КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра теоретичної кібернетики

«До захисту допущено»

Завідувач кафедри

Ю. В. Крак _____

(підпис)

« __ »_____20__р.

Дипломна робота на здобуття ступеня бакалавра

За спеціальністю 122 Комп'ютерні науки

на тему:

ІНТЕЛЕКТУАЛЬНІ МЕТОДИ ОБРОБКИ ЕЛЕКТРОКАРДІОГРАМ

Виконав студент 4-го курсу

Запольський Владислав Володимирович _____

(підпис)

Науковий керівник:

д.ф.-м.н., проф. Пашко А. О. _____

(підпис)

Засвідчую, що в цій роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____

(підпис)

Київ – 2021

РЕФЕРАТ

Робота складається зі вступу, 3 розділів, висновку, списку використаних джерел (45 найменувань). Робота містить 11 рисунків. Загальний обсяг становить 47 сторінок, основний текст роботи викладено на 41 сторінці.

Ключові слова: НЕЙРОННА МЕРЕЖА, АРИТМІЯ, МОВА ПРОГРАМУВАННЯ PYTHON, ДІАГРАМА КЛАСІВ, ЕЛЕКТРОКАРДІАОГРАМА.

Об’єктом роботи є методи і технології побудови нейронних мереж. Предметом роботи є обробка результатів ЕКГ методами інтелектуального аналізу з метою виявлення аритмій. Мета роботи полягає в розробленні нейронної мережі для виявлення можливих аритмій і їх типів на базі аналізу результатів електрокардіограми.

В якості засобу розроблення системи було обрано PyCharm та мову програмування Python. Інструментами розроблення були обрані бібліотека для наукових та інженерних розрахунків SciPy, бібліотека для імпорту та експорту даних Pickle та бібліотека для роботи з алгоритмами нейронних мереж sckit-learn.

В процесі роботи над проектом було виділено 4 датасети тестових даних для навчання моделей нейронних мереж, завдяки чому кожна з моделей може виявляти один із чотирьох видів аритмії, а поєднання 4-х моделей в одну мережу дало змогу одразу виявляти яка з аритмій присутня в поданому датасеті.

Результатом роботи став програмний продукт, який може бути використаний в реальних умовах медичних закладів як аналітичний засіб виявлення аритмій на великих кількостях результатів, що спростить роботи лікарів.

При додаванні в мережу моделей для виявлення решти видів аритмії додаток стане повністю функціональним і буде повноцінно покривати

функціонал лікаря-кардіолога при виявленні аритмії на основі результатів ЕКГ.

ЗМІСТ

ВСТУП

На сьогоднішній день світ є досить схильним до процесів діджиталізації та автоматизації. Ці процеси приводять в дію нейронні мережі – обчислювальні системи, які становлять штучну модель, що базується на образі функціонування біологічних нейронів. Нейронні мережі продукуються щодня у великій кількості. При цьому кожна з них має свою чітко визначену задачу.

Актуальністю цієї роботи є те, що нейронні мережі також є досить поширеними у медичній сфері. Яскравим прикладом функціонування штучних нейронних мереж у цій галузі є різноманітні додатки для допомоги медичному персоналу у здійсненні певних дій. При цьому такі мережі включають у себе «людський фактор», тобто похибки, які можуть бути спричинені різноманітними життєвими обставинами (стрес, емоційний стан, перевтома тощо).

Об'єктом дослідження, зважаючи на все вищезазначене, є застосування нейронних мереж у медичній сфері, а саме в кардіології.

Предмет дослідження – аналіз методів та засобів інтелектуального аналізу результатів електрокардіограми.

Мета роботи: розробка такої нейронної мережі, яка мала б змогу виявляти можливу аритмію та її типи на підставі результатів електрокардіограми.

Досягнення мети роботи є можливим за умови виконання таких **завдань:**

- проведення аналізу предметної області
  - аналіз поняття нейронної мережі
  - окреслення понятійної категорії штучних нейронів
  - детальний огляд парадигм навчання нейронних мереж
  - розгляд поняття моделей нейронної мережі
  - аналіз сфер застосування нейронних мереж

    - визначення суті поняття аритмії
- опис обраних засобів розробки нейронної мережі
    - огляд мови програмування Python
    - аналіз бібліотеки sckit-image
- реалізація програмного засобу
    - створення діаграми варіантів використання
    - створення діаграми класів
    - візуалізація графічного інтерфейсу користувача
    - опис інструкції користувача.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Поняття нейронної мережі

До штучних нейромереж (ANN), які зазвичай просто називають нейронними мережами (NN), відносять обчислювальні системи, які нечітко базуються на образі біологічних нейронних мереж, з яких складається мозок тварин. [1]

Нейромережі ґрунтуються на сукупності об’єднаних одиниць або вузлів, які називають штучними нейронами. Штучні нейрони мають здатність до вільного моделювання нейронів біологічного мозку. Кожне поєднання може передавати сигнал іншим нейронам. Надалі штучний нейрон, що отримав сигнал, починає обробляти його. Опісля він сигналізує про нейрони, які наразі підключені до нього. Такий сигнал становить дійсне число. Вихід кожного нейрона можна підрахувати за допомогою певної нелінійноюї функції суми його входів.

З'єднання називають ребрами. У нейронах та краях, як правило, міститься вага. Ця вага має здатність до врегулювання в процесі навчання. При цьому вона може збільшувати або зменшувати інтенсивність сили сигналу під час з'єднання.

У нейронів може бути наявний поріг, в якому сигнали надсилаються лише в разі перетину цього порогу сукупним сигналом. Зазвичай, нейрони мають здатність агрегуватися в шари. При цьому різні шари можуть виконувати різні перетворення на своїх входах. Сигнали рухаються від першого, вхідного шару до останнього шару, тобто вихідного. Це може відбуватись також внаслідок багаторазового обходу шарів.

Початок історії створення штучних нейронів становить факт того, що дослідники Уоррен Маккалок і Уолтер Піттс (1943) розробили обчислювальну модель для нейронних мереж. [3] Наступним досягненням у світі вчених є створення гіпотези навчання Д. О. Хеббом. Ця гіпотеза

базується на механізмі нервової пластичності. Зараз цей механізм більш відомий у світі як геббіанське навчання.

Визначним відкриттям було використання «калькуляторів», обчислювальних машин, Фарлі та Уеслі А. Кларком. Таким чином вчені створили імітацію геббіанської мережі. У 1958 р. Френк Розенблат [6] (1958) створив персептрон.

Перші функціональні мережі з багатьма шарами дослідили та опублікували радянські вчені Івахненко та Лапа в 1965 році. Вони представили такі мережі як груповий метод обробки даних. [8] [9] [10]

Основи безперервного зворотного розповсюдження [8] [11] [12] [13] були виведені у розрізі теорії управління Генрі Келлі [14] в 1960. Для цього дослідник використовував принципи динамічного програмування.

У 1970 р. Сеппо Ліннайнмаа опублікував дослідження стосовно загального методу автоматичного диференціювання (AD) дискретних підключених мереж вкладених диференційованих функцій [16] [17]. У 1973 році дослідник Дрейфус використав зворотне розповсюдження з метою адаптування параметрів контролерів відносно пропорційно до градієнтів помилок. [18] Алгоритм зворотного розповсюдження Пола Вербоса (1975) дав поштовх до практичного навчання багаторівневих нейромереж. У 1982 р. Він застосував метод автоматичного диференціювання Сеппо Ліннайнмаа до нейромереж. Після цього дослідниками Мінські та Папертою було виявлено факт нездатності персептронів до обробки ексклюзивів або схем. Також було виявлено те, що комп'ютерам не вистачає потужності для обробки корисних нейромереж. Дослідження зайшли у глухий кут і призупинились.

Після певного застою досліджень відбувся стрімкий розвиток масштабної інтеграції (ОМС) метал-оксид-напівпровідника (МОП). У формі додаткової технології інтеграція МОП дозволила значно збільшити кількість їхніх транзисторів у цифровій електроніці, що призвело до збільшення обробної потужності, що сприяла б розвитку дослідження і продукування нейромереж у 1980-х рр. [21]

У 1992 році було введено макс-пулінг (згорткова нейронна мережа) з метою допомоги 3D-розпізнаванню об'єктів. [22] [23] [24] Німецький вчений Юрген Шмідхубер запровадив багаторівневу ієрархію мереж у 1992. Ця ієрархія є попередньо впорядкована за одним рівнем з допомогою безконтрольного навчання і відрегульована процесом зворотного розмноження.

Джеффрі Хінтон разом зі співавторами у 2006 році запропонували вивчити подання високого рівня із використанням послідовних шарів двійкових або дійсних прихованих змінних із обмеженою машиною Больцмана [26] для моделювання кожного шару. У 2012 році Ендрю Нг та Джефф Дін разом створили мережу, що має здатність до розпізнавання концепцій вищого рівня, лише за умови перегляду зображень без міток [27].

«Ciresan and colleagues» (2010) [29] висвітлили той факт, що, не зважаючи на проблему зникнення градієнта, графічні процесори роблять зворотнє розповсюдження можливим для багатошарових нейронних мереж прямого зв'язку. [30] У період з 2009 по 2012 рр. A NN почали вигравати призи у різноманітних змаганнях нейромереж. При цьому ANN щоразу все більше наближаються до виконання різноманітних завдань людського рівня. Перш за все такі завдання полягають у розпізнаванні шаблонів та машинному навчанні. Наприклад, двобічно направлена і багатовимірна тривала короткочасна пам’ять (LSTM) від «Graves et al.» здобула перемогу у 3 конкурсах розпізнавання зв’язаного рукописного тексту в 2009 році. При цьому LSTM не мала попередніх знань про мови, які вона розпізнавала. [35] [34].

«Ciresan and colleagues» також створили перші шаблони розпізнавання для досягнення конкурентоспроможності людини/надлюдської діяльності [37] на таких орієнтирах, як розпізнавання дорожніх знаків (IJCNN 2012).

Нейронні мережі вчаться обробляючи приклади. Кожен із таких прикладів містить певні відомі «вхідні дані» та «результат». Ці приклади утворюють асоціації між собою, внаслідок чого потім зберігаються в

структурі даних власне мережі. Навчання нейромережі за таким принципом проводиться на основі встановлення різниці між передбачуваним виходом мережі, що оброблюється, і цільовим результатом. Це становить помилку. На підставі значення цієї помилки у подальшому розвитку подій мережа коригує свої зважені асоціації згідно з правилами навчання. Послідовне коригування може призвести до результату нейромережі, який все більше буде схожим на цільовий результат. Після достатньої кількості таких коригувань навчання може бути призупинено на основі певних критеріїв. Цей процес відомий під назвою «контрольоване навчання».

Ці системи навчаються виконувати завдання, розглядаючи при цьому приклади з умовою відсутності програмування відповідних правил для конкретних завдань. Тому, наприклад, під час ідентифікації зображень вони можуть навчитися розпізнавати зображення, що містять собак, через аналіз прикладів зображень, які вже вручну позначені як "собака" або "немає собаки", і використовувати такі результати для ідентифікації собак на інших зображеннях. Це відбувається без наявності попереднього знання про собак, їхніх особливостей вигляду, будови тіла тощо. Натомість такі системи можуть автоматично генерувати характеристики для ідентифікації на прикладах, які вони обробляють.

1.2. Штучні нейрони як основна складова мережі

ANN містять у собі штучні нейрони. Ці штучні нейрони базуються на образі нейронів біологічних. Кожен зі штучних нейронів має певний набір вхідних даних і виробляє єдиний вихід. Цей вихід можна надіслати декільком іншим нейронам.

До вхідних даних штучних нейронів ANN можна віднести значення характеристик вибірки зовнішніх даних або вони можуть бути виходами інших нейронів. Завданням виходів кінцевих вихідних нейронів нейромережі є розпізнання визначеного об’єкта на зображенні.

Для знаходження виходу нейрона потрібно взяти суму всіх виходів, що зважена вагами зв’язків від входів до нейрона. До суми, яку маємо, слід додати термін упередженості. Така зважена сума також називається активацією. Наприкінці підрахунків зважена сума передається через функцію активації для отримання вихідних даних. Ця функція зазвичай є нелінійною. Початковими даними при цьому називають зовнішні дані, такі як зображення та документи. Кінцеві результати виконують завдання, наприклад, розпізнавання об’єкта на зображенні. [40]

1.3. Парадигми навчання нейронних мереж

Нейронні мережі навчання діляться на 2 парадигми – за участі учителя і без учителя. Нейромережа навчання без учителя на вхідному векторі має готову відповідь. В іншому випадку нейронна мережа самонавчається.

У кожної парадигми є свій визначений набір завдань, які здебільшого не перетинаються. Наразі існує досить багато архітектур нейромереж та їхніх методів навчання, просто вихідними є «алгоритм зворотного поширення помилки» і алгоритми Хебба і Кохонена для навчання з учителем і навчання без участі вчителя відповідно.

Проте за останні кілька років сформувалась також третя парадигма – навчання із т. зв. підкріпленням.

Навчання з вчителем

Така категорія навчання має справу в парах X '; і Y '. Завданням цих пар є відображення їх у функції f: X → Y, де Y – це вчитель, планові бажані виходи, а X – це ті дані, що генерують Y дані.

Значною особливістю такої парадигми навчання є пряме посилання на помилку, яка просто порівнює між плановим і поточним результатом. Параметри мережі зосереджуються у вартісну функцію. Вартісна функція має здатність визначати різницю між бажаним і поточним висновками. Наявність взаємозалежних висновків є обов’язковою умовою навчання з учителем. Нейромережу при цьому варто наділити попередніми знаннями.

Навчання без учителя

В навчанні без учителя дані зазначаються з умовою відсутності міток та жодної класифікації. Натомість така нейронна структура будує логічні висновки на основі знань, отриманих із вхідних даних X.

До такого навчання можна провести аналогію із самонавчанням у дійсному світі. При цьому слід не забувати про те, що в навчанні без учителя неможливо визначити бажаний паттерн. При цьому важливу роль відіграє функція вартості, яка значною мірою впливає на значення нейрона так само

добре, як і на зв'язок між вхідними значеннями. Наприклад, кластеризація, компресія даних, статистичні моделі і мовні моделі.

1.4. Модель нейронної мережі

Модель нейромережі становить собою опис її архітектури та конфігурацію. Також моделі нейромережі властиві використовувані алгоритми навчання.

Архітектура нейронної мережі окреслює загальні принципи, за якими вона будується (пласкошара, повнозв’язна, слабозв'язна, прямого поширення, рекурентна тощо).

Конфігурація уточнює мережеву структуру в розрізі заданої архітектури, а саме число нейронів, входів та виходів мережі, які використовують активаційні функції.

Базові архітектури поділяються на:

1. мережі прямого поширення. У мережах прямого поширення усі зв'язки спрямовані чітко за напрямком від вхідних до вихідних нейронів. Серед таких мереж – персептрон Розенблатта, багатошаровий персептрон і т. д.;
2. рекурентні нейронні мережі. Рекурентні нейронні мережі окреслюються сигналом, який надходить з вихідних нейронів або нейронів прихованого шару. Цей сигнал надалі частково передається назад до нейронів вхідного шару мережі;
3. мережі радіально-базисних функцій. У таких мережах вміщено прихований шар із нейронами, які застосовують радіально-симетричну активаційну функцію. Ці нейрони вирішують задачі селекції та передбачення.
4. мережі Кохонена. Вони становлять цілий клас мереж. Ці мережі використовують навчання без учителя. Вони вирішують кластеризації. У мережах Кохонена наявно 2 прошарки: вхідний (розподільний) і вихідний (кластеризований);
5. карти Кохонена або карти ознак, які самоорганізовуються. Цей різновид мереж Кохонена містить у собі число вихідних нейронів, яке у

свою чергу є значно більшим за число сформованих кластерів. Карти Кохонена зазвичай використовують задля візуального представлення результатів, отриманих у процесі кластеризації багатовимірних даних;

6. повнозв'язні мережі – такі нейронні мережі, у яких кожен нейрон взаємопов’язаний з іншими. Ці нейромережі мають найбільший показник щільності зв'язків;
7. слабозв'язаних мережі. У таких нейромережах нейрони поєднуються виключно з тими, які розташовані до них найближче;
8. нейронні мережі з плоскими шарами – це нейромережі, в яких нейрони утворюють каскади, які також звуться шарами. Нейрони кожного такого шару пов’язані із усіма нейронами наступного і попереднього шару. Всередині шару зв’язки відсутні. Мережі, що містять пласкі шари, можуть бути одношарові (містити тільки один прихований шар) або багатошаровими (за умови наявності кількох прихованих шарів).

Кожна архітектура мережі призначена для того, щоб вирішити визначені задачі аналізу даних (регресії, класифікації, кластеризації, прогнозування). Для цього використовуються відповідні різноманітні спеціально призначені алгоритми навчання.

1.5. Використання нейронних мереж

Сьогодні найпоширенішими нейромережами можна назвати саме згорткові нейронні мережі, які також називають CNN. Принцип роботи CNN – імітування роботи зорової кори головного мозку. Таким чином ці нейронні мережі частково виконують функцію абстрактного мислення, наприклад, ідентифікація зображуваного для графічних планшетів задля створення апаратних платформ з елементами ШІ.

Застосування CNN досить розповсюджене у житті звичайних людей. Так, користувачі Iphone, які використовують розблокування смартфону за допомогою ідентифікації особи, є одним із таких прикладів використання згорткових нейронних мереж у повсякденному житті.

Достатньо поширеними також є RNN – рекурентні нейронні мережі. Особливістю RNN є володіння короткочасною пам’яттю, що допомагає цим мережам з легкістю аналізувати послідовності довільної довжини. RNN ділять потік інформації на елементарні частини, а потім оцінюють їхній взаємозв’язок. Рекурентні нейронні мережі зазвичай застосовуються для розпізнання тексту й мови. Прикладами використання RNN у буденності є програми Shazam, Siri, Google Now тощо.

Популярними є і LSTM – мережі, яким властива як довготривала, так і короткочасна пам’ять. За своєю суттю вони є послідовниками рекурентних нейромереж. Їх використовують для прогнозування коливання значень величин, для аналізу природної мови і т. д. Яскравим прикладом останнього є Google Translate.

GRU, керовані рекурентні блоки, - виникли не так давно. Вони є певною видозміною RNN і вже встигли набути певної популярності серед користувачів. GRU застосовують для синтезу емоційно забарвленого мовлення. Прикладом застосування керованих рекурентних блоків є Google Duplex і Microsoft XiaoIce. Під час тестування останнього людям було

достатньо складно розпізнати бота у співрозмовникові замість звичайної людини.

Поширення так само набувають глибокі нейронні мережі, які також називають DNN. DNN – це будь-яка нейромережа, яка містить у собі більш як 3 шари. Ці шари є основою механізмів поглибленого машинного навчання. Саме завдяки ним можна чітко простежувати не досить явні на перший погляд взаємозв’язки між різнорідними даними. До DNN можна зарахувати, наприклад, IBM Watson, мережу, яка здатна до пошуку відповідностей між розвитком хвороб та різноманітними потенційними факторами серед великої кількості наукових робіт, статей тощо.

Варто також відзначити GAN – генеративно-змагальні мережі. GAN – це набір нейромереж, одна з яких займається продукуванням варіантів, а інша – їх селекцією. Поєднання цього набору дає можливість втілити в життя машинне навчання без залучення додаткової особи вчителя. Це значною мірою підвищує автономність ІІ. PixelDTGAN (нейромережа, яка генерує зображення для каталогів мережевих магазинів), IBM GANPaint (програма, яка має змогу додавати і прибирати ті чи інші графічні об’єкти), DRAGAN (мережа, яка автоматично відтворює мультперсонажів) – найбільш яскраві приклади генеративно-змагальних мереж. Отже, особливість GAN – її вдосконалення з кожною новою отриманою порцією даних.

1.6. Аритмія

Аритмія, також відома як серцева аритмія або аритмія серця, - це група станів, при яких серцебиття є нерегулярним, занадто швидким або занадто повільним. [2] Занадто швидкий пульс - понад 100 ударів на хвилину у дорослих - називається тахікардією, а занадто повільний - менше 60 ударів на хвилину - брадикардією. [2] Деякі типи аритмій не мають симптомів. [1] Симптоми, коли вони є, можуть включати серцебиття або відчуття паузи між серцебиттям. [1] У більш серйозних випадках може спостерігатися

запаморочення, втрата свідомості, задишка або біль у грудях. [1] Хоча більшість типів аритмії не є серйозними, деякі схиляють людину до таких ускладнень, як інсульт або серцева недостатність. [2] [3] Інші можуть призвести до раптової смерті. [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]

Аритмія вражає мільйони людей. [4] У Європі та Північній Америці станом на 2014 рік фібриляція передсердь страждає приблизно від 2% до 3% населення. [8] Фібриляція передсердь та тремтіння передсердь призвели до

112 000 смертей у 2013 році проти 29 000 у 1990 році [9]. Раптова серцева смерть є причиною приблизно половини смертей через серцево-судинні захворювання та близько 15% всіх смертей у всьому світі. [10] Близько 80% раптової серцевої смерті є результатом шлуночкових аритмій. [10] Аритмії можуть виникати в будь-якому віці, але частіше зустрічаються серед людей похилого віку. [4]

РОЗДІЛ 2. ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Мова програмування Python

Python – це інтерпретована мова програмування загального призначення і високого рівня. В основі її філософії лежить читабельність коду, яка полягає у використанні значних відступів. Мовні конструкції Python, а також його об’єктно орієнтований підхід, допомагають програмістам у написанні досить чіткого логічного коду задля використання у різноманітних великих та малих проєктах.

Python має здатність динамічного збору сміття. Також Python відомий як мова, яка включає батареї, завдяки наявності власної повної стандартної бібліотеки.

Наприкінці 1980-х рр. над Python почав ґрунтовно працювати Гідо ван Россум, наступник мови програмування під назвою ABC. Він випустив Python 0.9.0. в 1991 році. [32] У 2000 році вже було випущено версію 2.0 із розширеним функціоналом: розуміння списків, система збору сміття, використання підрахунку посилань тощо. Служба Python 2.7 завершилась у 2020 році, хоча планувалося це завершення ще на початок 2015 року. [33] У 2008 році був запущений Python 3.0. У цй версії було досить ретельно переглянуто і змінено попередню версію, через що переважна частина коду Python 2 не працює на Python 3 без змін.

Python продовжує стабільно перебувати у списку найбільш популярних мов програмування.

Мова програмування Python містить у собі велику кількість парадигм. Зокрема варто зазначити те, що Python повністю підтримує об’єктно орієнтоване та структуроване програмування, а також функціональне та аспектно орієнтоване (в тому числі метапрограмування і метаоб’єкти). Велика кількість інших парадигм підтримуються розширеннями (дизайн за контрактом, логічне програмування тощо).

У Python використовується динамічне введення тексту, комбінація підрахунку посилань, а також збирач сміття, який визначає цикл для керування пам’яттю. Також Python властива динамічна роздільна здатність імен, тобто пізнє прив’язування, що дає змогу під час виконання програми пов’язувати між собою імена змінних та методів.

Дизайном Python пропонується підтримка функціонального програмування згідно з Lisp-традицією. Цей дизайн містить у собі такі функції, як фільтрація, картографія, зменшення, наявність словників, наборів, виразів генераторів тощо. Стандартна Python-бібліотека ж містить у собі модулі itertools та functools. У цих двох модулях реалізовані функціональні інструменти, які є запозиченням інструментарію Haskell та Standard ML.

Філософія Python відображена в документі під назвою Zen of Python (PEP 20). Цей документ містить у собі такі афоризми та вислови:

- Красиве – це краще, ніж потворне.
- Явне є кращим за неявне.
- Просте – значно краще, ніж складне.
- Підрахунок читабельності.

Замість інсталювання в ядро Python усіх його функціональних можливостей, ця мова програмування розроблена з наявністю у собі модулів задля розширення. Це робить Python достатньо широко популярним в якості засобу додавання програмованих інтерфейсів до наявних додатків.

Також, з огляду на філософію Python, його розробники намагаються передбачити та обійти передчасну оптимізацію. Вони відкидають можливість виправлень для частин стандартної еталонної реалізації CPython, які є некритичними та не досить терміновими. Якщо для розробника є важливою швидкість, у Python є можливість переміщення важливих для часу функцій модулів розширення, які написані такими мовами, як C. Так само є доступним Cython, що має здатність перекладати скрипт Python на C і здійснювати прямі виклики API рівня C в інтерпретатор Python.

У спільноті Python досить великого поширення набув неологізм «пітонічний». Цей неологізм має широкий семантичний діапазон. Це пов’язано безпосередньо зі стилем програми. Наприклад, словосполучення «пітонічний код» означає, що код є читабельним, демонструє вільне володіння мовою програмування та відповідає стандартам філософії Python.

Також існує неологізм «Pythonistas» на позначення користувачів і прихильників мови програмування Python. Особливістю Python є те, що ця мова є легкочитабельною, адже форматування коду зазвичай не є візуально нагромадженим, а також в ньому досить часто використовуються англійські ключові слова замість розділових знаків, як це притаманно іншим мовам програмування. Також у Python не використовуються фігурні дужки на позначення розмежування блоків. Крапка з комою після операторів зустрічається, проте це трапляється доволі рідко. У Python набагато менше синтаксичних винятків та особливих випадків, аніж у Pascal або ж C.

Отож, замість фігурних дужок або ключових слів для відмежування блоків у Python використовуються пробіли. Після певних тверджень відступ збільшується в розмірі, зменшення натомість означає те, що поточний блок закінчився. Так візуальна структура програми досить точно відображає семантичну. Ця функція подекуди називається «поза стороною», що властиве також іншим мовам, проте в них відступ частіше за все не набуває семантичного значення. Рекомендований розмір відступу наразі становить 4 пробіли.

Окрім цього, оператори Python включають:

- Оператор присвоєння, який використовує один лиш знак рівності =;
- Оператор if, що умовно виконує блок коду у поєднанні з else та elif;
- Оператор for, призначений для перегляду ітерабельного об’єкту.
- Оператор while для виконання блокування коду, якщо його умова відповідає істині.

- Оператор try, який ловити та обробляти вилученням, вміщеним у вкладеному блоці коду, за винятком речень; це також дає гарантію того, що код очищення в блоці буде виконано попри те, як блок виходить.
- Оператор raise, який застосовується з метою отримування вказаного винятку/повторного підняття спійманого винятку.
- Оператор class виконує блок коду, а також здійснює приєднування локального простору класу до імен задля подальшого використання в об'єктно-орієнтованому програмуванні.
- Оператор def, який визначає функцію/метод.
- Оператор with версії 2.5, яку було випущено у вересні 2006 р. [82]. Цей оператор охоплює блок коду в контекстному менеджері уможливлюючи поведінку, яка подібна до отримання ресурсів. Це – ініціалізація (RAII), що врешті-решт замінює загальну ідіому try /.
- Оператор break, який виходить із циклу.
- Оператор continue, який пропускає цю ітерацію, продовжуючи цим наступний пункт.
- Оператор del використовується для видалення змінної. Це означає, що посилання з імені на значення видаляється. Тоді спроба використання змінної може спричинити помилку. Є можливість перепризначення видаленої змінної.
- Оператор pass виконує функцію NOP. Це синтаксично необхідно для того, аби створити порожній блок код.
- Оператор assert використовується під час налагодження для перевірки умов, що застосовуються в подальшому використанні.
- Оператор yield, що повертає значення з функції генератора. У версії Python 2.5, yield так само є оператором. Така форма застосовується з метою реалізації спільних програм.

- Оператор return, що використовують з метою повернення значення функції.

Оператор присвоєння (=) працює наступним чином. Він прив'язує ім'я як поглинання на окремий об'єкт. Після цього змінні можуть «відскочити» в будь-який момент до абсолютно будь-якого об'єкта. В цій мові програмування ім'я змінної є загальним власником посилання. При цьому ім'я змінної не має фіксованого типу даних, які були б пов'язані із ним. Проте наразі змінна посилатиметься на якийсь об'єкт, що міститиме тип. Це є динамічним набором тексту. Динамічний набір тексту протиставляється статично набраним мовам програмування, у яких у кожній змінній може бути наявне лише значення визначеного типу.

Гвідо ван Россум зазначає, що у Python ніколи не було і не буде можливості підтримки оптимізації хвостових викликів та першокласних продовжень. Проте у версії 2.5 надається більш якісна підтримка схожих до корутинів функціональних можливостей, при цьому генератори Python розширюються. До випуску версії 2.5 ці генератори були ледачими ітераторами, адже інформація передавалась лише односторонньо з генератора. Опісля випуску 2.5 з'явилась можливість передачі інформації у функцію генератора назад, а у версії 3.3 передача інформації стала доступною через кілька рівнів стека.

Деякі вирази Python дуже схожі до тих, які зустрічаються у C та Java, а деякі мають значні відмінності:

- Додавання, віднімання та множення функціонують для мов програмування однаково, проте у Python відрізняється поведінка ділення. Python має два типи поділів: цілочисельний поділ // та плаваюча кома / поділом. Також у Python використовується оператор ** для піднесення до степені.
- У версії 3.5 було представлено оператор @ infix, який призначений для використання бібліотеками з метою множення матриць.

- У версії Python 3.8 було введений синтаксис: =. Цей синтаксис також називають «моржовим оператором».
- Оператор Python is is може використовуватися для порівняння ідентифікацій об'єктів. У Python порівняння можуть бути ланцюговими.
- Python має тип виразу, який зветься розумінням списку, а також більш загальний вираз, тобто генераторський.
- Анонімні функції реалізовуються завдяки лямбда-виразам, проте ці вирази обмежуються умовою, за якої тіло може становити лише 1 вираз.
- У Python розрізняються списки і кортежі. При цьому списки записуються як [1, 2, 3], мають змогу змінюватись, а також не можуть бути використані у якості ключів словників. Кортежі ж записуються як (1, 2, 3). Вони є незмінними, через що у них є змога використання як ключів словників, але лише за умови, що всі елементи кортежу – незмінні. Оператор + може використовуватись для поєднання між собою двох кортежів. Це не змінює їхній вміст, натомість створює новий кортеж, який містить у собі елементи обидвох передбачених кортежів. Водночас це відповідає незмінності кортежу. Дужки за цієї умови є необов’язковими для кортежів, які перебувають в однозначному контексті.
- Python має здатність розпаковувати послідовності. Вони ідентично пов’язані, бо формують кортежні літерали. Як ціле, вони розміщуються зліва від знаку рівності у твердженні про присвоєння. Оператор очікує на ітерабельний об'єкт, зазначений праворуч від знаку рівності, що видає однакову кількість значень, що і надані виразу для запису.

- Рядки в Python можуть бути об’єднані, якщо їх додати (однаковий оператор для додавання цілих чи значень з плаваючою точкою).
- у Python наявні різноманітні типи рядкових літералів:
  - Рядки, які розділяються одинарними/ подвійними лапками. При цьому вони функціонують однаково на відміну від оболонок Unix мови Perl. Ці типи рядків використовують зворотну скісну риску (\) на позначення символу виходу. Інтерполяція рядків стала доступною в Python 3.6 у якості «відформатованих рядкових літералів».
  - Рядки з використанням подвійних лапок, що починаються та закінчуються серією з 3 одинарних/подвійних лапок. Вони можуть охоплювати кілька рядків.
  - Сирі різновиди рядків, які позначені префіксом рядкового літералу перед r. Послідовності втечі не інтерпретуються, через що необроблені рядки можуть приносити користь там, де буквенні зворотні слеші є загальними.
- У Python наявні індекси масивів та вирази нарізування масивів у списках, які позначені як [ключ], [старт: зупинка] або [старт: зупинка: крок]. Індекси ґрунтуються на підставі нуля, а негативні нулі – відносно кінця. Для зрізів використовуються елементи початкового індексу, але при цьому не включає індекси зупинки. Третій параметр зрізу називається кроком. Він дозволяє пропуск та обертання елементів. Індекси зрізів може бути опущено. Як приклад, [:] повертає копію всього списку. Кожен елемент зрізу становить собою неглибоку копію.

Як правило, Python використовується в проєктах штучного інтелекту, а також під час машинного навчання. Допомагають йому у цьому такі бібліотеки, як TensorFlow, Keras, Pytorch, Scikit-learn і т. д. Python також дуже часто використовується для оброблення природної мови, адже саме Python є

яскравим представником мови сценаріїв, яка має простий синтаксис, інструментарій та модульну архітектуру. Python вбудовано у ПЗ методом нескінченних елементів як мову сценаріїв. Так, Python міститься в Abaqus, 3ds Max, Softimage, Blender, Lightwave, Cinema 4D, Houdini, Maya, modo тощо. Також ця мова використовується у декількох відеоіграх. Внаслідок цього вона була прийнята однією з перших доступних мов програмування в Google App Engine.

Велика кількість операційних систем містять Python як стандартний компонент. Так, Python наявний у більшості дистрибутивів ОС Linux, а також міститься як пакет у FreeBSDб NetBSD, OpenBSD тощо. Так само він може використовуватися з командного рядка, тобто терміналу.

Широке застосування Python набув також у такій галузі, як інформаційна безпека, зокрема під час розробки експлойтів.

Зважаючи на широкий функціонал Python, а також виходячи зі сфери використання, слід говорити про мультифункціональність цієї мови програмування, через що саме ця мова була обрана для виконання нашої роботи.

2.2. sckit-image

Scikit-image, раніше відомий як «scikits.image», є бібліотекою для обробки зображень, що мають відкритий код для мови програмування Python. [2] Цей код містить у собі сегментаційні компоненти, геометричні перетворення, маніпуляції кольорового простору, фільтрування, аналізу тощо. Він служить для того, щоб була можливою взаємодія з числовими та науковими бібліотеками, такими, як Python NumPy та SciPy.

Автором проєкту scikit-image є Стефан ван дер Вальт. Він започаткував його під назвою «scikits.image», пояснюючи назву як скорочення від «SciPy Toolkit», яке було розроблено і поширено таким розширенням, як SciPy. [4] Після цього оригінальну базу кодів переписали інші розробники, а вже у листопаді 2012 року серед розмаїття науково-дослідницьких робіт проєкт вже було позначено як «добре досліджений та популярний» [5]. Scikit-image також активно використовувався у Google Summer of Code. [6]

Основною мовою програмування для scikit-image є Python, проте кілька важливих алгоритмів задля досягнення більшої продуктивності написані на Cython.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1. Діаграма варіантів використання

Базова діаграма використання становить представлення взаємодії користувача із системою. При цьому система висвітлює взаємозв'язок користувача із різними випадками використання, в яких користувач бере безпосередню участь. Діаграма випадків використання може розрізняти та ідентифікувати велику кількість різних типів користувачів системи, а також різні випадки використання. Часто така діаграма зазначається у супроводі із іншими типами діаграм, в яких варіанти використання зазвичай представлені еліпсами або кругами.

Попри те, що власне випадок використання дає змогу ретельно розглянути і вивчити кожну можливість, діаграма прикладів використання при цьому може допомогти при забезпеченні більш якісного аналізу системи.

Через те, що схеми використання мають досить спрощений характер, вони можуть становити дієвий інструмент комунікації для зацікавлених сторін. Вченими також було виявлено факт, що діаграми випадків використання передають намір системи у більш простий спосіб для зацікавлених сторін. При цьому діаграми випадків інтерпретуються більш цілісно, ніж діаграми класів.

Діаграма використання має на меті зобразити динамічний аспект системи. Також у діаграмі можуть бути використані додаткові схеми і документація з метою надання цілісного уявлення про функціональні та технічні особливості системи.

Елементи:

- рамки системи – прямокутник, що містить назви у верхніх частинах та еліпси (прецеденти) всередині. Часто рамки системи можуть бути опущені за умови відсутності важливої інформації,

- актор – це стилізований людський персонаж, що є умовним позначенням для набору ролей користувача, який взаємодіє з певною сутністю (системою, підсистемою, класом). Варто пам’ятати, що актори не можуть бути пов'язані між собою з іншим (за винятком відносин щодо обробки/дослідження),
- прецедент – це еліпс із надписом на позначення виконуваної систематичної дії (може включати можливі варіанти), яка призводить до спостережуваного результату. Надпис може бути ім'ям або описом (з точки зору актора) того, "що" робить система (а не "як"). Ім’я прецедента пов’язане із безперервним (атомарним) сценарієм, тобто певною визначеною послідовністю дій, проілюстрованих повіддю. Під час сценарію актори обмінюються систематичними повідомленнями. Сценарій може бути висвітлений на діаграмі прецедентів у відео UML-коментарі. З одним прецедентом може бути пов'язано кілька різних сценаріїв.

На малюнку 3.1 можемо розглянути діаграму варіантів використання, що описує можливі дії користувача в системі.

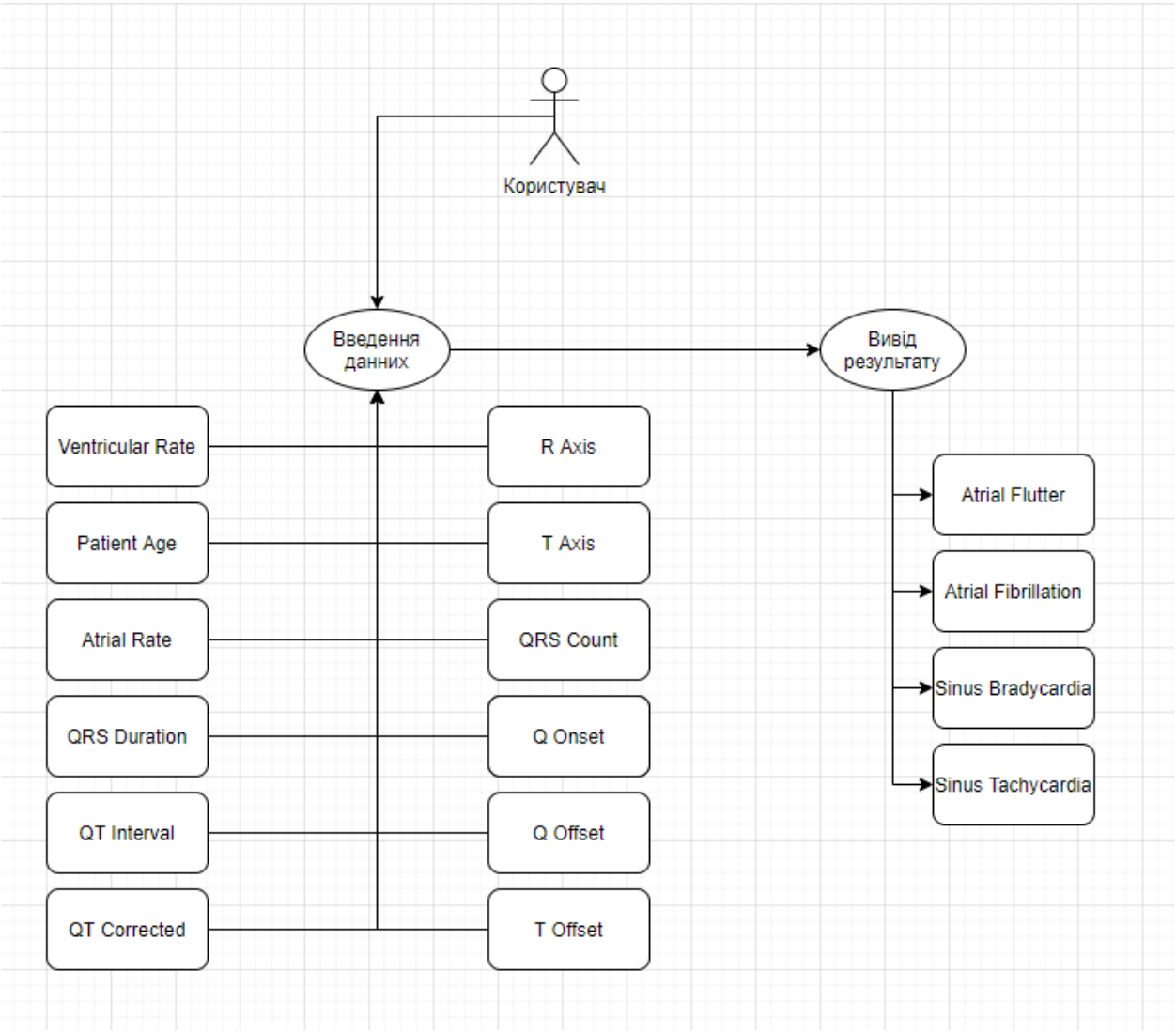

Рис. 3.1 — Діаграма варіантів використання

3.2. Діаграма класів програмного продукту

У програмній інженерії діаграма класів в Уніфікованій мові моделювання (UML) є типом статичної структурної діаграми, яка описує структуру системи, висвітлюючи її класи, атрибути цих класів, операції (або методи) та взаємозв'язки між об'єктами.

Діаграма класів є основним компонентом, на якому базується об’єктно-орієнтоване моделювання. Діаграма класів використовується задля загального концептуального моделювання структури програми та для детального моделювання переведення моделей у програмовий код. Діаграми класів також використовують задля моделювання даних. Класи на таких діаграмах зображуються у вигляді основних елементів і класів, що програмуються.

На схемі класи зазначені у вигляді вікон, що вміщують у собі 3 відділення:

- У верхньому відділенні розміщена назва класу. Її надруковано жирним шрифтом і відцентровано, а перша літера є великою.
- Відділення посередині вміщує класові атрибути. Вони вирівняні за лівим краєм, а перша буква мала.
- У нижньому відділенні вміщені операції, які можуть виконуватись класом. Також вони мають вирівнювання за лівим краєм, а перша буква є малою.

Під час проектування системи ряд класів визначається та групується у схему класів, що допомагає окреслити сталі відносини між ними. За детального моделювання класи концептуального ряду досить часто поділяються на підкласи.

Залежність – це семантичний зв’язок між залежними та незалежними елементами моделі, що існує між двома елементами за умови спричинення змін одного елемента іншим. Така асоціація є односпрямованою. Залежність

при цьому зображується як штрихова лінія з відкритою стрілкою у напрямку від клієнта до постачальника.

Для подальшого опису поведінки систем ці діаграми класів можна доповнити діаграмою стану UML.

Асоціація є родиною посилань. Двійкова асоціація зазвичай представлена у вигляді рядка. Асоціація, що містить 3 ланки, називається потрійною. Слід враховувати, що асоціація може пов'язувати між собою абсолютно будь-яку кількість класів. На кінці асоціації можна зазначити імена ролей/кратність/показники власності тощо.

Існують такі типи асоціацій: двонаправлена, односпрямована, агрегаційна (включає агрегацію композиції) та рефлексивна. Двонаправлені та односпрямовані асоціації є найбільш розповсюдженими.

Агрегація може здійснюватися за умови представлення класу як контейнеру з інших класів. При цьому вміщені класі не мають сильної залежності життєвого циклу від контейнера. Вміст контейнера продовжує існувати навіть тоді, коли контейнер знищено.

В UML такий контейнер графічно зображений як порожниста форма ромба із вміщеним класом на одному рядку, який пов'язує його із вміщеним класом.

Сукупність – це семантично розширений об'єкт, що у більшості операцій трактується як одиниця, хоча фактично цей об'єкт містить у собі декілька менших об'єктів. Це свідчить про те, що 1 з 2 взаємопов'язаних класів, тобто підклас, є спеціалізованою формою другого (супертип), при цьому суперклас є узагальненням підкласу. Практично це означає, що будь-який екземпляр підтипу є також екземпляром суперкласу. Такий зв'язок дуже просто зрозуміти на прикладі формулювання «A=B».

Графічне представлення UML узагальнення представлене у формі порожнистого трикутника на кінці суперкласу рядка (або дерева рядків), що зв'язує його з одним або декількома підтипами.

Відносини узагальнення також називають спадщиною або ж відносинами «є».

Суперклас (базовий клас) у дискурсі відносин узагальнення також називається «батьківським» або базовим типом. Підтип у відносинах спеціалізації також відомий як «дочірній», підклас, похідний клас, похідний тип, клас успадкування або тип успадкування.

Узагальнення може бути висвітлене тільки в діаграмах класів і в діаграмах використання.

Під час моделювання UML взаємозв'язком реалізації вважають взаємозв'язок між 2 елементами моделі, серед яких перший (клієнт) реалізує таку поведінку, яку вказує другий компонент моделі (постачальник).

Графічна візуалізація UML реалізації представлена як порожниста форма трикутника на кінці інтерфейсу штрихової лінії, яка з'єднана з одним або кількома реалізаторами. Проста голівка стрілки зазначається наприкінці інтерфейсу штрихової лінії і з'єднується з користувачами.

Залежність є слабкішою формою зв’язку. Вона вказує на залежність одного класу від іншого через те, що цей клас використовує інший у певний проміжок часу. Один клас є залежним від іншого, за умови якщо незалежний клас є змінною параметра або локальною змінною методу залежного класу. Це є відмінним від асоціації, адже в асоціації наявний атрибут залежного класу як екземпляр незалежного. Буває так, що взаємовідносини між двома класами є доволі слабкими. Вони взагалі не реалізовані зі змінними-членами. Швидше вони можуть бути реалізовані як аргументи функції-члена.

Представлення UML асоціації є лінією, що поєднує між собою 2 взаємопов’язані класи. При цьому на кожному кінці рядка є додаткові позначення.

Класи сутності можуть моделювати довгострокову інформацію, яка обробляється системою, а також інколи і поведінку, що пов'язана з цією інформацією. Проте їх не можна визначати як таблиці баз даних чи інших сховищ даних.

Вони зображені у вигляді кіл з короткою лінією, яка прикріплена до нижньої частини кола. Також їх можна візуалізувати у вигляді звичайних класів із позначенням стереотипу «сутність» над назвою класу.

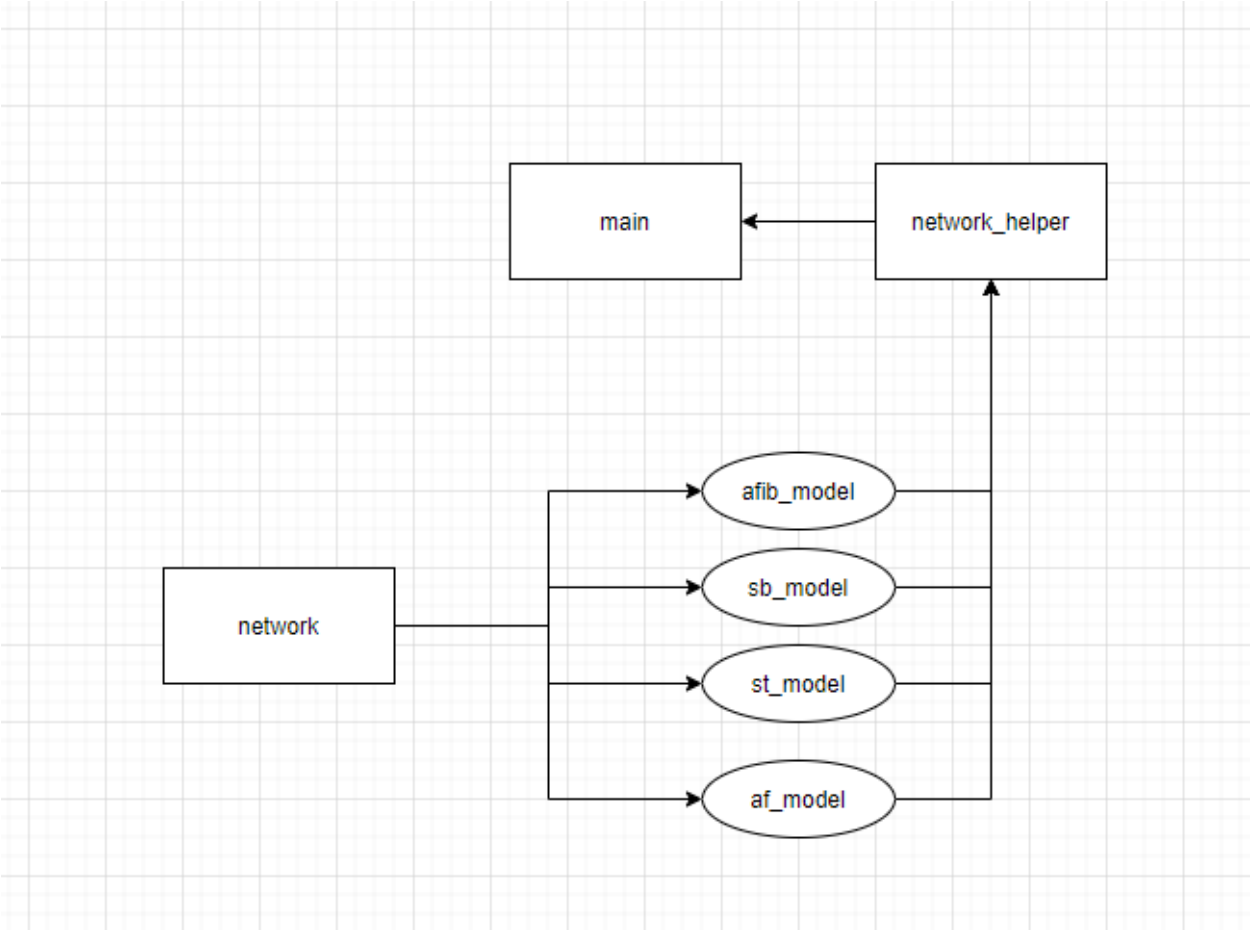

Рис. 3.2 — Діаграма класів

3.3. Структура нейронної мережі

Розроблена нейронна мереж, 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

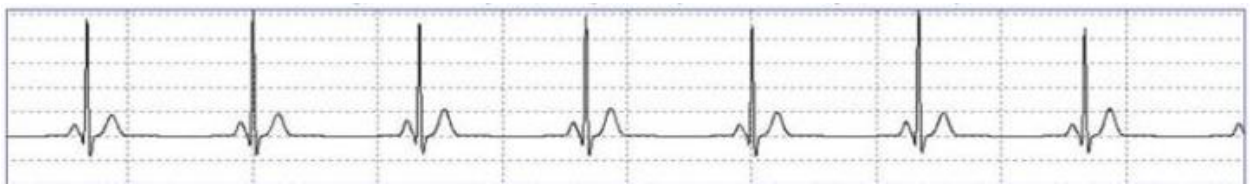

Рис. 3.5 — Синусова брадикардія на ЕКГ

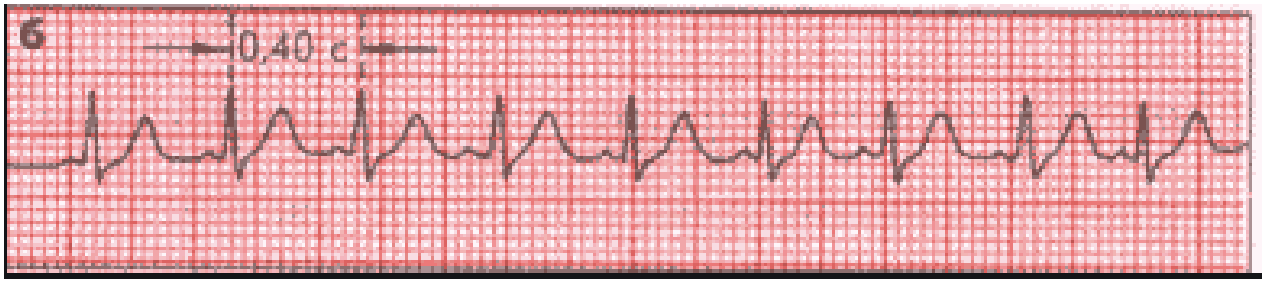

Рис. 3.6 — Синусова тахікардія на ЕКГ

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

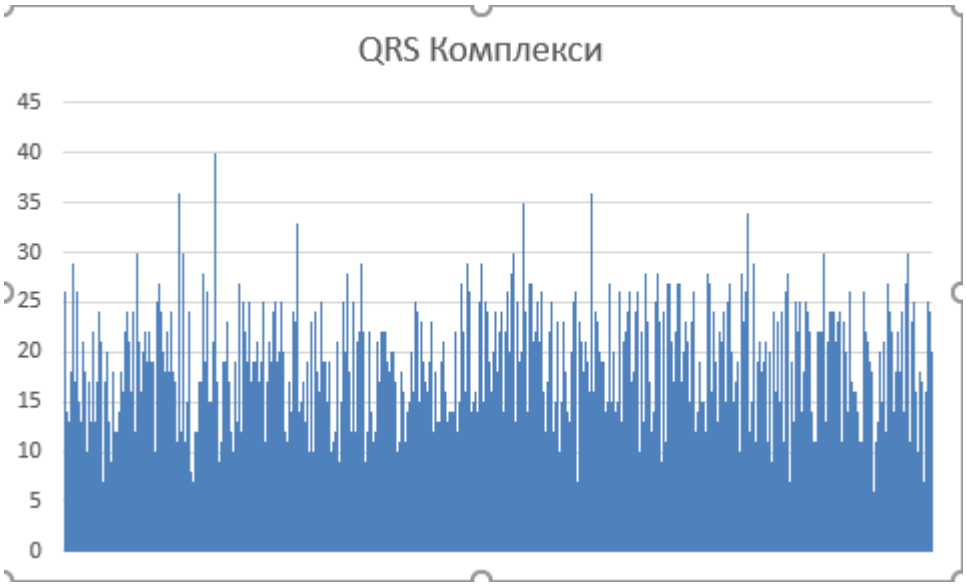

Рис. 3.7 — Графік розподілу кількості QRS-комплексів для фібриляції передсердь

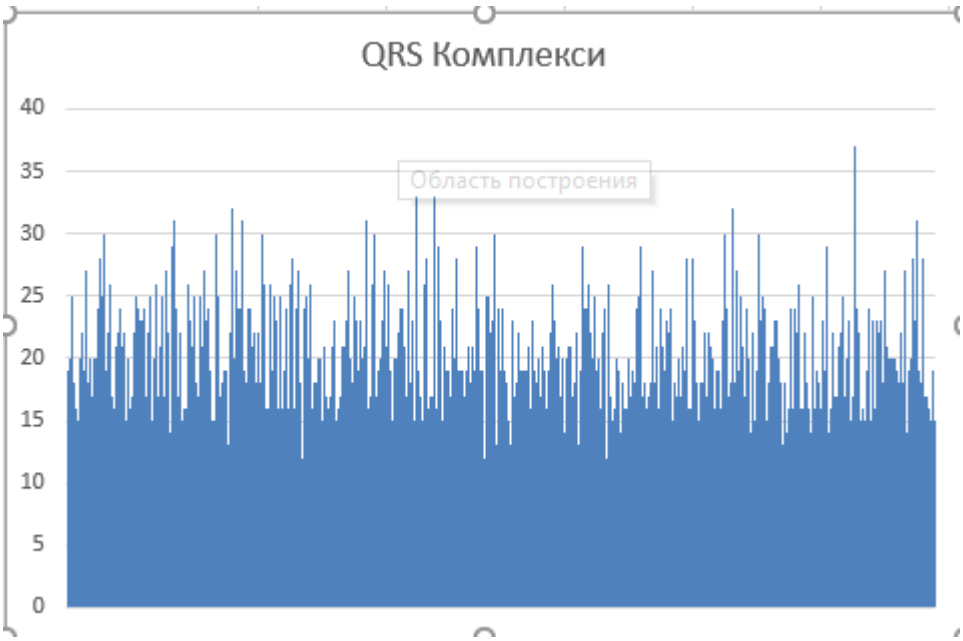

Рис. 3.8 — Графік розподілу кількості QRS-комплексів для миготливої аритмії

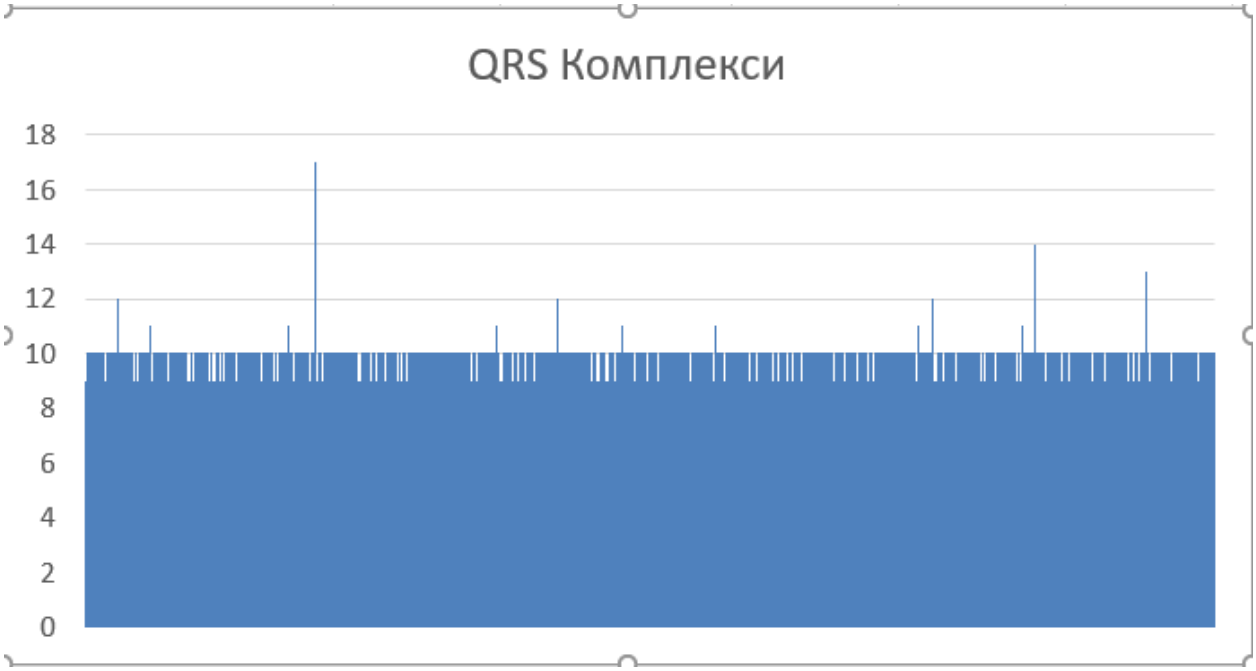

Рис. 3.9 — Графік розподілу кількості QRS-комплексів для синусової брадикардії

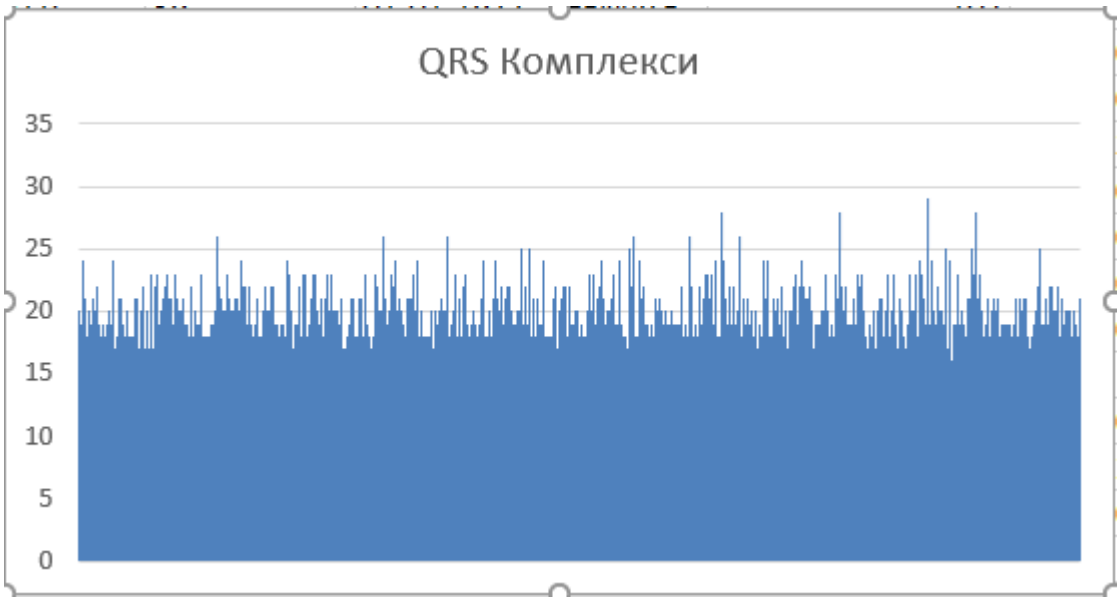

Рис. 3.10 — Графік розподілу кількості QRS-комплексів для синусової тахікардії

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.

3.3. 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.

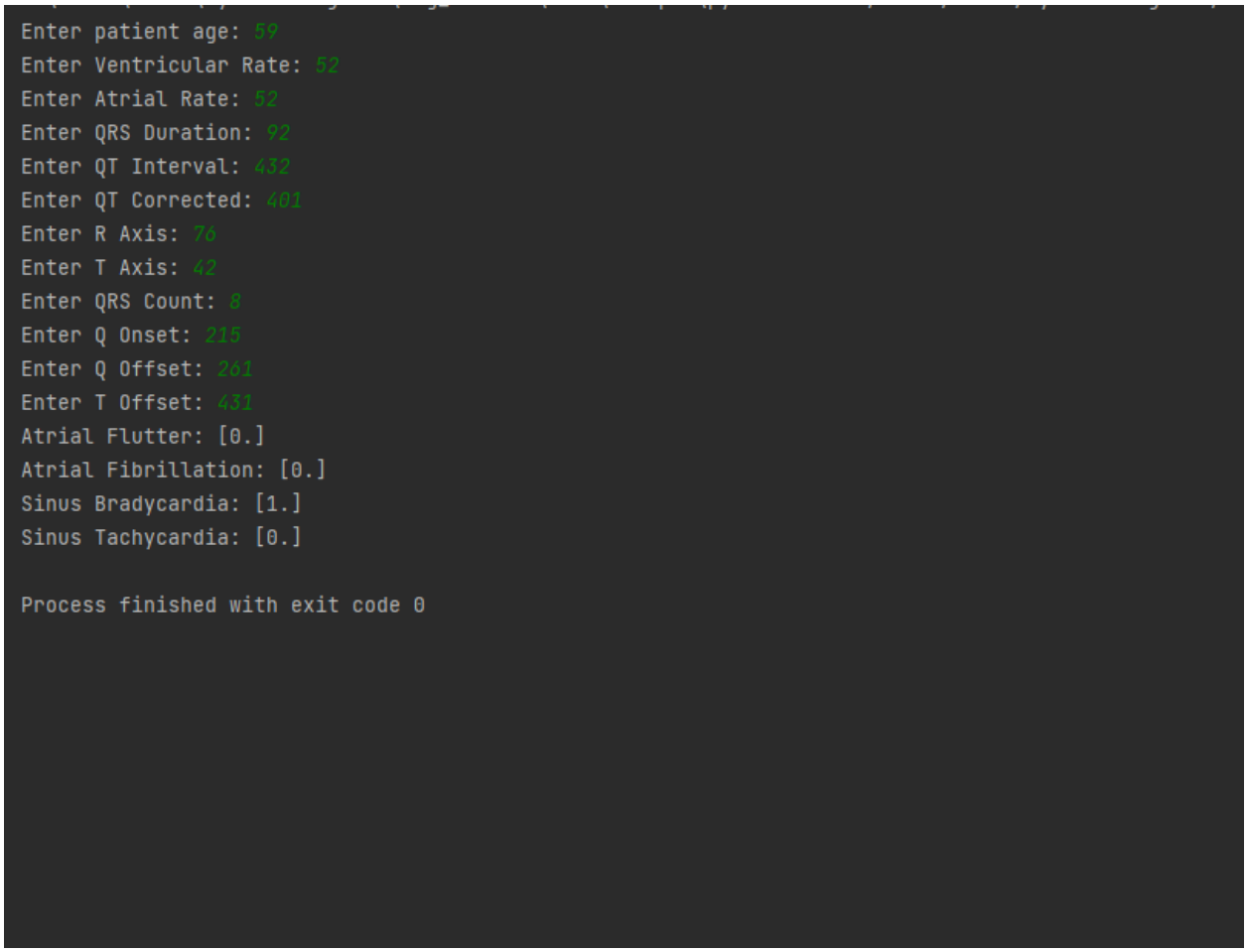

Рис. 3.11 — Інтерфейс користувача

3.4. Інструкція користувача

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

Запуск програми через IDE:

- Запустити IDE PyCharm
- Натиснути кнопку «Файл» в меню вгорі вікна
- Натиснути кнопку «Відкрити проект»
- Обрати папку проекту
- Вгорі вікна, зліва від зеленої кнопки «Пуск» перевірити, що конфігурація встановлена у значенні «main», якщо це не так — змінити
- Натиснути зелену кнопку «Пуск»
- Слідувати рекомендаціям щодо роботи з програмою

Запуск програми через командний рядок:

- Запустити командний рядок із директорія проекту
- В командному рядку ввести наступну команду
  - python main.py
- Слідувати рекомендаціям щодо роботи з програмою

Для роботи з програмою:

- Запустити програму одним з описаних вище способів
- Слідуючи підказкам інтерфейсу ввести всі необхідні данні
- Після кожного вводу натискати клавішу «Єнтер»
- Вводити лише цифри
- Після вводу останнього значення отримати результати

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 виконано наступні завдання:

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

ПЕРЕЛІКИ 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.