. J. Wiley. ISBN 978-0-471-04963-0. OCLC 29877717.
18. Ripley, Brian D. (2007). Pattern Recognition and Neural Networks. Cambridge University Press. ISBN 978-0-521-71770-0.
19. Siegelmann, H.T.; Sontag, Eduardo D. (1994). "Analog computation via neural networks". Theoretical Computer Science. 131 (2): 331–360. doi:10.1016/0304-3975(94)90178-3. S2CID 2456483.
20. Smith, Murray (1993). Neural networks for statistical modeling. Van Nostrand Reinhold. ISBN 978-0-442-01310-3. OCLC 27145760.
21. Wasserman, Philip D. (1993). Advanced methods in neural computing. Van Nostrand Reinhold. ISBN 978-0-442-00461-3. OCLC 27429729.
22. Wilson, Halsey (2018). Artificial intelligence. Grey House Publishing. ISBN 978-1-68217-867-6.

23. Oliphant, Travis (2007). "Python for Scientific Computing". Computing in Science and Engineering. 9 (3): 10–20. Bibcode:2007CSE.....9c..10O. CiteSeerX 10.1.1.474.6460. doi:10.1109/MCSE.2007.58. S2CID 206457124.
24. Millman, K. Jarrod; Aivazis, Michael (2011). "Python for Scientists and Engineers". Computing in Science and Engineering. 13 (2): 9–12. Bibcode:2011CSE....13b...9M. doi:10.1109/MCSE.2011.36.
25. Science education with SageMath, Innovative Computing in Science Education, retrieved 22 April 2019
26. "OpenCV: OpenCV-Python Tutorials". docs.opencv.org. Retrieved 14 September 2020.
27. Dean, Jeff; Monga, Rajat; et al. (9 November 2015). "TensorFlow: Large-scale machine learning on heterogeneous systems" (PDF). TensorFlow.org. Google Research. Retrieved 10 November 2015.
28. Piatetsky, Gregory. "Python eats away at R: Top Software for Analytics, Data Science, Machine Learning in 2018: Trends and Analysis". KDnuggets. KDnuggets. Retrieved 30 May 2018.
29. "Who is using scikit-learn? — scikit-learn 0.20.1 documentation". scikit-learn.org.
30. Jouppi, Norm. "Google supercharges machine learning tasks with TPU custom chip". Google Cloud Platform Blog. Retrieved 19 May 2016.
31. Aahz; Baxter, Anthony (15 March 2001). "PEP 6 – Bug Fix Releases". Python Enhancement Proposals. Python Software Foundation. Retrieved 27 June 2009.
32. "Python Buildbot". Python Developer's Guide. Python Software Foundation. Retrieved 24 September 2011.
33. "1. Extending Python with C or C++ — Python 3.9.1 documentation". docs.python.org. Retrieved 14 February 2021.
34. "PEP 623 -- Remove wstr from Unicode". Python.org. Retrieved 14 February2021.

35. "PEP 634 -- Structural Pattern Matching: Specification". Python.org. Retrieved 14 February 2021.
36. "Documentation Tools". Python.org. Retrieved 22 March 2021.
37. Jump up to:a b "Whetting Your Appetite". The Python Tutorial. Python Software Foundation. Retrieved 20 February 2012.
38. "In Python, should I use else after a return in an if block?". Stack Overflow. Stack Exchange. 17 February 2011. Retrieved 6 May 2011.
39. Lutz, Mark (2009). Learning Python: Powerful Object-Oriented Programming. O'Reilly Media, Inc. p. 17. ISBN 9781449379322.
40. Fehily, Chris (2002). Python. Peachpit Press. p. xv. ISBN 9780201748840.
41. "TIOBE Index". TIOBE - The Software Quality Company. Retrieved 26 February 2021.
42. Blake, Author Troy (18 January 2021). "TIOBE Index for January 2021". Technology News and Information by SeniorDBA. Retrieved 26 February 2021.
43. TIOBE Software Index (2015). "TIOBE Programming Community Index Python". Retrieved 10 September 2015.
44. Prechelt, Lutz (14 March 2000). "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl" (PDF). Retrieved 30 August 2013.
45. "Quotes about Python". Python Software Foundation. Retrieved 8 January2012.

Додатки

Додаток А Лістинг програмного коду

```
import network_helper as nh
PatientAge = float(input('Enter patient age: '))
VentricularRate = float(input('Enter Ventricular Rate: '))
AtrialRate = float(input('Enter Atrial Rate: '))
QRSDuration = float(input('Enter QRS Duration: '))
QTInterval = float(input('Enter QT Interval: '))
QTCorrected = float(input('Enter QT Corrected: '))
RAxis = float(input('Enter R Axis: '))
TAxis = float(input('Enter T Axis: '))
QRSCount = float(input('Enter QRS Count: '))
QOnset = float(input('Enter Q Onset: '))
QOffset = float(input('Enter Q Offset: '))
TOffset = float(input('Enter T Offset: '))
array_of_data = [PatientAge, VentricularRate, AtrialRate, QRSDuration, QTInterval,
        QTCorrected, RAxis, TAxis, QRSCount, QOnset, QOffset, TOffset]
print('Atrial Flutter: ' + nh.get_af(array_of_data))
print('Atrial Fibrillation: ' + nh.get_afib(array_of_data))
print('Sinus Bradycardia: ' + nh.get_sb(array_of_data))
print('Sinus Tachycardia: ' + nh.get_st(array_of_data))


import numpy as np
import pandas as pd
from sklearn import tree
import pickle
df_af_temp = pd.read_csv("AF_DS.csv")
df_afib_temp = pd.read_csv("AFIB_DS.csv")
df_sb_temp = pd.read_csv("SB_DS.csv")
```

```
df_st_temp = pd.read_csv("ST_DS.csv")
af_tmp_len = len(df_af_temp.index)
afib_tmp_len = len(df_afib_temp.index)
sb_tmp_len = len(df_sb_temp.index)
st_tmp_len = len(df_st_temp.index)
af_res = {'result':[]}
afib_res = {'result':[]}
sb_res = {'result':[]}
st_res = {'result':[]}
res1 = []
res2 = []
res3 = []
res4 = []
for i in range(0, af_tmp_len):
   res1.append(1.0)
for i in range(0, afib_tmp_len+sb_tmp_len+st_tmp_len):
   res1.append(0.0)
af_res['result'] = res1
af_frames = [df_af_temp, df_afib_temp, df_sb_temp, df_st_temp]
df_af_final = pd.concat(af_frames)
for i in range(0, afib_tmp_len):
   res2.append(1.0)
for i in range(0, af_tmp_len+sb_tmp_len+st_tmp_len):
   res2.append(0.0)
afib_res['result'] = res2
afib_frames = [df_afib_temp, df_af_temp, df_sb_temp, df_st_temp]
df_afib_final = pd.concat(afib_frames)
for i in range(0, sb_tmp_len):
   res3.append(1.0)
```

```
for i in range(0, af_tmp_len+afib_tmp_len+st_tmp_len):
    res3.append(0.0)
sb_res['result'] = res3
sb_frames = [df_sb_temp, df_af_temp, df_afib_temp, df_st_temp]
df_sb_final = pd.concat(sb_frames)
for i in range(0, st_tmp_len):
    res4.append(1.0)
for i in range(0, af_tmp_len+afib_tmp_len+sb_tmp_len):
    res4.append(0.0)
st_res['result'] = res4
st_frames = [df_st_temp, df_af_temp, df_afib_temp, df_sb_temp]
df_st_final = pd.concat(st_frames)
df_res_af = pd.DataFrame(af_res)
df_res_afib = pd.DataFrame(afib_res)
df_res_sb = pd.DataFrame(sb_res)
df_res_st = pd.DataFrame(st_res)
model_af = tree.DecisionTreeClassifier(criterion="entropy")
model_af.fit(df_af_final, df_res_af)
model_afib = tree.DecisionTreeClassifier(criterion="entropy")
model_afib.fit(df_afib_final, df_res_afib)
model_sb = tree.DecisionTreeClassifier(criterion="entropy")
model_sb.fit(df_sb_final, df_res_sb)
model_st = tree.DecisionTreeClassifier(criterion="entropy")
model_st.fit(df_st_final, df_res_st)
with open('model_af.pkl', 'wb') as f:
    pickle.dump(model_af, f)
with open('model_afib.pkl', 'wb') as f:
    pickle.dump(model_afib, f)
with open('model_sb.pkl', 'wb') as f:
```

```
    pickle.dump(model_sb, f)
with open('model_st.pkl', 'wb') as f:
    pickle.dump(model_st, f)


import numpy as np
import pickle
def get_af(arr):
    arr = np.array(arr).reshape((1, -1))
    with open('model_af.pkl', 'rb') as f:
        model_af = pickle.load(f)
    return str(model_af.predict(arr))
def get_afib(arr):
    arr = np.array(arr).reshape((1, -1))
    with open('model_afib.pkl', 'rb') as f:
        model_afib = pickle.load(f)
    return str(model_afib.predict(arr))
def get_sb(arr):
    arr = np.array(arr).reshape((1, -1))
    with open('model_sb.pkl', 'rb') as f:
        model_sb = pickle.load(f)
    return str(model_sb.predict(arr))



def get_st(arr):
    arr = np.array(arr).reshape((1, -1))
    with open('model_st.pkl', 'rb') as f:
        model_st = pickle.load(f)
    return str(model_st.predict(arr))
```