НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інформаційне забезпечення робототехнічних систем»
спеціальності 126 «Інформаційні системи та технології»
на тему: «Система автоматизованого проектування моделей нейронних мереж для задач прогнозування»

Виконав:
студент IV курсу, групи ІК-93
Паєвський Дмитро Вікторович _____

Керівник:
Доцент кафедри ІСТ, канд. техн. наук, доцент,
Остапченко Костянтин Борисович _____

Рецензент:
доцент кафедри ІПІ, канд. техн. наук, доцент
Лісовиченко Олег Іванович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2023 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проєкт студенту

Паєвському Дмитру Вікторовичу

1. Тема проєкту «Система автоматизованого проектування моделей нейронних мереж для задач прогнозування», керівник проєкту Остапченко Костянтин Борисович, доцент кафедри ІСТ, к.т.н, доцент, затверджені наказом по університету від «31» травня 2023 р. № 2101-с

2. Термін подання студентом проєкту: 12 червня 2023 року

3. Вихідні дані до проєкту: проблема автоматизованого проектування моделей нейронних мереж для задач прогнозування.

4. Зміст пояснювальної записки:

1. Аналіз проблеми проектування моделей нейронних мереж для задач прогнозування.
2. Розробка архітектури автоматизованої системи проектування моделей нейронних мереж для задач прогнозування.
3. Програмна реалізація та експериментальне дослідження системи.

5. Перелік графічного матеріалу
1. Діаграма компонентів серверної частини
2. Діаграма компонентів системи
3. Блок схема планування задач
4. Діаграма класів

6. Дата видачі завдання 1 березня 2023

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Затвердження теми	01.04.2023	
2.	Аналіз проблеми проєктування моделей нейронних мереж для задач прогнозування	02.04.2023 – 10.04.2023	
3.	Аналіз існуючих рішень автоматизованого проєктування моделей нейромереж	11.04.2023 – 12.04.2023	
4.	Проектування структури програмного продукту	13.04.2023 – 30.04.2023	
5.	Розробка програмного продукту	01.05.2023 – 25.05.2023	
6.	Тестування отриманого продукту	25.05.2023 – 31.05.2023	
7.	Оформлення пояснювальної записки	01.06.2023 – 10.06.2023	

Студент Дмитро ПАЄВСЬКИЙ

Керівник Костянтин ОСТАПЧЕНКО

АНОТАЦІЯ

Паєвський Д.В. Система автоматизованого проектування моделей нейронних мереж для задач прогнозування. КПІ ім. Ігоря Сікорського, Київ, 2023.

Проєкт містить 68 с. тексту, 3 розділи, 31 рисунки, 5 таблиць, 16 джерел, додатки та 4 графічні документи.

Ключові слова: автоматизація, автоматизована система, активаційна функція, втрати, генерація, датасет, інтерфейс, нейрон, нейронна мережа, прогнозування, проектування, точність.

Мета розробки – підвищення ефективності процесу створення моделей нейронних мереж за рахунок формалізованого опису і подання етапів проектування структури і параметрів нейронних мереж для задач прогнозування.

Для реалізації системи було використано: мови програмування C#, JavaScript, мову запитів SQL, бібліотеки Keras, Numpy, pythonnet, React, фреймворк EntityFramework, середовища розробки Visual Studio, Visual Studio Code, MS Sql Management Studio, Postman.

У дипломному проєкті було розроблено систему автоматизованого проектування моделей нейронних мереж, яка генерує задану кількість моделей з випадковою кількістю нейронів у внутрішніх шарах, навчає їх та зберігає навчені моделі та їх результати, а також візуалізує їх.

Дана розробка у майбутньому може бути удосконалена та використана для автоматизованого проектування моделей нейромереж для різноманітних задач, зокрема прогнозування.

SUMMARY

Paievskyi D.V. A system for automated design neural network models for forecasting tasks. Igor Sikorsky KPI, Kyiv, 2023.

The project contains 68 pages text, 3 chapter, 31 figures, 5 tables, 16 literary sources, annexes and 4 graphic documents.

Keywords: automation, automated system, activation function, loss, generation, dataset, interface, neuron, neural network, prediction, design, accuracy.

The aim of the development is to improve the efficiency of the process of creating neural network models through a formalized description and presentation of the stages of designing the structure and parameters of neural networks for prediction tasks.

The following tools and technologies were used for the implementation of the system: C# and JavaScript programming languages, SQL query language, Keras, Numpy, pythonnet libraries, React framework, EntityFramework, Visual Studio, Visual Studio Code, MS SQL Management Studio, and Postman development environments.

In the diploma project, an automated system for designing neural network models was developed. It generates a specified number of models with a random number of neurons in the internal layers, trains them, stores the trained models and their results, and visualizes them.

This development can be further improved and utilized for automated design of neural network models for various tasks, including prediction.

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІК93.100БАК.005 ПЗ	Пояснювальна записка	68		
6	А3	ІК93.100БАК.005 Д1	Діаграма компонентів	1		
7			серверної частини			
8	А3	ІК93.100БАК.005 Д2	Діаграма компонентів	1		
9			системи			
10	А3	ІК93.100БАК.005 Д3	Блок схема планування	1		
11			задач			
12	А3	ІК93.100БАК.005 Д4	Діаграма класів	1		
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІК93.100БАК.005 ТП		
Розроб.		Паєвський Д.В.			Система автоматизованого проектування моделей нейронних мереж для задач прогнозування. Відомість проєкту	Літ.	Аркуш 1 / Аркушів 68
Керівн.		Остапченко К.Б.				Т	
Затв.						КПІ ім. Ігоря Сікорського	

Пояснювальна записка
до дипломного проєкту
на тему: «Система автоматизованого проектування моделей нейронних мереж для задач прогнозування»

Київ – 2023 року

ЗМІСТ

ВСТУП

В сучасному світі, використання штучного інтелекту стає все більш поширеним. Технології машинного навчання полегшують нам життя у багатьох сферах, від медицини – до фізики й математика. Проте, як і будь-яка технологія, ця також має свої недоліки. Один з таких недоліків, це складність проектування моделей нейронних мереж та нерозвинений напрямок автоматизованого проектування, який міг би значно спростити та пришвидшити процес створення моделей нейронних мереж для різних задач, зокрема прогнозування.

Отже, актуальність проблеми, яка зумовила вибір теми дипломного проекту, полягає в необхідності розвитку та впровадження системи автоматизованого прогнозування моделей нейронних мереж для задач прогнозування. В сучасному світі, де обсяги даних постійно зростають, точні та ефективні методи прогнозування є вирішальними факторами для успішної діяльності в різних галузях, таких як фінанси, економіка, медицина та промисловість.

Об'єктом дослідження – процес проектування для прогнозування, який породжує проблемну ситуацію у зазначених галузях.

Предмет – система автоматизованого проектування моделей нейронних мереж, яка визначає тему даного дипломного проекту.

Метою даного проекту є підвищення ефективності процесу створення моделей нейронних мереж за рахунок формалізованого опису і подання етапів проектування структури і параметрів нейронних мереж для задач прогнозування. Це досягається шляхом розробки автоматизованої системи, яка надає зручний і одностайний інтерфейс для визначення параметрів моделей, використання різноманітних конфігурацій та автоматичного навчання. Цей проект має на меті забезпечити користувачів інструментами, які дозволяють ефективно проектувати і налаштовувати моделі нейронних мереж. Формалізований опис структури і параметрів моделей допомагає уникнути непослідовностей і помилок при розробці, а також сприяє швидкому і точному порівнянню різних конфігурацій. Крім того,

система надає можливість автоматичного навчання моделей і перегляду результатів, що сприяє швидкому циклу розробки та вдосконаленню моделей.

Практичне значення одержаних результатів полягає в тому, що розроблена система автоматизованого проектування моделей нейронних мереж має потенціал впровадження у різних галузях діяльності. Вона може стати цінним інструментом для фахівців з прогнозування, дослідників та аналітиків, дозволяючи їм швидко та ефективно генерувати та навчати моделі нейронних мереж для різних прогнозуючих завдань.

Таким чином, цей проект пропонує новий підхід до автоматизованого проектування моделей нейронних мереж для задач прогнозування. Розроблена система дозволить зменшити зусилля та час, необхідні для генерації та навчання таких моделей, та покращити їх точність і ефективність. Це відкриває нові можливості для розробників, аналітиків та дослідників у використанні нейронних мереж для прогнозування, сприяючи розвитку інноваційних рішень та покращенню якості прийняття рішень в різних галузях.

Дана робота має важливе практичне значення та відповідає актуальним потребам розвитку галузей, які використовують прогнозування для своєї діяльності. Розроблена система автоматизованого прогнозування моделей нейронних мереж є потужним інструментом для поліпшення точності та ефективності прогнозування. Завдяки цій системі, фахівці зможуть швидко та зручно створювати та навчати моделі нейронних мереж, а результати їхньої роботи допоможуть в прийнятті обґрунтованих рішень та досягненні успіху в різних галузях.

Проектування виконувалося в рамках ініціативної теми “Створення гібридної обчислювальної технології побудови квазі-формалізованої моделі прогнозування в умовах неоднорідності даних та ненормативних відхилень в системах організаційного управління”. Номер роботи – 0117U002448

Структура дипломного проекту включає наступні розділи: аналіз проблеми проектування моделей нейронних мереж для задач прогнозування, розробка архітектури автоматизованої системи проектування моделей нейронних мереж для задач прогнозування, програмна реалізація та експериментальне дослідження

системи, висновки, список використаних джерел та додатки. Графічна частина проекту включає діаграми та блок схеми алгоритмів.

Загальний обсяг дипломного проекту становить 68 сторінки.

1 АНАЛІЗ ПРОБЛЕМИ ПРОЕКТУВАННЯ МОДЕЛЕЙ НЕЙРОННИХ МЕРЕЖ ДЛЯ ЗАДАЧ ПРОГНОЗУВАННЯ

1.1 Принципи і основні елементи конструювання моделей нейронних мереж

Нейронна мережа є математичною моделлю, яка імітує роботу людського мозку. У цьому підрозділі ми розглянемо базові поняття та принципи, які лежать в основі нейронних мереж.

Нейронна мережа, як і будь-яка система, складається з певних складових, які обов’язково присутні у кожній з них. Кожна нейронна мережа складається з трьох основних складових, а саме нейрона, звязків між ними та функції активації, що описує нейрон [1]:

- Нейрон – базовий елемент нейронної мережі. Він отримує вхідні сигнали, обчислює суму входів, помножених на ваги цих сигналів та застосовує функцію активації до цієї суми, що визначає його вихідний сигнал.
- Зв'язки – вагові коефіцієнти. Нейрони в нейронній мережі з'єднані зв'язками, які мають ваги. Вага визначає важливість сигналу, який передається між нейронами. Ваги налаштовуються під час навчання мережі.
- Функція активації визначає вихідний сигнал нейрона на основі його вагованої суми вхідних сигналів.

Нейронні мережі можуть мати різні архітектури, які відповідають різним типам задач. Розглянемо декілька прикладів архітектур нейронних мереж.

Перцептрон - це найпростіша форма нейронної мережі, яка складається з одного вхідного шару, одного вихідного шару та, можливо, одного або декількох прихованих шарів. Вона використовується для задач класифікації та регресії.

Згорткова – архітектура, яка використовується для обробки зображень та виявлення рис. Вона включає в себе шари згортки, пулінгу та повнозв'язаних шарів.

Рекурентна – архітектура, яка дозволяє моделювати залежності в часі. Вона має зв'язки, які формують цикли і дозволяють передавати інформацію від попередніх часових кроків до майбутніх.

Функції активації визначають поведінку нейрона і можуть впливати на швидкість збіжності та стабільність навчання мережі. Розглянемо різні функції активації та їх властивості [1]:

- Сигмоїдна функція (Sigmoid) – це нелінійна функція, яка перетворює вхідні значення в діапазон між 0 та 1. Вона широко використовується в задачах бінарної класифікації, де потрібно визначити ймовірність належності до класу;
- ReLU (Rectified Linear Unit) – це нелінійна функція, яка повертає вхідне значення, якщо воно більше нуля, і нуль в іншому випадку. Вона є однією з найпоширеніших активаційних функцій, оскільки має швидку обчислювальну швидкодію і допомагає уникнути проблеми з виціканням градієнту;
- Гіперболічний тангенс (Tanh) – функція, що подібна до сигмоїдної функції, але перетворює значення в діапазон між -1 та 1. Вона може бути корисною в нейронних мережах з багатьма шарами і допомагає уникнути проблеми з виціканням градієнту;
- Softmax – функція, яка використовується для перетворення вектору значень в дискретний розподіл ймовірностей. Вона часто використовується в задачах класифікації з багатьма класами;
- Leaky ReLU – це модифікація ReLU, де від'ємні значення мають невеликий нахил. Це допомагає уникнути проблеми "мертвих нейронів", яка може виникнути при використанні ReLU;
- Parametric ReLU (PReLU) - розширена версія Leaky ReLU, де нахил може бути навчений шляхом оптимізації. Це дозволяє моделі самостійно визначати оптимальні значення нахилу;
- Exponential Linear Unit (ELU) – функція, яка апроксимує експоненціальну функцію для негативних значень і є лінійною для позитивних значень. Вона дозволяє моделі виразити як лінійні, так і нелінійні залежності;
- Лінійна функція (Linear) – проста функція, яка повертає вхідне значення без змін. Вона використовується в моделях, де потрібна лінійна залежність між вхідними та вихідними значеннями.

1.2 Особливості застосування нейронних мереж у задачах прогнозування

Існуюсть різні типи задач прогнозування, в яких можна успішно використовувати нейронні мережі. Задачі можна розділити на 3 основних типи:

- Часовий ряд;
- Класифікація;
- Регресія.

Розглянемо детальніше кожний тип.

Часовий ряд – це задача прогнозування, де ми спробуємо передбачити майбутні значення на основі попередніх спостережень у вигляді послідовності часових вимірювань.

Класифікація – у цьому типі задачі нам потрібно визначити категорію або клас, до якого належить вхідний зразок. Нейронні мережі можуть допомогти вирішувати задачі класифікації, використовуючи свою здатність виявляти складні залежності у вхідних даних [2].

Регресія – задача, яка полягає у прогнозуванні числових значень на основі вхідних даних. Нейронні мережі можуть бути ефективними інструментами для розв'язання задач регресії, оскільки вони можуть виявляти складні нелінійні залежності між вхідними та вихідними даними [2].

Відповідно до теми проекту, розглянемо приклади успішного використання нейронних мереж у задачах прогнозування. Візьмемо конкретні домени та сфери, де нейронні мережі показали високі результати. Наприклад:

- Прогнозування цін на фондовому ринку;
- Прогнозування погоди та кліматичних змін;
- Прогнозування продажів та попиту на товари та послуги;
- Прогнозування фінансових показників, таких як прибуток та вартість активів.

Також зазначимо, що існують певні обмеження та виклики, пов'язані з застосуванням нейронних мереж у прогнозуванні. Розглянемо певні обмеження та виклики, які зустрічаються найчастіше:

Однією з найпоширеніших проблем є недостатня кількість даних для навчання моделі. Іноді даних може бути настільки мало, що цього буде не достатньо щоб модель нейронної мережі в процесі навчання віднайшла та сформувала оптимальні зв’язки між нейронами.

Також має місце проблема перенесення (застосування навченої моделі на нових даних). Проблема полягає в тому, що часто навчальні дані, дещо відрізняються від реальних. Тобто іноді під час навчання модель підлаштовується під них і не враховує фактору реальних даних. Таким чином, коли модель починає працювати з реальними даними, які можуть дещо відрізнятися від тих, на яких вона формула зв’язки, результати її роботи можуть мати похибку, яка буде більшою ніж очікувалося. Цю проблему може вирішити донавчанням моделей нейронних мереж на нових даних.

І насамкінець - обробка великих обсягів даних та обчислювальна складність. Іноді навчальних даних настільки багато, що для їх обробки, підготовки та використанні у навчанні потрібно великі обрахункові потужності, які можуть бути досить дорогими, та не завжди легкодоступними.

Цей підрозділ дозволить нам усвідомити особливості та виклики, пов'язані з використанням нейронних мереж у задачах прогнозування та зробити правильні вибори в процесі розробки нашого проєкту.

1.3 Проблеми побудови моделей нейронних мереж для задач прогнозування

Розглянемо детальні етапи побудови та навчання моделей нейронних мереж для задач прогнозування. Загалом можна виділити 4 основних етапи, а саме: постановка задачі та збір даних, вибір архітектури, навчання моделі та оцінка та валідація навченої моделі. Розглянемо кожний етап більш детально.

Перший етап, постановка задачі та збір даних, полягає в чіткому визначенні цілей прогнозування та зборі відповідних даних. Визначається тип задачі прогнозування (часовий ряд, класифікація, регресія) та відповідні характеристики даних, які будуть використовуватись для навчання моделі.

На другому етапі, вибір архітектури моделі, необхідно вибрати відповідну архітектуру нейронної мережі для вирішення поставленої задачі прогнозування. Розглядаються різні типи архітектур, такі як перцептрон, згорткові мережі, рекурентні мережі. Крім того, встановлюються параметри моделі, такі як кількість шарів, розмір шарів та зв’язків між ними.

На етапі навчання моделі застосовуються оптимізаційні алгоритми та функції втрати для навчання моделі на тренувальних даних. В процесі навчання модель здійснює ітеративну оптимізацію параметрів, з метою зменшення помилки прогнозування та покращення точності моделі.

Оцінка та валідація моделі – останній етап включає оцінку та валідацію навченої моделі. Для цього використовуються тестові дані, які не використовувалися під час навчання. Застосовуються метрики оцінки, такі як точність, середньоквадратична помилка чи довірчі інтервали, щоб оцінити якість прогнозів моделі.

Визначившись з етапами, можна розібрати підходи до їх реалізації. Загалом можна виділити 2 підходи: ручний та автоматизований підхід. Варто зазначити, що автоматизувати можна один, або декілька етапів. Основною ідеєю проекту є автоматизація саме вибору архітектури, навчання та оцінки моделі. Розглянемо детальніше переваги та недоліки ручного та автоматизованого проектування моделей нейронних мереж.

Основні недоліки ручного проектування моделей:

- Велика кількість параметрів для налаштування, що вимагає експертних знань та великих зусиль;
- Важкість вибору оптимальної архітектури моделі, особливо при складних задачах прогнозування;
- Обмежені можливості врахування взаємозв'язків та слабкостей ручного моделювання.

Переваги ручного проектування моделей:

- Контроль над процесом проектування та можливість враховувати експертні знання та інтуїцію;

- Можливість створення високо інтерпретованих моделей, які дозволяють зрозуміти, як саме модель здійснює прогнози.

Щодо автоматизованого проектування моделей можна виділити наступні переваги та недоліки:

Недоліки автоматизованого проектування моделей:

- Недостатня якість результатів у порівнянні з ручним проектуванням.
- Обмеженість вибору архітектур та параметрів моделі.

Переваги автоматизованого проектування моделей:

- Швидкість та ефективність процесу проектування моделей.
- Можливість виявлення оптимальних архітектур та параметрів шляхом використання алгоритмів оптимізації та автоматичного пошуку.

На підставі аналізу етапів визначено вимоги до можливості автоматизованого проектування в залежності від обмежень конкретної задачі прогнозування.

1.4 Аналіз існуючих рішень автоматизованого проектування моделей нейронних мереж

На сьогоднішній день на ринку існує декілька інструментів та платформ, які надають можливість автоматизованого проектування моделей нейронних мереж для задач прогнозування. Розглянемо найбільш застосовані в практиках дослідників задач прогнозування у різних галузях:

- AutoML: AutoML є набором інструментів, які дозволяють автоматизувати процес проектування моделей нейронних мереж. Він надає можливість автоматичного вибору архітектури мережі, гіперпараметрів та оптимізації. Приклади AutoML-платформ включають Auto-Keras, Google AutoML, H2O.ai та деякі інші [3].
- Neural Architecture Search (NAS): NAS використовує еволюційні алгоритми, генетичні алгоритми або підхід Reinforcement Learning для пошуку оптимальної архітектури нейронних мережі. Він дозволяє автоматично визначити структуру, розмір та з'єднання нейронів в мережі. Приклади NAS-інструментів включають

Microsoft Neural Architecture Search (NASNet), Google AutoML, ENAS (Efficient Neural Architecture Search) та інші [4].

- Генетичні алгоритми: Генетичні алгоритми базуються на природному відборі і еволюційних принципах. Вони дозволяють автоматично генерувати та оптимізувати архітектури нейронних мереж шляхом комбінування, мутації та відбору найкращих кандидатів. Приклади генетичних алгоритмів для автоматизованого проектування моделей нейронних мереж включають NeuroEvolution of Augmenting Topologies (NEAT), Genetic Algorithms for Rule Set Production (GARP) та інші [5].

Існуючі рішення для автоматизованого проектування моделей нейронних мереж мають свої переваги та недоліки. Деякі з них варто розглянути.

До переваг слід віднести:

- Значне скорочення часу, необхідного для проектування моделей, оскільки процес автоматизований і виконується швидше.
- Можливість виявлення більш оптимальних моделей, які можуть мати кращу точність і ефективність у порівнянні з вручну створеними моделями.
- Легкість використання для недосвідчених користувачів, оскільки багато інструментів мають інтуїтивний і простий інтерфейс.

Виявлено наступні недоліки:

- Відсутність повного контролю над проектуванням моделі, оскільки автоматизовані методи часто обмежені вибором архітектур та гіперпараметрів.
- Потенційна недостатня ефективність або неадекватність автоматично згенерованих моделей в деяких випадках.
- Залежність від якості та репрезентативності тренувальних даних, які використовуються для автоматизованого проектування.

Далі розглянемо доступності існуючих рішень. Більшість існуючих рішень автоматизованого проектування моделей нейронних мереж є відкритими джерелами та доступними для використання дослідниками та розробниками. Багато з них мають відповідні репозиторії на платформах, таких як GitHub, де можна знайти вихідний код, документацію та приклади використання. Це сприяє подальшому

вдосконаленню та розширенню існуючих рішень. Проте більшість потужних рішень потребують дорогої ліцензії, яка може бути доступною для небагатьох користувачів.

Не зважаючи на існування різних інструментів та платформ для автоматизованого проектування моделей нейронних мереж, актуальність створення нових рішень залишається високою. Це пояснюється постійним розвитком галузі машинного навчання та постійно зростаючою потребою в удосконаленні ефективності та точності моделей. Нові рішення можуть пропонувати інноваційні методи та підходи до автоматизованого проектування, що сприятиме подальшому прогресу в цій галузі.

Важливим фактором, що підтримує актуальність створення нових рішень, є необхідність вирішення викликів та обмежень, з якими стикаються існуючі рішення. Наприклад, розвиток технологій, які дозволяють працювати з великими обсягами даних, вимагає нових методів автоматизованого проектування, які можуть ефективно працювати з великими наборами даних. Також, зростання складності задач та вимог до точності моделей створює потребу в розробці нових рішень, які можуть ефективно працювати з цими викликами.

Нові рішення можуть пропонувати розширений функціонал та покращені алгоритми автоматизованого проектування. Наприклад, можуть бути використані нові методи оптимізації, які дозволяють знаходити більш оптимальні архітектури мереж та гіперпараметри. Крім того, нові рішення можуть враховувати специфічні особливості різних типів задач прогнозування, таких як часові ряди, класифікація або регресія, і пропонувати підходи, які краще відповідають цим типам задач.

Однак, варто враховувати, що існуючі рішення мають свої переваги. Наприклад, деякі існуючі платформи для автоматизованого проектування моделей нейронних мереж можуть мати простий та зрозумілий інтерфейс, що полегшує їх використання для непрофесійних користувачів. Деякі рішення можуть мати широкий спектр вбудованих алгоритмів і методів, що дозволяє використовувати їх для різних задач без необхідності власноручно налаштовувати параметри.

Актуальність створення нових рішень для автоматизованого проектування моделей нейронних мереж залишається високою, оскільки це сприятиме

подальшому прогресу у галузі автоматизованого проектування моделей нейронних мереж для задач прогнозування. Враховуючи зростання о0431сягів даних, складність задач та вимоги до точності, нові рішення можуть відповідати сучасним викликам та надавати покращений функціонал для швидкого та ефективного проектування моделей.

1.5 Постановка задачі створення системи автоматизованого проектування моделей нейронних мереж

Ідея створення автоматизованої системи проектування нейронних мереж для задач прогнозування полягає в розробці комплексного програмного рішення, яке надає можливість користувачам автоматизовано створювати, навчати і оцінювати нейромережі для вирішення своїх конкретних проблем прогнозування. Дана система повинна мати ряд функцій, які допомагатимуть ефективно виконувати ці завдання.

У процесі створення системи автоматизованого проектування моделей нейронних мереж, необхідно вирішити декілька ключових задач, які впливають на функціональність та ефективність системи. Розглянемо кожну з цих задач детальніше.

Під час аналізу бібліотек для роботи з нейромережами, необхідно оцінити їх функціональність, широту підтримки, ефективність та зручність використання. Можливі кроки включають:

- Аналіз доступних бібліотек та їх функціональності;
- Порівняння різних бібліотек за швидкістю виконання та пам'яті, наявністю алгоритмів оптимізації та навчання;
- Оцінка якості документації та підтримки спільноти розробників;
- Вибір бібліотеки, що найкраще відповідає потребам проекту.

Аналіз та вибір архітектури програми (системи проектування моделей нейронних мереж). Для створення ефективної системи проектування моделей нейронних мереж, необхідно ретельно спланувати архітектуру програми. Можливі кроки включають:

- Аналіз вимог до системи та ідентифікацію основних функцій, які має виконувати система;
- Розробка концепції архітектури, у тому числі вибір підходів та патернів проектування;
- Вибір технологій та інструментів, які використовуватимуться для реалізації системи;
- Розробка детальної архітектури програми, включаючи взаємодію між компонентами та модулями.

Для ефективного управління моделями, результатами навчання та датасетами, необхідно знайти оптимальне рішення для їх зберігання. Можливі кроки включають:

- Аналіз доступних сховищ для зберігання файлів, таких як локальні файли, бази даних або хмарні рішення;
- Оцінка можливостей кожного сховища, включаючи швидкість доступу, масштабованість та надійність;
- Вибір сховища, яке найкраще відповідає вимогам проекту, забезпечує зручність керування моделями та дозволяє зберігати результати навчання та датасети.

Для автоматизації процесу навчання моделей нейронних мереж, необхідно знайти планувальник задач, який забезпечує ефективне розподілення ресурсів та автоматичний запуск навчання. Можливі кроки включають:

- Аналіз доступних планувальників задач та їх функціональності;
- Порівняння планувальників за швидкістю, масштабованістю та підтримкою різних архітектур мереж;
- Вибір планувальника задач, який найкраще відповідає вимогам проекту та забезпечує автоматизований запуск навчання моделей.

Для забезпечення зручного доступу до системи проектування моделей нейронних мереж, необхідно розробити серверну частину, веб API та базу даних. Можливі кроки включають:

- Аналіз вимог до серверної частини та веб API;

- Розробка архітектури серверної частини, включаючи взаємодію з бібліотеками для роботи з нейронними мережами;
- Вибір технологій та інструментів для реалізації серверної частини та веб API;
- Розробка бази даних для зберігання інформації про моделі, результати навчання та датасети.

Проектування веб клієнту є важливим для забезпечення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів, необхідно розробити веб клієнт, який взаємодіє з системою проектування моделей нейронних мереж. Можливі кроки включають:

- Аналіз вимог до веб клієнта та інтерфейсу користувача.
- Розробка дизайну та архітектури веб клієнта;
- Вибір технологій та інструментів для реалізації веб клієнта;
- Розробка функціоналу веб клієнта для управління моделями, запуску навчання та аналізу результатів.

Аналіз результатів роботи отриманої системи та експериментальне дослідження. Готова система потребую аналізу її роботи та проведення експериментів для дослідження її ефективності. Можливі кроки включають:

- Зібрання та аналіз даних про продуктивність, ефективність та точність отриманих моделей;
- Виявлення можливих проблем або вузьких місць у функціональності системи;
- Розробка плану для вдосконалення системи на основі отриманих результатів.

Кожна з цих задач є важливою для успішної реалізації системи автоматизованого проектування моделей нейронних мереж і вимагає ретельного аналізу, вибору оптимальних рішень та їх реалізації.

Таким чином можна виділити основні поставлені задачі, а саме:

- Аналіз та вибір бібліотек для роботи з моделями нейронних мереж;
- Аналіз та вибір архітектури додатку;
- Пошук рішень для зберігання моделей та результатів навчання;
- Пошук рішень для планування задач;
- Проектування серверної частини системи;

- Проектування клієнтської частини системи;
- Аналіз результатів та проведення експериментального дослідження.

Отже, проведено аналіз проблеми проектування моделей нейронних мереж для задач прогнозування. Розглянуто принципи і основні елементи конструювання моделей нейронних мереж, а також особливості їх застосування у задачах прогнозування. Виявлено проблеми, які виникають при побудові таких моделей та проаналізовано існуючі рішення автоматизованого проектування моделей нейронних мереж.

2 РОЗРОБКА АРХІТЕКТУРИ АВТОМАТИЗОВАНОЇ СИСТЕМИ ПРОЕКТУВАННЯ МОДЕЛЕЙ НЕЙРОННИХ МЕРЕЖ ДЛЯ ЗАДАЧ ПРОГНОЗУВАННЯ

2.1 Формалізація процесу конструювання моделі нейромережі та вимоги до його автоматизації

Основна ідея автоматизації проектування навчання моделей нейронних мереж полягає в генерації n моделей з заданою конфігурацією, де єдиним випадковим параметром є кількість нейронів внутрішніх шарів. Інші характеристики моделей залишаються незмінними.

Автоматизований процес розпочинається зі створення n моделей з однаковою конфігурацією, включаючи архітектуру, функції активації та швидкість навчання. Єдиним фактором, який варіюється, є випадкова кількість нейронів внутрішніх шарів.

Далі, ці моделі піддаються навчанню з використанням підходящих алгоритмів, таких, як зворотне поширення помилки. Кожна модель навчається на однаковому наборі даних і з використанням однакових параметрів навчання.

Після завершення навчання отримуємо результати, які дозволяють оцінити ефективність кожної моделі. Це може бути досягнення певної точності на тестовому наборі даних, середня квадратична помилка або інші метрики оцінки.

На основі отриманих результатів можна вибрати найбільш ефективну модель серед згенерованих. Ця модель буде мати оптимальну кількість нейронів внутрішніх шарів, що дозволяє досягнути найкращих результатів в контексті задачі навчання.

Таким чином, шляхом автоматизованої генерації і навчання n моделей з випадковими кількостями нейронів внутрішніх шарів, ми можемо визначити найбільш ефективну модель, яка досягає найкращих результатів у заданій задачі навчання.

Основною метою автоматизованої системи проектування моделей нейронних мереж для задач прогнозування є надання зручного та ефективного інструменту для автоматизації процесу створення та навчання нейронних мереж. Система має

допомагати в розробці оптимальних моделей, швидкому навчанні та аналізі результатів для досягнення кращих прогнозних показників.

Автоматизована система проектування моделей нейронних мереж повинна мати низку основних функцій та можливостей, які надаються користувачеві. До них входять:

- Створення сесій проектування. Система дозволяє користувачеві створювати окремі сесії для кожного проекту або задачі прогнозування. Це дозволяє легко організувати та керувати проектами та їх компонентами.
- Конфігурація генерації моделей: Користувач може задати конфігурацію генерації моделей, включаючи кількість моделей та випадкову кількість нейронів у прихованих шарах. Це дозволяє отримувати різноманітні варіації моделей для подальшого навчання та аналізу.
- Автоматизоване навчання. Система забезпечує автоматизований процес навчання моделей нейронних мереж. Користувач може вибрати потрібний алгоритм навчання та визначити параметри. Система проводить навчання автоматично, що дозволяє ефективно використовувати час та ресурси.
- Перегляд результатів навчання та графіків. Користувач може переглядати результати навчання для кожної навченої моделі в межах сесії. Система відображає графіки, які показують процес навчання, метрики, помилки та іншу інформацію, що допомагає оцінити ефективність моделей.

Також опишемо вимоги до автоматизованої системи проектування моделей нейронних мереж повинна задовольняти певні ключові вимоги для забезпечення ефективності та зручності використання. До основних вимог входять:

- Можливість створення сесій проектування для організації різних проектів та задач прогнозування.
- Наявність конфігурації генерації моделей для отримання різних варіацій моделей.
- Автоматизоване навчання моделей з можливістю вибору алгоритмів навчання та налаштування параметрів.

- Зручний інтерфейс для перегляду результатів навчання та графіків, що відображають ефективність моделей.
- Можливість збереження результатів навчання в файлове хмарне сховище для зручного доступу та подальшого використання.

Таким чином, надано загальний огляд автоматизованої системи проектування моделей нейронних мереж для задач прогнозування. Визначено загальна мета системи, основні функції та можливості, а також ключові вимоги до системи. Далі розділ буде присвячений більш детальному опису архітектури системи, включаючи інтерфейси взаємодії між модулями, API та розробку бажаного користувацького інтерфейсу.

2.2 Архітектура системи автоматизації проектування моделей нейромережі

Існує кілька популярних архітектурних рішень для back-end розробки, такі як N-шарова архітектура (N-layer), Clean Architecture, Domain-Driven Design (DDD) і Hexagonal Architecture. Для вибору архітектури, потрібно проаналізувати їх сутність детальніше, для кращого розуміння , яке з рішень краще використати в програмному реалізації системи.

N-шарова архітектура (N-layer) – класичний підхід. N-шарова архітектура розділяє back-end додаток на рівні абстракції (шари) з мінімальним зв'язком між ними. Зазвичай, ця архітектура має три основних шари: Presentation Layer (презентаційний шар), Business Logic Layer (логічний шар бізнесу) і Data Access Layer (шар доступу до даних). Кожен шар виконує конкретні функції та має відповідальність за певні аспекти розробки. Цей підхід дозволяє забезпечити розбиття функціональності на логічні групи та полегшує тестування та розширення системи [6]. Схема N-шарової архітектури наведено на рисунку 2.1.

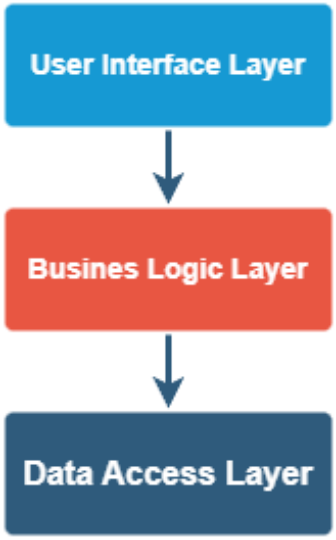

Рисунок 2.1 – Діаграма N-шарової архітектури

Clean Architecture (Чиста архітектура) – принциповою системою розробки програмного забезпечення, яка ставить акцент на розділення бізнес-логіки від деталей реалізації. Цей підхід базується на засаді "залежностей від напрямку", де внутрішні шари не залежать від зовнішніх, і весь код логіки розташований в середині системи. Clean Architecture використовує концепції, такі як Entities (сутності), Use Cases (використання), Interfaces (інтерфейси) та Gateways (шлюзи), що дозволяють розділити систему на логічні компоненти та забезпечити їх незалежність [7]. На рисунку 2.2 зображено базову архітектуру чистої архітектури.

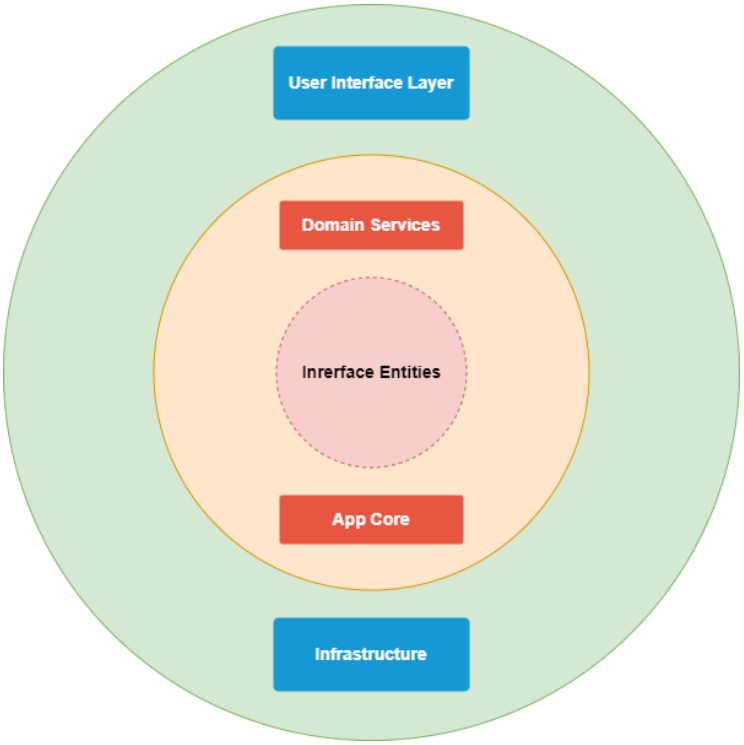

Рисунок 2.2 – Діаграма чистої архітектури

Проєктування з орієнтацією на домен - це підхід до розробки програмного забезпечення, який ставить акцент на розуміння та моделювання домену додатку. DDD пропонує створення якомога ближче до реального світу моделей домену та використання спеціальних термінів та концепцій для опису функціональності системи. Цей підхід використовує техніки, такі як Aggregate (агрегати), Entity (сутності), Value Object (об'єкти значень) та Repository (репозиторії), щоб забезпечити чітку модульність та розбиття функціональності на контексти домену [7]. Проста діаграма DDD архітектури зображено на рисунку 2.3.

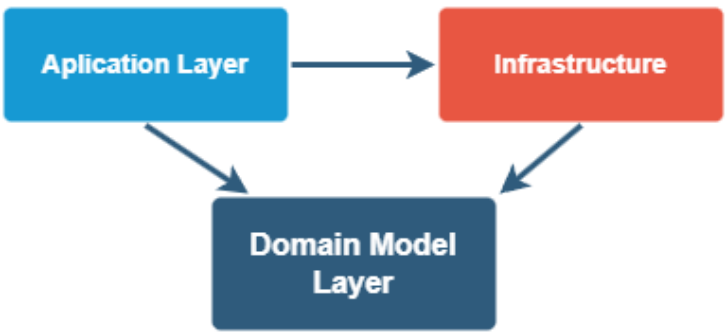

Рисунок 2.3 – Діаграма DDD

Hexagonal Architecture (Гексагональна архітектура), відома також як Ports and Adapters (Порти та адаптери), пропонує розділити систему на внутрішні та зовнішні компоненти, що мають ясно визначені межі. За допомогою цього підходу, вся бізнес-логіка розташована в середині гексагонального ядра, що є незалежним від зовнішніх факторів. Вхідні та вихідні дані обробляються за допомогою портів та адаптерів, які забезпечують взаємодію зі зовнішніми системами та компонентами. Дана архітектура зручна для побудови мікросервісних систем [8]. Діаграма шестикутної архітектури зображена на рисунку 2.4.

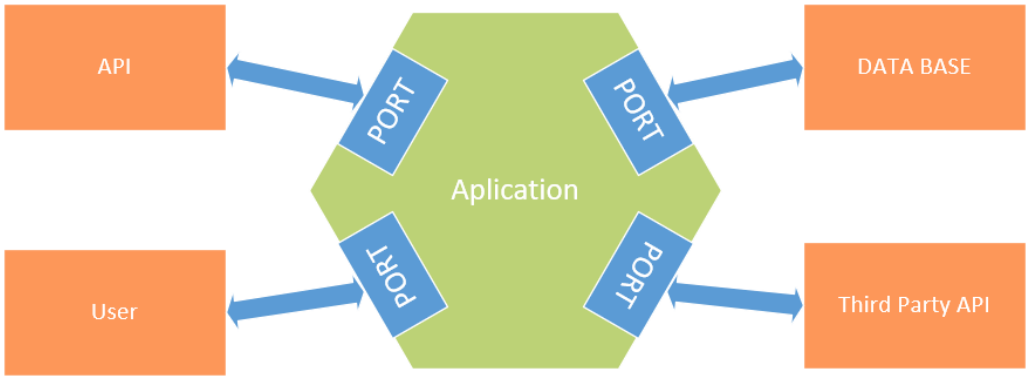

Рисунок 2.4 – Діаграма чистої архітектури

Кожен з цих підходів має свої переваги та підходить для різних вимог та сценаріїв розробки. Вибір підходу залежить від конкретних потреб проекту та може бути визначений командою розробників у процесі проектування архітектури back-end системи.

Для розробки даної системи обрано Clean Architecture, оскільки вона відповідає кільком важливим факторам.

В першу чергу, Clean Architecture забезпечує чітке розділення відповідальностей між різними компонентами системи. Вона надає структурний шаблон, який дозволяє відокремити бізнес-логіку від інфраструктурних аспектів. Це забезпечує більшу гнучкість, підтримку масштабованості та зручність у підтримці кодової бази [6].

Другий важливий аспект Clean Architecture - це його зосередженість на бізнес-правилах і вимогах домену. Ця архітектура надає можливість використовувати принципи DDD (Domain-Driven Design), що сприяє впровадженню моделей, які відображають доменну специфіку, та забезпечує більшу зрозумілість та прозорість коду [6].

Крім того, Clean Architecture забезпечує гнучкість і підтримку змін у вимогах та технологіях. Шарова структура архітектури дозволяє замінювати окремі компоненти без впливу на решту системи. Це забезпечує зручність у розробці, тестуванні та підтримці системи з часом [6].

Враховуючи обрану архітектуру, розроблено структуру системи загалом, а саме вирішено використовувати клієнт серверну архітектуру, з іншими зовнішніми компонентами, такими як база даних, планувальник задач, інтерпретатор Python, файлове сховище. Детальніше на діаграмі на кресленику ІК93.100БАК.005 Д2.

Загалом, Clean Architecture допомагає побудувати гнучку, розширювану та підтримувану систему, яка зосереджується на бізнес-логіці та вимогах домену. Вона пропонує чіткі границі між компонентами системи та дозволяє розробникам працювати відокремлено один від одного. Це робить Clean Architecture відмінним вибором для вашого проекту.

2.3 Структура серверної частини системи

Відповідно обраної архітектури, вирішено, що даному випадку, Clean Architecture включатиме наступні модулі:

- Model – модуль містить інтерфейси та моделі, які використовуються в системі. Він представляє загальну специфікацію даних, яка доступна усім іншим модулям для реалізації та використання;
- Data – містить репозиторії, що відповідають за доступ до бази даних. Вони забезпечують оперативне отримання, зберігання та маніпуляцію даними. Цей модуль використовує інтерфейси та моделі з модуля Model для роботи з даними;

- Domain – містить базову бізнес-логіку системи. Він описує ключові сценарії, правила та обмеження, які визначають роботу системи. Цей модуль використовує інтерфейси та моделі з модуля Model для виконання бізнес-логіки. Також в цьому модулі очікується реалізація зв’язку з фреймворком для роботи з нейронними мережами;
- Orchestrator – проміжний модуль між API та рештою застосунку. Він координує роботу різних компонентів системи, включаючи виклик методів з модулів Domain та Data для обробки запитів, передачу даних та керування потоком виконання;
- Infrastructure – реалізує сервіси, що використовуються для взаємодії з зовнішніми системами, такими як планувальник задач та файлове сховище. Він надає конкретну реалізацію інфраструктурних компонентів, які використовуються в системі;
- API – верхній рівень програми і містить контролери та їх методи для обробки запитів від зовнішніх клієнтів. Цей модуль використовує сервіси та моделі з інших модулів, включаючи Domain, Data та Infrastructure, для обробки запитів та надання відповідей.

Ця структура Clean Architecture дозволяє зберігати чистоту та незалежність кожного модуля, що сприяє легкості розширення, тестуванню та підтримці системи. Кожен модуль має свою конкретну відповідальність та може бути замінений або модифікований без впливу на інші частини системи. Така архітектура підходить для проекту, оскільки дозволяє чітко виділити бізнес-логіку, інфраструктурні складові та взаємодію з зовнішніми системами. Ідейна діаграма архітектури зображена на рисунку 2.5. На кресленику ІК93.100БАК.005 Д1 наведено діаграму компонентів серверної частити.

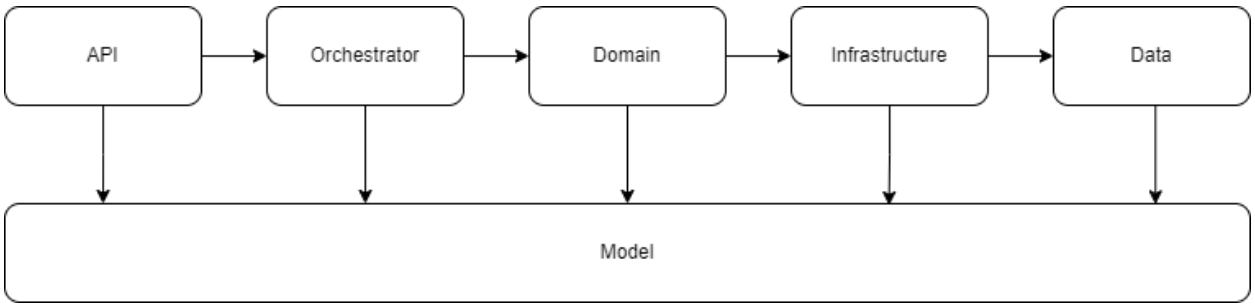

Рисунок 2.5 – Діаграма архітектури системи

2.4 Структура веб клієнту системи

Для розробки веб додатку, обрано архітектурний підхід на основі Single-Page Application (SPA). Дане рішення має кілька вагомих переваг порівняно з іншими підходами. SPA дозволяє створити динамічний та інтерактивний веб-інтерфейс, де взаємодія з сервером відбувається без перезавантаження сторінки. Це покращує користувацький досвід та забезпечує швидку відповідь на дії користувача.

Одна з головних переваг SPA полягає в зменшенні трафіку. Перша сторінка додатку завантажується один раз, а подальша взаємодія з сервером відбувається асинхронно, запитуючи лише необхідні дані. Це сприяє економії пропускної здатності мережі та забезпечує оптимальну продуктивність, особливо в умовах обмеженого інтернет-з'єднання або на мобільних пристроях.

SPA також спрощує розробку та підтримку. Вона дозволяє розробникам зосередитись на фронтенд-логіці без необхідності вирішувати проблеми, пов'язані зі сторінками, переходами та оновленнями. Компонентний підхід SPA дозволяє легко організувати код в окремі компоненти, що полегшує розробку, розширення та тестування додатку [9].

Веб-додаток на основі SPA повинен мати чотири основні компоненти для ефективної роботи:

- Компонент, на якому відображається список сесій та кнопка з можливістю створення нових сесій. Цей компонент забезпечує користувачу огляд доступних сесій та дозволяє створювати нові;

- Компонент-форма для створення сесії. Цей компонент містить поля та елементи керування, що дозволяють користувачу ввести необхідну інформацію для створення нової сесії;

- Компонент для задання конфігурації та початку генерації та навчання моделей. Цей компонент надає користувачу можливість налаштувати параметри для генерації та навчання моделей, а також почати цей процес;

- Компонент, який відображає результати навчання, графіки, діаграми та має можливість завантажити навчену модель та ваги для неї. Цей компонент відображає користувачу результати навчання, що можуть бути представлені у вигляді графіків, діаграм або інших візуальних елементів, і також дозволяє завантажити навчену модель та використовувати її для подальшого використання.

Використання цих компонентів у веб-додатку на основі SPA дозволить створити зручний та ефективний інтерфейс для користувачів, де вони зможуть легко керувати сесіями, налаштовувати та спостерігати за процесом генерації та навчання моделей, а також аналізувати результати навчання.

2.5 Архітектура взаємодії складових через API

API повинне надавати доступ до різних функціональних груп для взаємодії з системою проектування моделей нейронних мереж для задач прогнозування. Також для зручної обробки помилок, статусів та відповіді запиту вирішено використовувати модель обгортку – Result, зображену на рисунку 2.6.

```
{
    "entity": "<response>",
    "status": 0,
    "message": "",
    "isSuccessful": true,
    "isError": false
}
```

Рисунок 2.6 – JSON модель Result

Опис полів моделі Result:

- “entity”: поле, що містить дані відповіді. Тип та структура цього поля можуть варіюватись в залежності від контексту використання;
- “status”: числове поле, що представляє статус відповіді. Значення 0 може використовуватись для позначення успішного виконання запиту, але можуть бути визначені й інші значення для інших статусів або помилок;
- “message”: текстове поле, яке може містити додаткове повідомлення або опис стану відповіді;
- “isSuccessful”: булеве поле, що показує, чи був запит успішним. Значення true вказує на успішне виконання запиту, а значення false вказує на невдале виконання;
- “isError”: булеве поле, що показує, чи виникла помилка під час виконання запиту. Значення true вказує на наявність помилки, а значення false вказує на відсутність помилки.

Ця JSON модель може використовуватись для представлення відповідей системи або сервісу, де “entity“ містить основну інформацію, а решта полів надають контекст та результати виконання запиту.

Для повноцінної роботи додатку, вирішено створити 5 основних груп методів:

- Data – взаємодія з файловим сховищем;
- Execution – керування процесом проектування;
- Sessions – керування сесіями;
- Models – керування моделями;
- FitResults – керування результатами навчання.

Детальніше розглянемо кожну групу. Перша група, Data, відповідає за взаємодію з файловим сховищем. Ця група методів дозволяє отримувати файли зі сховища, наприклад, датасети, моделі, ваги та інші ресурси. Опис метода наведено в таблиці 2.1. Основна ідея це отримання файлів зі хмарного сховища за шляхом, для подальшої обробки.

Таблиця 2.1 – Методи контролера Data

Метод	Запит	Опис
GET	/data/get?filePath=filePath	Отримання файлу зі сховища по його шляху

Група Execution забезпечує можливість запуску процесу генерації та навчання моделей. Група матиме 1 метод, який повинен дозволяти ініціювати процес створення моделей з встановленими параметрами та автоматично навчати їх. Опис метода наведено в таблиці 2.2.

Таблиця 2.2 – Методи контролера Execution

Метод	Запит	Опис
POST	/executions/1/search_optimal_model	Запуск пошуку моделі для заданої сесії по її ідентифікатору

Метод пошуку оптимальної моделі, буде приймати два параметра, а саме csv файл з датасетом для навчання, а також Json конфігурацію зображену на рисунку 2.7. Метод повинен створити n моделей, де n – кількість моделей, які потрібно згенерувати та навчити, та зберегти їх у вказаній сесії. Після чого створити по 2 заплановані задачі, для кожної моделі в планувальнику задач, а саме створення та збереження моделі і відповідно навчання. Після завершення навчання у файловому сховищі та базі даних зберегти результати навчання.

```
{
    "modelRangeConfig": {
      "countOfModels": 2,
      "inputCount": 1,
      "outputCount": 1,
      "countOfInternalLayers": 1,
      "neuronsInLayerMin": 1,
      "neuronsInLayerMax": 100,
      "activationFunc": "tanh",
      "useBias": true
    },
    "compileConfig": {
      "lossFunc": "mean_squared_error",
      "optimizer": "Adam"
    },
    "fitConfig": {
      "epochs": 50,
      "batchSize": 5,
      "isLearnWithValidation": "true",
      "useEarlyStopping": "true",
      "minDelta": 0.01,
      "patience": 10
    },
    "separator": ","
  }
```

Рисунок 2.7 – JSON конфігурація процесу генерації та навчання

Опис полів моделі:

1. “modelRangeConfig”: об'єкт, що містить конфігурацію діапазону моделей нейронних мереж. Включає наступні поля:

- “countOfModels”: кількість моделей, які потрібно створити;
- “inputCount”: кількість вхідних нейронів у моделі;
- “outputCount”: кількість вихідних нейронів у моделі;
- “countOfInternalLayers”: кількість прихованих шарів у моделі;
- “neuronsInLayerMin”: мінімальна кількість нейронів у прихованих шарах;
- “neuronsInLayerMax”: максимальна кількість нейронів у прихованих шарах;
- “activationFunc”: функція активації для нейронів у моделі;
- “useBias”: прапорець, який вказує, чи використовувати зсув у моделі.

2. “compileConfig”: об'єкт, що містить конфігурацію компіляції моделі. Включає наступні поля:

- “lossFunc”: функція втрат для моделі;
- “optimizer”: оптимізатор для моделі.

3. “fitConfig”: об'єкт, що містить конфігурацію навчання моделі. Включає наступні поля:
- “epochs”: кількість епох навчання;
- “batchSize”: розмір пакета даних для однієї ітерації навчання;
- “isLearnWithValidation”: прапорець, який вказує, чи використовувати дані для перевірки під час навчання;
- “useEarlyStopping”: прапорець, який вказує, чи використовувати ранню зупинку навчання;
- “minDelta”: мінімальна зміна втрати для ранньої зупинки навчання;
- “patience”: кількість епох, яку необхідно зачекати до зупинки навчання, якщо втрата не змінюється достатньо швидко.

4. “separator”: символ-роздільник в для читання CSV файлів.

Ця JSON модель використовується для передачі конфігураційних параметрів системи або сервісу, пов'язаних з генерацією, компіляцією та навчанням моделей нейронних мереж.

Даний метод буде отримувати json конфігурацію генерації та навчання моделі.

Група FitResults надає можливість отримувати результати навчання моделей. Ці методи дозволяють отримувати інформацію про результати навчання, такі як метрики, графіки та інші характеристики моделей. Опис контролера наведено в таблиці 2.3. Всі методи мають повертати модель Result, рисунок 2.6, де міститься інформація про успішність запиту та сам результат в полі “entity”. Моделі FitResult, які повинні повертатися в полі “entity” списком зображено на рисунку 2.8.

Таблиця 2.3 – Методи контролера FitResults

Метод	**Запит**	**Опис**
GET	/fit_results/get_all	Отримання всіх результатів навчання з бази даних

GET	/fit_results/1/get_by_model	Отримання результатів навчання за ідентифікатором моделі з бази даних
GET	/fit_results/1/get_by_session	Отримання результатів навчання за ідентифікатором сесії з бази даних

Опис полів моделі FitResults:

- “fitResultId”: ідентифікатор результату навчання;
- “accuracy”: точність моделі, виміряна на тестових даних;
- “loss”: значення функції втрати моделі;
- “historyPath”: шлях до файлу, де зберігається історія навчання моделі;
- “weightPath”: шлях до файлу, де зберігаються ваги моделі;
- “modelId”: ідентифікатор моделі, до якої відноситься результат навчання.

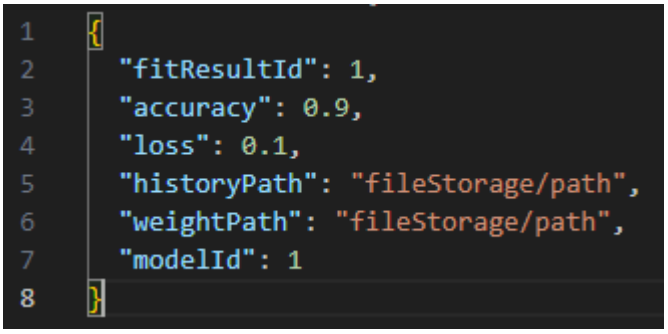

Рисунок 2.8 – JSON модель результатів навчання

Ця JSON модель використовується для представлення результатів навчання моделі нейронної мережі, включаючи метрики точності та втрати, шляхи до файлів з історією навчання та вагами моделі, а також ідентифікатори, що дозволяють пов'язати результати з конкретною моделлю.

Група Models дозволяє отримувати інформацію про моделі, які створені в системі. Ці методи дозволяють отримувати список всіх моделей, інформацію про конкретну модель за її ідентифікатором або за ідентифікатором

сесії, до якої вона належить, детальніший опис методів в таблиці 2.4. Так само, як у випадку з результатами навчання методи будуть повертати результат в полі “entity” json моделі обгортки Result, рисунок 2.6. Усі методи будуть повертати json модель або список json моделей Model, рисунок 2.6.

Таблиця 2.4 – Методи контролера Models

Метод	Запит	Опис
GET	/models/get/all	Отримання всіх моделей з бази даних
GET	/models/1/get	Отримання моделі за ідентифікатором з бази даних
GET	/models/1/get_by_session	Отримання моделей за ідентифікатором сесії з бази даних

Опис полів моделі Model:

- “modelId”: ідентифікатор моделі;
- “modelConfig”: конфігурація моделі у форматі JSON;
- “modelConfigPath”: шлях до файлу, де зберігається конфігурація моделі;
- “sessionId”: ідентифікатор сесії, до якої відноситься модель.

Ця JSON модель використовується для представлення моделі нейронної мережі разом з її конфігурацією. Вона містить ідентифікатор моделі, саму конфігурацію моделі у форматі JSON, шлях до файлу, де зберігається конфігурація моделі, а також ідентифікатор сесії, до якої відноситься модель.

Рисунок 2.9 – JSON модель моделі нейронної мережі

Група Sessions відповідає за роботу з сесіями проектування моделей. Ці методи дозволяють створювати нові сесії, отримувати інформацію про наявні сесії та взаємодіяти з ними. Опис групи наведено в таблиці 2.5. Відповідно до інших методів GET метод буде повертати результат в полі “entity” json моделі обгортки Result, рисунок 2.6. Усі методи будуть повертати json модель або список json моделей Model, рисунок 2.10.

Таблиця 2.5 – Методи контролера Sessions

Метод	Запит	Опис
GET	/sessions/get/all	Отримання всіх сесій з бази даних
POST	/sessions/create	Створення нової сесії

Опис полів моделі:

- “sessionId”: ідентифікатор сесії;
- “sessionName”: назва сесії;
- “modelCount”: кількість моделей у сесії.
- “datasetPath”: шлях до набору даних;
- “isTrained”: прапорець, що позначає, чи були навчені моделі у сесії (значення true означає, що моделі були навчені).

Ця JSON модель використовується для представлення інформації про сесію проектування моделей нейронних мереж. Вона містить ідентифікатор сесії, назву

сесії, кількість моделей у сесії, шлях до набору даних та прапорець, що позначає, чи були навчені моделі у сесії.

Рисунок 2.10 – JSON модель Session

Кожна група методів забезпечує взаємодію з конкретним аспектом системи проектування моделей нейронних мереж для задач прогнозування. Використовуючи ці методи, користувач може отримувати необхідну інформацію, керувати процесом генерації та навчання моделей, а також взаємодіяти з файловим сховищем системи.

2.6 Структура бази даних підтримки процесу конструювання моделей

У процесі розробки структури бази даних для даного проекту, зроблено обґрунтований вибір реляційної бази даних. Реляційні бази даних є широко використовуваними і мають багато переваг, що робить їх привабливими в багатьох проектах.

Однією з головних переваг реляційних баз даних є їх структурованість і організація даних. Реляційні бази даних дозволяють організувати дані у вигляді таблиць, рядків і стовпців, що спрощує розробку, підтримку та керування базою даних. Це забезпечує логічну структуру, що дозволяє зручно працювати з даними для розробників і аналітиків.

Крім того, реляційні бази даних надають потужні механізми запитів, такі як SQL (Structured Query Language), що спрощує отримання необхідної інформації з бази даних. За допомогою SQL можна виконувати складні запити з фільтрацією, сортуванням і з'єднаннями, що дозволяє легко отримувати потрібні дані.

Реляційні бази даних також підтримують цілісність даних. За допомогою обмежень, таких як унікальність, зовнішні ключі і інші обмеження, можна забезпечити консистентність та точність даних у базі даних. Це дозволяє запобігати введенню некоректних або невалідних даних, що є важливим у проектах, де необхідна надійність та точність даних.

Незважаючи на переваги реляційних баз даних, вони також мають деякі недоліки. Один з них - складність масштабування. Реляційні бази даних можуть бути вимогливими до обчислювальних ресурсів і можуть бути складними для масштабування, особливо у випадках з великим обсягом даних або високою навантаженістю. Існують підходи для масштабування реляційних баз даних, такі як реплікація та шарування, але вони можуть бути складними в реалізації і вимагати додаткових зусиль.

Порівняно з іншими рішеннями, такими як нереляційні бази даних, реляційні бази даних часто є надійним та добре вивіреним варіантом для проектів, де важлива структурованість даних, потрібність у складних запитах та забезпечення цілісності даних. Однак, вирішення може залежати від конкретних потреб проекту, таких як розмір даних, швидкість доступу, гнучкість схеми і потреба у горизонтальному масштабуванні [10].

Таким чином розуміючи, що для реалізації проекту, буде використано реляційну базу даних, потрібно визначитись, які сутності будуть зберігатися в базі, та зв’язки з цими сутностями. Відповідно до поставленої задачі, для реалізації проєкту потрібно буде створити 3 таблиці:

- Session – сутність сесії потрібна для розподілу груп моделей на підставі обраної задачі;
- Model – сутність, що описує окрему моделі нейронної мережі, містить конфігурацію, та посилання на файлове сховище, де вона зберігається та її ваги. Модель має зв’язок з сесією завдяки зовнішньому ключу, який посилається на ідентифікатор сесії;
- FitResult – результати навчання, які містять загальні результати навчання, такі як втрати та точність на тестових даних та посилання на файлове сховище де

зберігається більш детальна інформація. Результати навчання містять зовнішній ключ, який посилається на ідентифікатор моделі.

Відповідно оскільки кожна сесія, може мати певну кількість моделей, то зв’язок між ними повинен бути оди до багатьох, при цьому не обов’язковий, оскільки при створенні сесії, моделей в ній може не бути. Зв’язок моделі з результатами навчання, вирішено також зробити один до багатьох, при необхідності це дасть можливість декілька раз навчати одну й ту ж модель, при цьому маючи результати кожного навчання. Ознайомитися з діаграмою відносин між сутностями можна на рисунку 2.11.

Рисунок 2.11 – Діаграма відносин між сутностями

Отже, у даному розділі, проаналізовано та обрано архітектурний підхід, концепт розробки веб-клієнта та розробці структури бази даних. Для розробки серверної частини проекту обрано Clean Architecture, оскільки вона надає чітку структуру, розділення відповідальностей та сприяє підтримці масштабованих і добре організованих додатків. Цей підхід дозволить забезпечити легку розширюваність та зручну підтримку серверної частини проекту.

Вибір концепту розробки веб клієнта на основі single page application обгрунтований з метою забезпечення кращої користувацької взаємодії, швидкості та зручності використання. Цей підхід дозволить зменшити навантаження на сервер, забезпечити швидке відгук на дії користувача та покращити загальний досвід використання додатка.

Структура бази даних була розроблена з використанням реляційної бази даних. Вибір реляційної бази даних обґрунтований на основі її структурованості, можливості використання потужних механізмів запитів та забезпечення цілісності даних. Втілення реляційної бази даних дозволить забезпечити надійність, точність та легкість управління даними проекту.

У подальших розділах проекту буде детальніше розглянуто і реалізовано вибрані архітектурні підходи, концепти розробки та структуру бази даних з метою досягнення поставлених цілей та реалізації потреб проекту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ

3.1 Аналіз і вибір засобів реалізації системи автоматизованого проектування моделей нейромереж

Для програмної реалізації системи, в якості основної платформи для розробки API обрано .NET, враховуючи його широкі можливості та популярність у сфері веб-розробки.

.NET є однією з найбільш розповсюджених платформ для розробки серверних додатків. Вона надає різноманітні фреймворки та інструменти, що спрощують процес розробки, підтримують широкий спектр мов програмування, забезпечують високу продуктивність та безпеку. Обираючи .NET, ми отримуємо доступ до розширеного набору бібліотек і сервісів, що дозволяють ефективно розробляти та підтримувати серверну частину проекту.

Окрім цього, .NET забезпечує хорошу масштабованість і здатність до інтеграції з іншими технологіями та системами. Це важливо для нашого проекту, оскільки ми плануємо інтегрувати систему автоматизованого проектування моделей нейронних мереж з іншими компонентами інфраструктури. Будучи одним з провідних фреймворків, .NET дозволяє нам ефективно взаємодіяти з іншими системами та сервісами, що сприяє створенню комплексного рішення для нашого проекту [11].

Отже, обираючи .NET як основну платформу для розробки API серверної частини, ми отримуємо потужний набір інструментів та можливостей, що дозволять нам ефективно реалізувати функціональність проекту, забезпечуючи стабільну та масштабовану роботу системи.

Для роботи з нейронними мережами в проекті обрано фреймворк Tensorflow разом з об'єктною обгорткою Keras. Однак, оскільки Tensorflow та Keras не підтримують мову програмування C# напряму, було необхідно знайти рішення для інтеграції цих фреймворків з платформою .Net. Для цього використано Nuget пакет Keras.Net, який дозволяє писати код на мові C# і виконувати його за допомогою Python інтерпретатора, що виконується в фоновому режимі [12].

Для проектування бази даних в даному проекті обрано Microsoft SQL Server, існує кілька обгрунтувань цього вибору.

По-перше, Microsoft SQL Server є одним з провідних реляційних Систем Управління Базами Даних (СУБД) на ринку. Вона відома своєю надійністю, широким спектром функцій та підтримкою великих обсягів даних. Використання Microsoft SQL Server дозволить нам створити структуровану базу даних зі зв'язками між таблицями та забезпечити цілісність даних.

По-друге, Microsoft SQL Server має багатий набір інструментів для розробки, адміністрування та оптимізації бази даних. Вона надає зручні засоби для створення та моделювання схеми бази даних, написання запитів SQL, розгортання бази даних і забезпечення безпеки даних. Це дозволить нам ефективно працювати з базою даних і забезпечити її оптимальну продуктивність.

По-третє, Microsoft SQL Server є добре підтримуваним продуктом з великою спільнотою користувачів і широкими можливостями для розвитку. Ми можемо розраховувати на підтримку та допомогу з боку Microsoft, а також мати доступ до багатьох ресурсів, документації та онлайн-форумів, що дозволяють нам вирішувати потенційні проблеми та розширювати функціональність бази даних [13].

Хоча існують інші альтернативи реляційних та нереляційних баз даних, Microsoft SQL Server обрано з урахуванням його надійності, функціональності, підтримки та розширюваності. Він надає нам потужні засоби для ефективної організації та управління даними, що є критичним аспектом успішної розробки проекту бази даних.

Після успішної реалізації основної логіки у серверній частині, настала потреба у створенні веб-клієнта, який забезпечував би зручну та ефективну взаємодію з додатком. У процесі вибору технології для розробки веб-інтерфейсу була обрана бібліотека React для JavaScript з вагомих причин.

React є потужним інструментом для розробки інтерактивних веб-інтерфейсів, здатним забезпечити швидкість, продуктивність та масштабованість додатку. Ця бібліотека дозволяє розбити веб-інтерфейс на компоненти, які можна повторно

використовувати та легко управляти. Такий підхід дозволяє ефективно організувати код та полегшує розробку, тестування та підтримку додатку у майбутньому.

Один з головних переваг React полягає у використанні віртуального DOM (Document Object Model), що дозволяє знизити навантаження на реальний DOM та оптимізувати продуктивність додатку. React самостійно визначає, які частини веб-сторінки потребують оновлення, та ефективно маніпулює віртуальним DOM, щоб забезпечити мінімальну кількість змін, які потрібно вносити до реального DOM. Це дозволяє зменшити витрати на операції з DOM та підвищити продуктивність додатку в цілому [14].

Крім того, React має велику та активну спільноту розробників, що забезпечує доступ до багатьох готових компонентів, бібліотек та інструментів, які полегшують розробку веб-додатків. Це значно прискорює процес розробки та допомагає уникнути повторного винайдення велосипеда, використовуючи готові рішення та найкращі практики спільноти.

Отже, обрання бібліотеки React для розробки веб-клієнта, обґрунтовано його потужним функціоналом, ефективним використанням віртуального DOM та доступністю розширеної спільноти розробників. Це дозволяє забезпечити швидку, масштабовану та легко управляєму розробку інтерактивних веб-інтерфейсів для зручного користування додатком.Веб-клієнт складається з чотирьох основних сторінок, які забезпечують різноманітні функції і можливості додатку.

3.2 Програмування API веб інтерфейсу системи

Вирішено, що додаток буде мати клієнт серверну архітектуру. Тобто буде веб додаток, та серверна частина, яка буде містити основну бізнес логіку додатку. В свою чергу, серверна частина буде мати монолітну архітектуру.

Створено проект, який складається з 7-ми підпроектів, рисунок 3.1:

- API – точка входу сервеного додатку, містить API методи;
- Data – проект, що описує інтерфейст спілкування з базою даних;
- Database – проект, містить SQL скрипти створення таблиць;

- Infrastructure – проект, що реалізує інтерфейси роботи з планувальником задач та файловим сховищем;
- Domain – реалізує базову бізнес логіку проекту;
- Orchestrator – проект, який реалізує інтерфейси роботи з базовими моделями, проміжна ланка між API та рештою проекту;
- Model – описує основні моделі та інтерфейси взаємодії всіх модулів проетку.

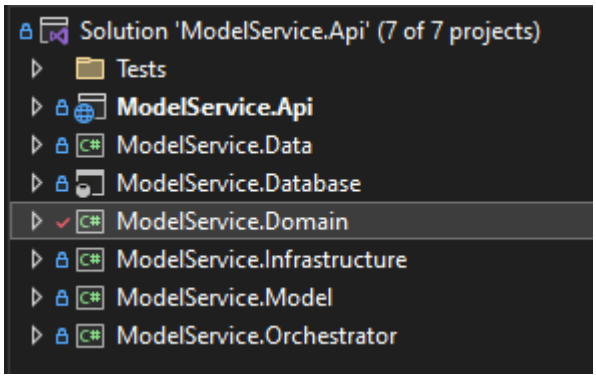

Рисунок 3.1 – Структура проекту

Загалом ідея звязків підпроектів полягає у тому, що усі проекти посилаються лише на Model і не знає один про одного. Пов’язання інтерфейсів та їх реалізацій відбувається за допомогою використання інферсії залежностей. У кожному проекті є клас ProjectNameModule.cs, в якому ми описуємо які інтерфейси реалізують класи в даному проекті, приклад InfrastructureModule.

В проекті Infrastructure, реалізовано основний функціонал роботи з планувальником задач, в якості якого, обрано Hangfire, та файловго сховища. А також опис глобальної виключення для обробки помилок, рисунок 3.2.

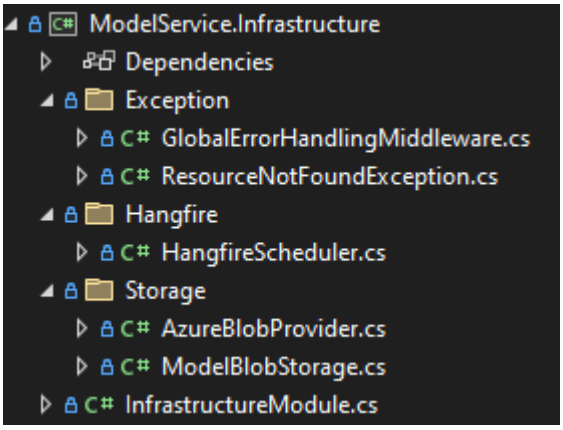

Рис. 3.2 – Зв’язки між проектами

Також налаштовано підключення до бази даних для роботи з планувальником задача та створено Azure Blob Storage та встановлено зв’язок з ним.

Для забезпечення зручного спілкування з додатком розроблено веб API, яке дозволяє веб-клієнту взаємодіяти з сервером. У рамках проекту Api створено 5 контролерів, в яких реалізовані основні API методи для взаємодії з системою:

DataController - цей контролер відповідає за спілкування з сервером щодо даних. Він надає методи для отримання даних з сервера, створення нових даних, оновлення існуючих даних та видалення даних.

ExecutionController - цей контролер призначений для запуску процесу навчання. Він надає методи, які дозволяють клієнту ініціювати процес навчання моделей.

FitResultController - цей контролер відповідає за роботу з моделями результатів навчання. Він надає методи для отримання результатів навчання моделей, збереження результатів та взаємодії з ними.

ModelsController - цей контролер використовується для роботи з моделями. Він надає методи для отримання доступних моделей, створення нових моделей, оновлення існуючих моделей та видалення моделей.

SessionsController - цей контролер призначений для роботи з сесіями. Він надає методи для створення нових сесій, отримання деталей про сесії, оновлення сесій та видалення сесій.

Детальний список методів кожного контролера можна знайти в документації Swagger, яка зображенна у додатку А. На даний момент цей список містить мінімально необхідний набір методів для створення базового функціоналу. На рисунку 3.3 зоображено структура API проекту, в якому знаходяться відповідно контролери, та точка входу – Program.cs та Startup.cs – файл, який кофігурує сервіси, та реалізує зв’язку інтерфейсів та їх реалізацію.

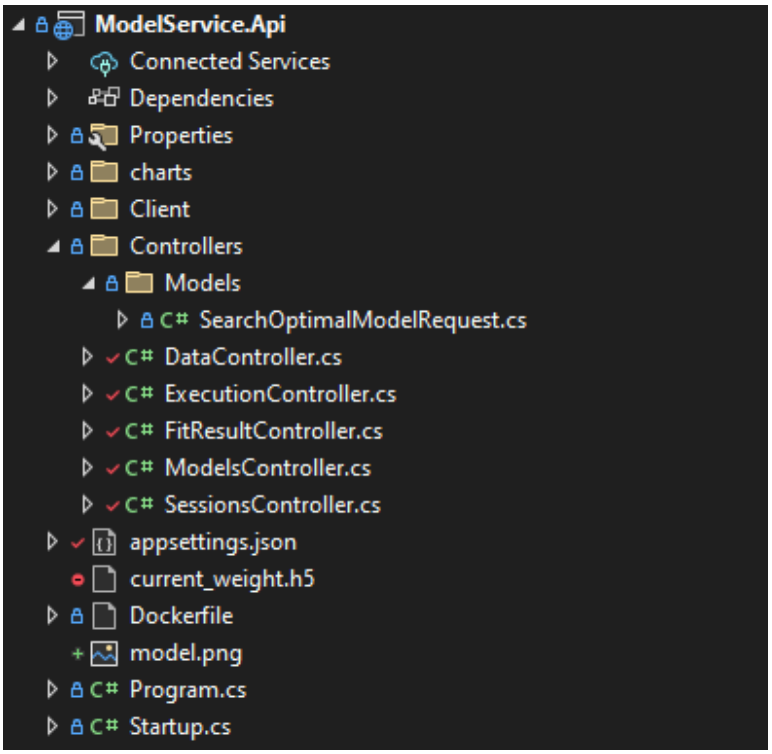

Рисунок 3.3 – Структура Orchestrator

Крім того, у проекті Orchestrator реалізовано базові проміжні сервіси:

- DataOrchestrator – даний сервіс відповідає за взаємодію з файловою системою. Він забезпечує функціонал для зчитування та запису даних у файлову систему.

- CsvOrchestrator – виконує операції зчитування Csv файлів. Він надає можливість системі зчитувати дані з файлів у форматі Csv та використовувати їх для подальшої обробки.

- FitResultOrchestrator – сервіс, який взаємодіє з репозиторієм результатів навчання. Він виконує операції збереження результатів навчання та взаємодії з базою даних. Також він відповідає за валідацію даних, пов'язаних з результатами навчання.

- ModelOrchestrator – цей сервіс забезпечує взаємодію з репозиторієм моделей. Він виконує операції збереження, оновлення та взаємодії з моделями у базі даних. Крім того, він відповідає за валідацію даних, пов'язаних з моделями.

- SessionOrchestrator – сервіс взаємодіє з репозиторієм сесій. Він виконує операції збереження, оновлення та взаємодії з сесіями у базі даних. Він також відповідає за валідацію даних, пов'язаних з сесіями.

- FitFlowOrchestrator – сервіс, який взаємодіє запускає прокцес генерації та навчання мереж, зокрема звертається до планувальника задач, та створює в ньому відповідні задачі, а саме задачі створення моделей та їх навчання.

Ці проміжні сервіси розширюють функціонал проекту Orchestrator та забезпечують необхідну логіку для взаємодії з файловою системою та базою даних у контексті відповідних об'єктів додатку. Також в даних сервісах відбувається основна валідація, та відслідковування виключень, що дозволяє при помилці, легко віднайти її та зрозуміти причину. Вміст та структура проекту зображена на рисунку 3.4.

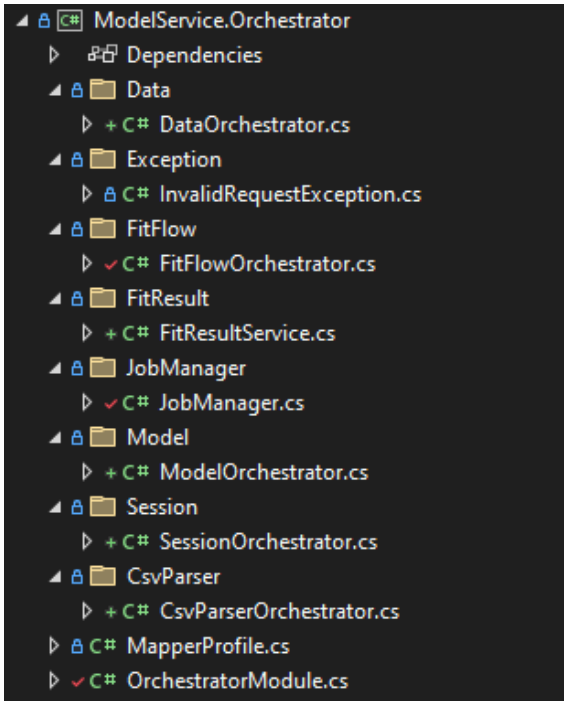

Рисунок 3.4 – Структура Orchestrator

Проект Data є ключовим компонентом програмного забезпечення, який відповідає за реалізацію репозиторіїв для зв'язку з базою даних. Використовуючи

DbContext EF (Entity Framework), цей проект забезпечує ефективний та зручний спосіб доступу до даних для сутностей Session, Model та FitResult.

У проекті Data реалізовані репозиторії, які використовують DbContext EF для взаємодії з базою даних. Репозиторій для сутності Session дозволяє зберігати, отримувати, оновлювати та видаляти об'єкти сесій. Це дає можливість ефективно керувати сесіями в додатку та виконувати операції з ними, такі як створення нових сесій або отримання списку існуючих.

Репозиторій для сутності Model забезпечує можливість зберігання та отримання моделей, пов'язаних з конкретною сесією. Це важливий аспект додатку, оскільки моделі використовуються для генерації та навчання. Завдяки репозиторію можна ефективно керувати моделями, додавати нові моделі до сесій, оновлювати їх параметри та отримувати доступ до існуючих моделей для подальшого використання.

Репозиторій для сутності FitResult дозволяє зберігати та отримувати дані про результати навчання моделей. Це важлива інформація для аналізу та оцінки продуктивності моделей. Завдяки репозиторію можна зручно зберігати та отримувати дані про точність та втрати кожної навченої моделі. Це допомагає виробникам додатків зрозуміти, як добре працюють їх моделі та вносити необхідні зміни для поліпшення результатів.

Проект Data забезпечує надійний та ефективний доступ до бази даних за допомогою DbContext EF для сутностей Session, Model та FitResult. Це дозволяє розробникам зосередитися на функціональності додатку, не витрачаючи зайвий час на роботу з базою даних. З використанням цих репозиторіїв можна зручно керувати сесіями, моделями та результатами навчання, що робить процес розробки та підтримки додатку ефективним та зручним. Структура проекту зображено на рисунку 3.5.

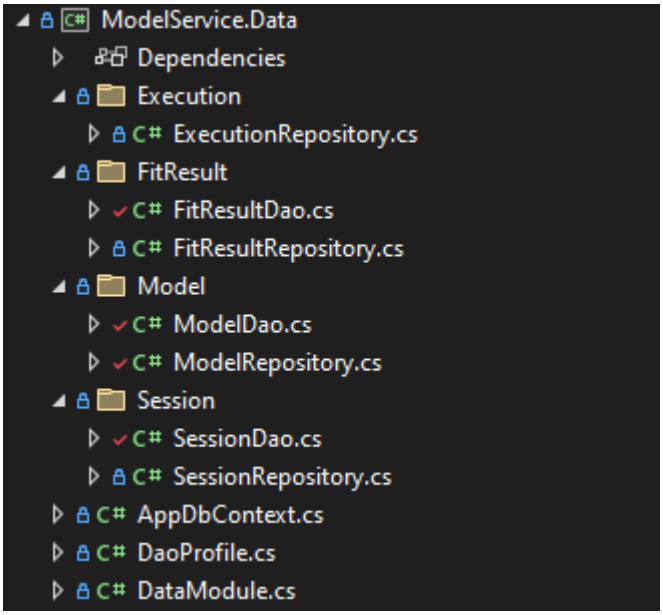

Рисунок 3.5 – Структура Data

Загалом основна логіка міститься в проектах Data, Infrastructure, Domain та Orchestrator. Діаграма класів цих проектів наведено на кресленику ІК93.100БАК.005 Д4. Model – описує лише інтерфейси та моделі, при цьому не рписуючи жодних алгоритмів. API – містить контролери, які адресуються до проекту Orchestrator.

3.3 Створення бази даних системи

Для повноцінної роботи додатку спроектовано базу даних. В програмі буде використовуватися 3 основних моделів – це сессія, модель та результат навчання. Отже, створено 3 таблиці: Session, Model та FitResult. В ієрархій моделей на вищому рівні зниходиться сесія, потім модель і на найнижчому рівні результат навчання. Таким чинном в сесії може бути декілька моделей, які описують певну задачу, а в моделі може бути декілька результатів навчання. SQL скрипт створення таблиць навдено в додатку А.

Для реалізації спілкування додатку з базою даних використовувався Entity Framework. Створено репозиторії для кожної моделів, в яких описано основні CRUD операції, які необхідні будуть в проекті.

3.4 Інтеграція бібліотек інструментальних засобів реалізації системи

З метою забезпечення зручності та структурованості роботи з Keras, створено клас KerasCore.cs у проекті Domain. Цей клас містить опис основних операцій з моделями нейронних мереж та інкапсулює логіку Keras від решти проекту. Це дозволяє відокремити неконтрольований Python код від основної частини проекту на мові C#, що спрощує управління та налагодження коду. Крім того, ізольовання неконтрольованого коду в одному класі сприяє виявленню та усуненню помилок, оскільки можна легко локалізувати джерело проблеми. Інтерфейс, який реалзує KerasCore, зображено на рисунку 3.6, реалізація наведена у додатку Б.

```
using ModelService.Model.Models;
using ModelService.Model.ResultModels;

namespace ModelService.Model.Keras;

/// <summary> IKerasCore.
4 references
public interface IKerasCore
{
    /// <summary> Creates the model.
    2 references
    Result<string> CreateModel(ModelConfig modelConfig);

    /// <summary> Compiles the model.
    1 reference
    Result<string> CompileModel(string sequential, CompileConfig compilationConfig);

    /// <summary> Fits the model.
    2 references
    Result<FitHistory> FitModel(string modelJson, FitConfig config, CompileConfig compileConfig, TrainData trainData);

    /// <summary> Predicts the specified model.
}
```

Рисунок 3.6 – Інтерфейс IKerasCore.cs

Цей підхід дозволяє розробникам зосередитися на реалізації логіки моделей нейронних мереж та використовувати підтримувані засоби для роботи з Keras. Клас KerasCore.cs дозволяє зручно викликати функції та методи Keras у контрольованому середовищі мови C#, зменшуючи ризик помилок і спрощуючи взаємодію з фреймворком Tensorflow та Keras [15].

Також реалізовано алгоритми автоматизації, та реалізації методів генерації створення та навчання моделей нейромереж. Основна ідея проекту полягає в

автоматизованому створенні моделей та їх навчанні. Для досягнення цієї мети розроблено кілька основних алгоритмів, які відіграють ключову роль в проекті.

По-перше, розроблено алгоритм генерації моделей з випадковою кількістю нейронів у внутрішній моделі в певному діапазоні. Це дозволяє отримати різноманітні моделі з різними розмірами та складністю, що важливо для експериментування та пошуку оптимальних структур моделей.

По-друге, реалізовано алгоритм, який додає створення, компіляцію та навчання моделей до планувальника задач. Цей алгоритм дозволяє систематизувати та автоматизувати процес створення та навчання моделей, розподіляючи ресурси та контролюючи послідовність виконання задач.

Для реалізації цих алгоритмів створено клас FitFlow у проекті Domain. Цей клас містить основну реалізацію логіки генерації моделей, їх створення в Keras, збереження в базі даних та файловому сховищі, а також навчання створених моделей та збереження результатів навчання та навчених ваг.

Клас FitFlow забезпечує інтерфейс для взаємодії з моделями, навчання та збереження їх результатів. Він включає методи, такі як CreateModelAsync, який створює конкретну та випадкову Keras модель, FitModelAsync, який здійснює навчання моделі, та GenerateRandomModelConfig, який генерує конфігурацію випадкової моделі.

Перший метод “CreateModelAsync” приймає вхідні параметри “sessionId”, “modelId”, “modelRangeConfig” та “model”. Він викликає метод “GenerateRandomModelConfig” з переданим “modelRangeConfig” для отримання випадкової конфігурації моделі. Якщо генерація конфігурації успішна, то метод викликає метод “CreateModelAsync” з отриманою випадковою конфігурацією та “model” для створення моделі. Якщо створення моделі в базі даних успішне, метод повертає успішний результат зі створеною моделлю. Якщо виникають помилки під час генерації випадкової конфігурації або створення моделі, метод повертає помилковий результат з відповідним повідомленням про помилку.

Друга метод “CreateModelAsync” також приймає вхідні параметри “sessionId”, “modelId”, “modelConfig” та “model”. Він викликає метод “_kerasCore.CreateModel”

з переданим “modelConfig” для створення моделі за допомогою фреймворку Keras. Якщо створення моделі успішне, то метод оновлює властивості “modelId”, “modelConfig” та “modelConfigPath” об’єкта “model”. Далі, він викликає метод “_modelService.UpdateModelAsync” для оновлення моделі в базі даних. Якщо оновлення моделі в базі даних успішне, метод зберігає конфігурацію моделі у файловому сховищі та зберігає зображення моделі. На кінці, метод повертає успішний результат з оновленою моделлю. Якщо виникають помилки під час створення моделі, оновлення моделі в базі даних або збереження файлів, метод повертає помилковий результат з відповідним повідомленням про помилку.

Метод “FitModelAsync” виконує процес навчання моделі на основі вхідних даних. Він приймає параметри “sessionId”, “modelId”, “compileConfig”, “fitConfig” і “trainData”.

Спочатку метод отримує конфігурацію створеної моделі з файлового сховища за допомогою “_storage.GetFileAsync”. Після цього відбувається читання вмісту файлу у форматі JSON за допомогою “StreamReader” і “ReadToEndAsync”. Отриманий JSON представляє конфігурацію моделі.

Наступним кроком є виклик методу “_kerasCore.FitModel” з отриманою конфігурацією моделі, “fitConfig”, “compileConfig” та “trainData”. Метод “_kerasCore.FitModel” виконує процес навчання моделі з використанням вхідних даних і повертає результат у вигляді історії навчання.

Якщо навчання моделі пройшло успішно, метод створює об'єкт “FitResult”, який містить інформацію про результати навчання, такі як “modelId”, “Accuracy”, “Loss”, “HistoryPath” і “WeightPath”. Об'єкт “FitResult” зберігається в базі даних за допомогою “_fitResultService.CreateFitResultAsync”.

Далі, метод зберігає ваги навченої моделі та історію навчання в файлове сховище. Загальний шлях до збережених файлів формується “sessionId” та унікального ідентифікатора.

У кінці метод повертає успішний результат, якщо всі операції пройшли успішно. У разі виникнення помилки під час виконання будь-якої операції, метод повертає помилковий результат з відповідним повідомленням про помилку.

Метод “GenerateRandomModelConfig” генерує випадкову конфігурацію моделі на основі переданої конфігурації діапазонів “modelRangeConfig”. Метод приймає параметр “modelRangeConfig”, який містить інформацію про межі діапазонів для створення випадкової конфігурації моделі.

У методі створюється екземпляр генератора випадкових чисел “random” з використанням поточного часу.

Далі виконується ітерація через кількість внутрішніх шарів моделі, від 0 до “modelRangeConfig.CountOfInternalLayers”. В кожній ітерації створюється об'єкт “Layer”, який представляє шар моделі. У цьому об'єкті встановлюються наступні властивості: “ActivationFunc” (функція активації), “NeuronsCount” (кількість нейронів у шарі, випадково вибирається з діапазону від “modelRangeConfig.NeuronsInLayerMin” до “modelRangeConfig.NeuronsInLayerMax”), та “UseBias” (використовувати або не використовувати зміщення, значення встановлюється з “modelRangeConfig.UseBias”). Створений шар додається до списку “internaLayers”.

Після цього створюється об'єкт “createConfig” типу “ModelConfig”, який містить структуру конфігурації моделі. У цьому об'єкті встановлюються наступні властивості: “InputShape” (форма вхідних даних, встановлюється як масив з одного елементу – “modelRangeConfig.InputCount”), “InternalLayers” (список внутрішніх шарів, встановлюється зі згенерованого списку “internaLayers”), та “OutputLaye” (вихідний шар моделі, встановлюється зі специфікованої конфігурації в “modelRangeConfig.OutputCount”).

Наприкінці метод повертає успішний результат зі створеною випадковою конфігурацією моделі “createConfig” у вигляді об'єкта ”Result<ModelConfig>”. Код метода наведено в додатку В.

Окрім цього, в проекті Orchestrator створено клас JobManager.cs, наведено у додатку Д, який реалізує інтерфейс IJobManager.cs. Цей клас відповідає за планування задач для створення та навчання моделей. Основна логіка цього класу полягає у запланування створення та навчання n моделей. Тобто для навчання n моделей, спочатку планується n задач для створення кожної моделі, а потім ще n

задач для навчання кожної з них. Це певно є основною частиною, системи, адже вона забезпечує основну логукі автоматизації. Блок схема алгоритмів планування задач зображено на кресленику ІК93.100БАК.005 Д3.

Ці алгоритми та класи узгоджуються та співпрацюють між собою, створюючи систему, що дозволяє автоматизувати процес створення, навчання та експериментування з моделями нейронних мереж. Використання цих алгоритмів та класів спрощує розробку, збереження та навчання моделей, забезпечуючи ефективне використання ресурсів та прискорення процесу прогнозування.

3.5 Опис інтерфейсу веб клієнта системи

Сторінка “Список існуючих сесій” зображено на рисунку 3.7. На цій сторінці відбувається отримання з бази даних усіх доступних сесій навчання. Сесії відображаються у вигляді таблиці з трьома колонками: ідентифікатор сесії, назва та кількість моделей, що належать до кожної сесії. Крім того, в кожному рядку таблиці є дві кнопки - одна для переходу на сторінку з результатами навчання даної сесії, а інша для переходу на сторінку налаштування та початку навчання моделей. Додатково, зверху сторінки розташована кнопка, що дозволяє перейти на сторінку створення нової сесії навчання.

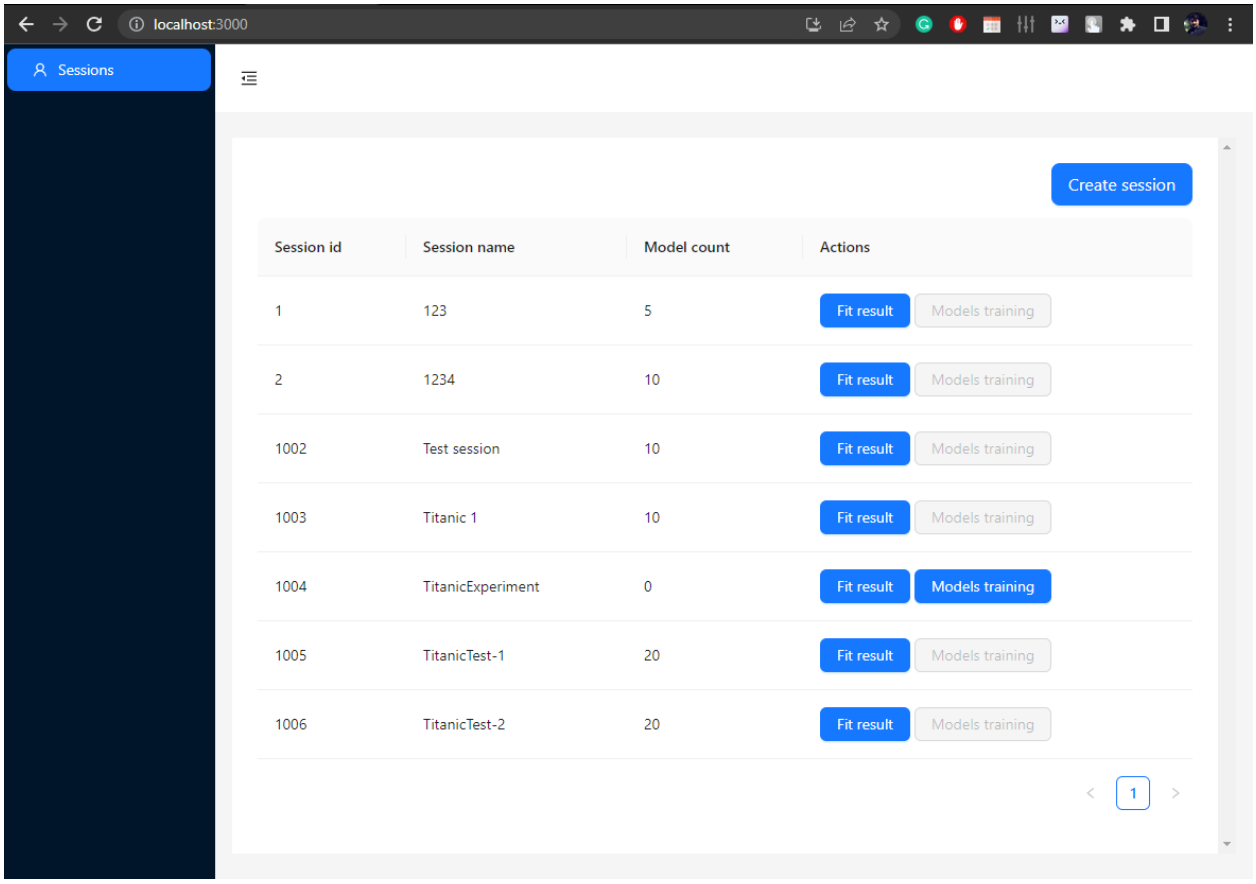

Рисунок 3.7 – Список існуючих сесій

Сторінка “Створення сесії”, зображено на рисунку 3.8. На цій сторінці розміщена форма, яка містить одне поле для введення назви сесії. Користувач може ввести назву моделі, а потім натиснути кнопку "Створити сесію", щоб створити нову сесію навчання.

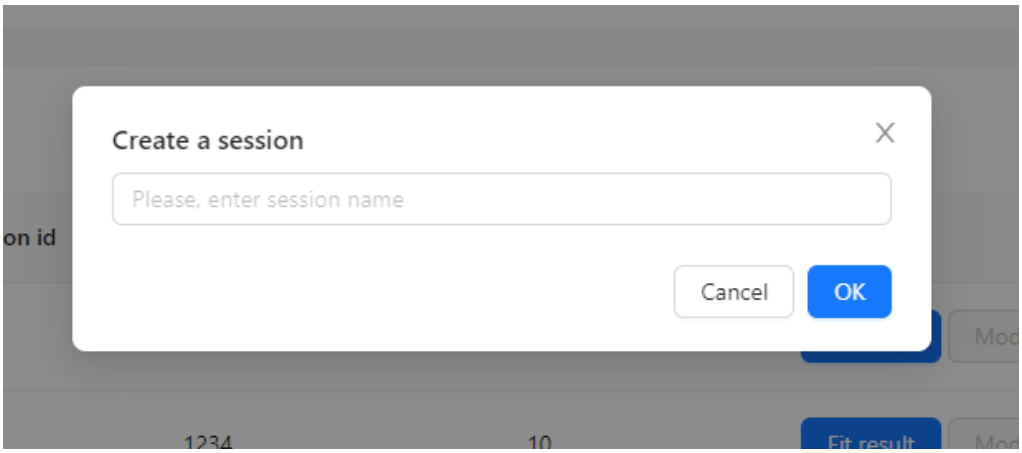

Рисунок 3.8 – Створення сесії

Сторінка “Налаштування сесії” зображена на рисунку 3.9. Ця сторінка призначена для налаштування конфігурацій генерації моделей та їх навчання для кожної окремої сесії. Користувач може потрапити на цю сторінку, натиснувши кнопку на сторінці списку сесій. На цій сторінці знаходиться форма, до якої можна завантажити CSV-файл з датасетом, а також текстове поле для введення конфігурації у форматі JSON. Крім того, на сторінці розташована кнопка, яка запускає процес створення та навчання моделей для обраної сесії.

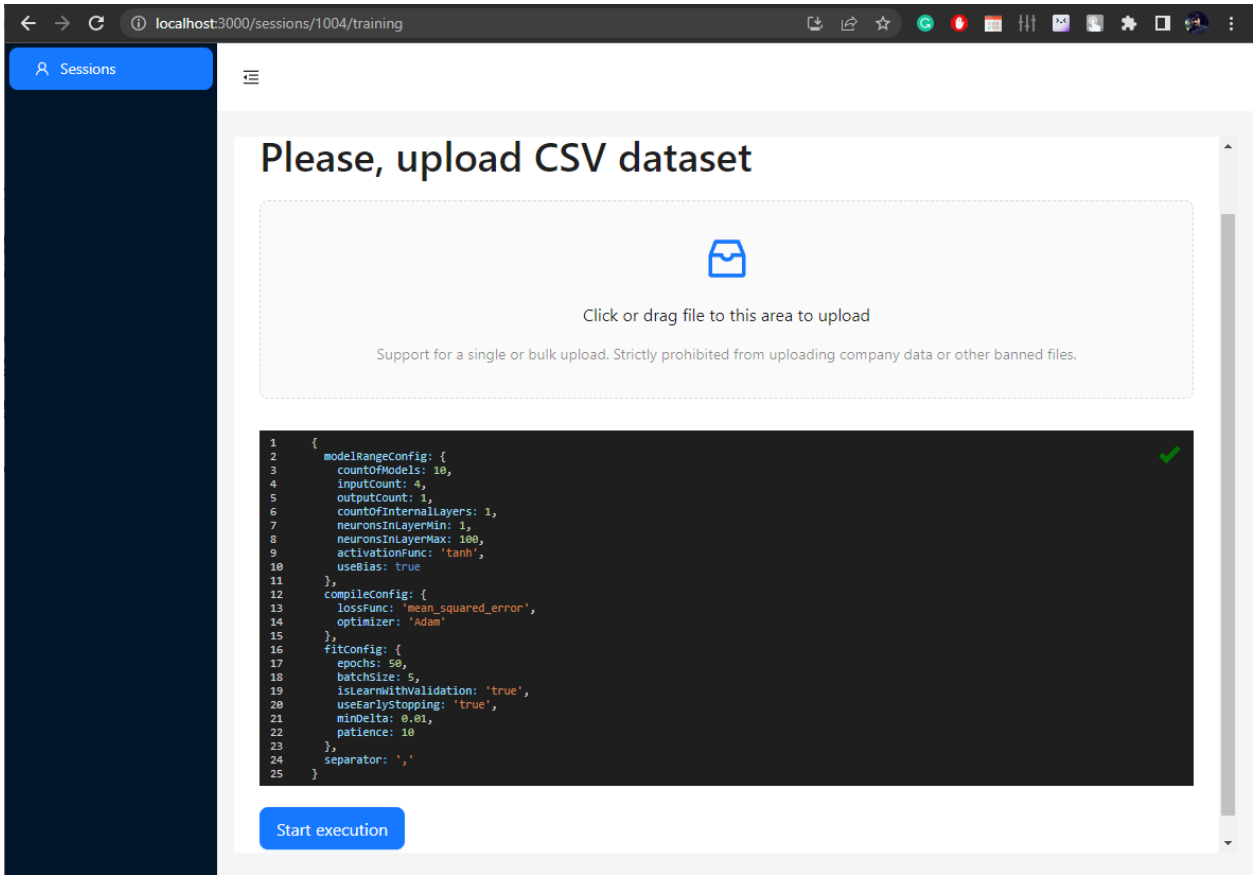

Рисунок 3.9 – Налаштування сесії

Сторінка “Результати навчання” зображено на рисунку 3.10. Ця сторінка призначена для відображення результатів навчання моделей. Користувач може потрапити на цю сторінку, натиснувши кнопку “FitResult” біля відповідної сесії на сторінці списку сесій. У верхній частині сторінки відображаються дві гістограми: одна показує точність моделей, а інша - втрати (помилки) для кожної навченої моделі. Нижче розташовані блоки, кількість яких відповідає кількості згенерованих

та навчених моделей. Кожен блок містить результати окремої моделі, а також два графіки: перший відображає зміну точності моделі в процесі навчання залежно від кількості епох, а другий - втрати моделі. Дана панель зображена на рисунку 3.11.

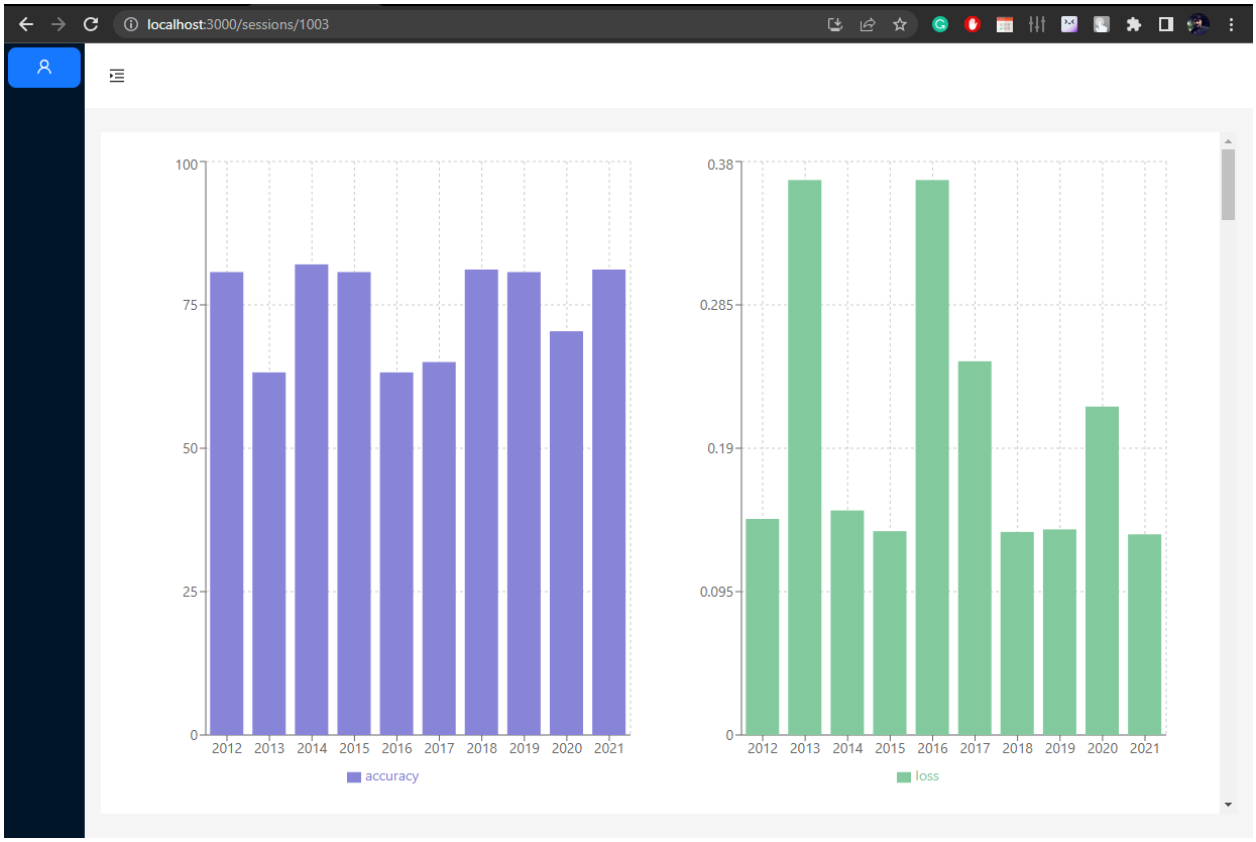

Рисунок 3.10 – Результати навчання

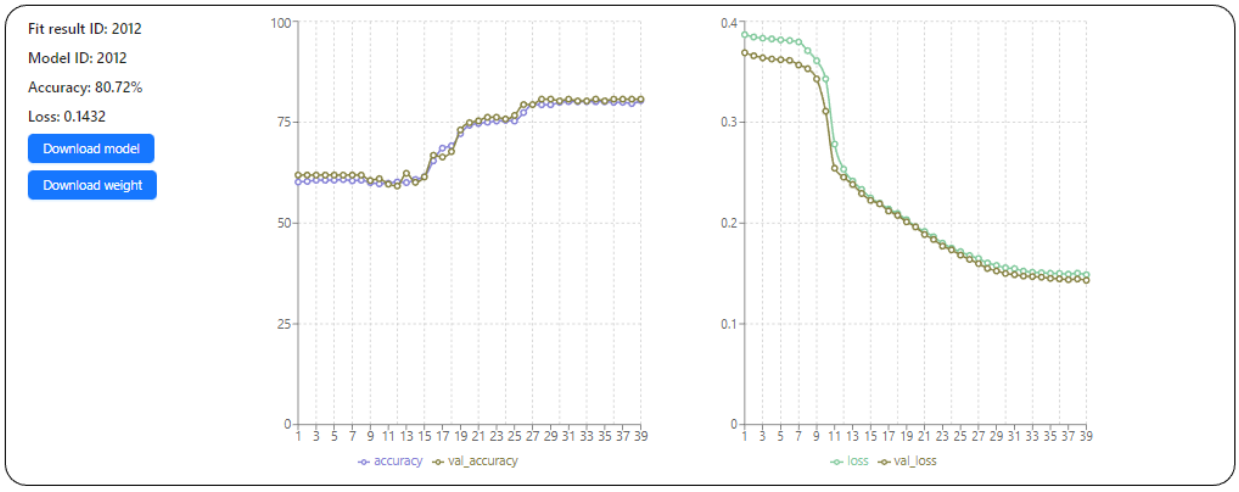

Рисунок 3.11 – Результати навчання моделі

Загалом, веб-клієнт додатку забезпечує користувачам зручність та широкі можливості для керування сесіями навчання моделей та відображення їх результатів. Використання бібліотеки React дозволяє створити ефективний та інтуїтивно зрозумілий інтерфейс для зручної роботи з додатком.

3.6 Експеримент та аналіз результатів

Для проведення експерименту обрано задачу про Титанік. Задача про Титанік є однією з відомих задач машинного навчання і базується на реальних подіях. У цій задачі потрібно прогнозувати, які пасажири виживуть під час катастрофи на лайнері "Титанік". Використовуються різні ознаки, такі як вік, стать, клас кабіни, наявність сімейних членів тощо, для побудови моделі, яка зможе передбачити, чи виживе пасажир.

Метою експерименту є дослідження ефективності різних моделей машинного навчання в розв'язанні задачі про титанік. Основні задачі експерименту включають:

- Автоматизоване проектування та тренування різних моделей машинного навчання для прогнозування виживання пасажирів на "Титаніку".
- Оцінку ефективності цих моделей на основі відповідних метрик.
- Виявлення найбільш ефективної моделі для задачі про титанік.

Задача про титанік є популярною у галузі машинного навчання, оскільки вона пропонує цікаву задачу класифікації з чіткими вхідними ознаками. Крім того, наявність реальних даних про подію на "Титаніку" дозволяє перевірити ефективність моделей на практичному прикладі. Це робить задачу про титанік привабливою для проведення експерименту з автоматизації проектування навчання моделей нейромереж. Розмір датасету – 891 запис.

Початковий датасет, має 12 колонок значень, з них 11 навчальних та один, який містить бінарне значення, вижив пасажир – чи ні:

- PassangerId – унікальний ідентифікатор пасажира;
- PClass – клас квитка, який вказує на соціально-економічний статус пасажира;
- Name – повне ім'я пасажира;

- Sex – стать пасажира;
- Age – вік пасажира;
- SibSp – кількість братів, сестер або подружжя пасажира, що були на борту;
- Parch – кількість батьків або дітей пасажира, що були на борту;
- Ticket – номер квитка;
- Fare – вартість квитка;
- Cabin – номер кабіни, де перебував пасажир;
- Embarked – порт посадки пасажира;
- Survived – вижив пасажир чи ні (1 – вижив, 0 – не вижив).

В результаті аналізу датасету, вирішено прибрати деякі колонки, які мало впливають на виживання, або ті, які важко піддаються обробці для максимального спрощення задачі. Таким чином прибрано наступні поля: PassangerId, Name, Age, Ticket, Fare, Cabin, Embarked. Таким чином нейронна мережа повинна приймати на вхід 4 параметра: Pclass, Sex, SibSp та Parch. Вихідне значення одне – одиниця або нуль (вижив, не вижив) [16].

Для провдення експерименту створено сесію “TitanicTest-2”, рисунок 3.12 та 3.13.

Рисунок 3.12 – Створення сесії

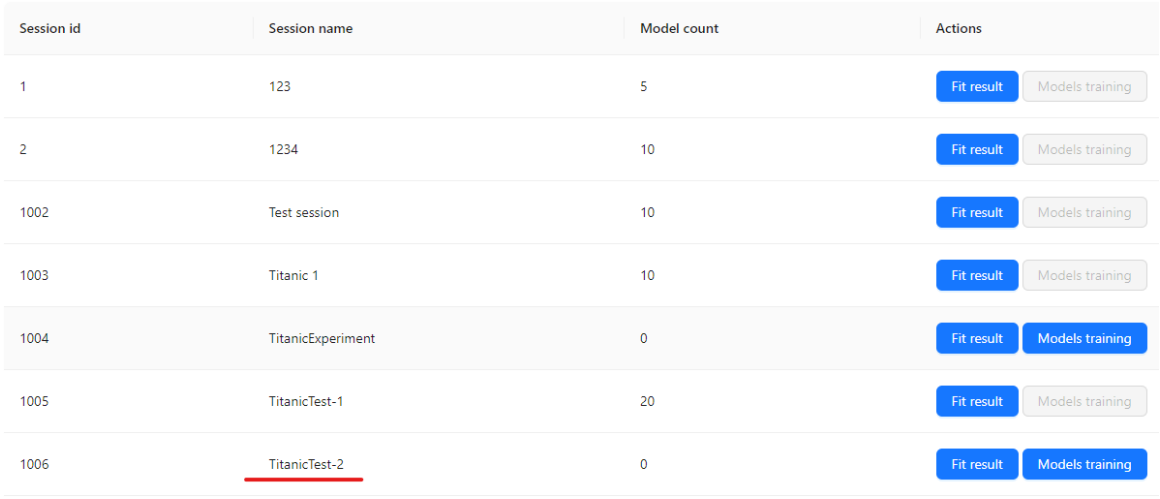

Рисунок 3.13 – Створена сесія

Створена сесія, початково має 0 моделей, отже її можна використовувати для проектування. Початкова конфгурація для проектування та навчання має параметри, що зображені на рисунку 3.14. Як видно з конфігурації, буде згенеровано 20 моделей, модель буде мати 4 входи та 1 вихід, 3 внутрішніх шари в кожному з яких може бути від 1 до 8 нейронів. Варто зазначити, що рекомендована кількість нейронів в одному шарі не повинна бути меншою за суму кількості входів та виходів, проте, при виборі значення мінімальної кількості нейронів, це правило вирішено порушити, для того, щоб була вірогідність генерації відверто невдалих моделей для їх порівняння з вдалими моделями. З інших параметрів, активаційна функція – гіперболічний тангенс, також не використовувати нейрон зміщення.

Щодо парметрів компіляції, то оборано в якості функції втрат середньоквадратичну помилку, та алгорим оптимізації “Adam” - метод стохастичного градієнтного спуску, який базується на адаптивній оцінці моментів першого та другого порядку.

Параметри навчання, мають такі значення, як максимальна кількість епох - 100, розмір партій навчання – по 10 за крок, також дані повинні ділитися на тренувальні та валідаційні у співвідношенні, програмно задано фіксоване значення - 75% тренувальні, 25 валідаційні, а також буде використовуватися рання зупинка навчання за виконання одної з двох умов:

- Різниця значення між епохами буде менша ніж 0.01;
- 10 епох поспіль не буде покращення в ефективності моделі (працює тільки якщо виористовуються валідаційні дані).

Окрім того, в систему завантажено датасет в форматі CSV - “train.csv”. Після задання навчального датасету та конфігурації проектування, необхідно запустити навчання.

```
{
  modelRangeConfig: {
    countOfModels: 20,
    inputCount: 4,
    outputCount: 1,
    countOfInternalLayers: 2,
    neuronsInLayerMin: 1,
    neuronsInLayerMax: 8,
    activationFunc: 'tanh',
    useBias: true
  },
  compileConfig: {
    lossFunc: 'mean_squared_error',
    optimizer: 'Adam'
  },
  fitConfig: {
    epochs: 100,
    batchSize: 10,
    isLearnWithValidation: 'true',
    useEarlyStopping: 'true',
    minDelta: 0.01,
    patience: 10
  },
  separator: ','
}
```

Рисунок 3.14 – Конфігурація для проектування

Генерація та та навчання 20 моделей зайняла 54 секунди. Після чого на сторінці “FitResult” можна перегалянути рузультати. З гістограми точності моделей на тестових даних видно, що зображено на рисунку 3.15, що найвищу точність має модель з ідентифікатором 2049 – 82.6%, а найнижчу 2044 – 60.09%. Наступна гістограма втрат, з якої добре видно, залежність з гістограмою точності, оскільки модель 2049, в даному випадку, має значення функції втрат на тестових даних 0.13, а 2044, яка має найнижчу точність – 0.22, що є найбільшим значенням серед всіх, рисунок 3.16. Далі переглянемо більш детальну інформацію для найкращої та нійгіршої моделі. З графіку точності, рисунко 3.17, видно, що максимальна точність настає на 15 епосі, поісля чого через 10 епох навчання зупинилось, тобто спрацював фактор передчасної зупинки, після 10 епох без підвищення ефективності. Проте по

графіку втрат, рисунок 3.18, видно, що навіть після 15 епохи, втрати продовжували зменшуватися, при цьому точність перестала збільшуватися, це можна пояснити тим, що або модель почала поступово перенавчатися, або датасет занадто малий, для того щоб зафіксувати подальше збільшення точності.

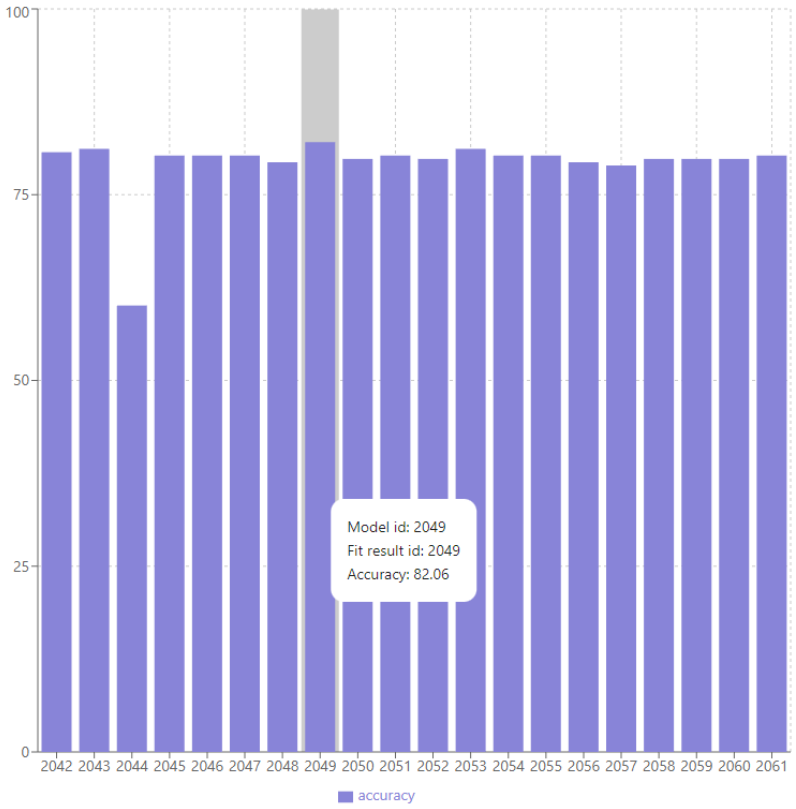

Рисунок 3.15 – Гістограма точності навчених моделей

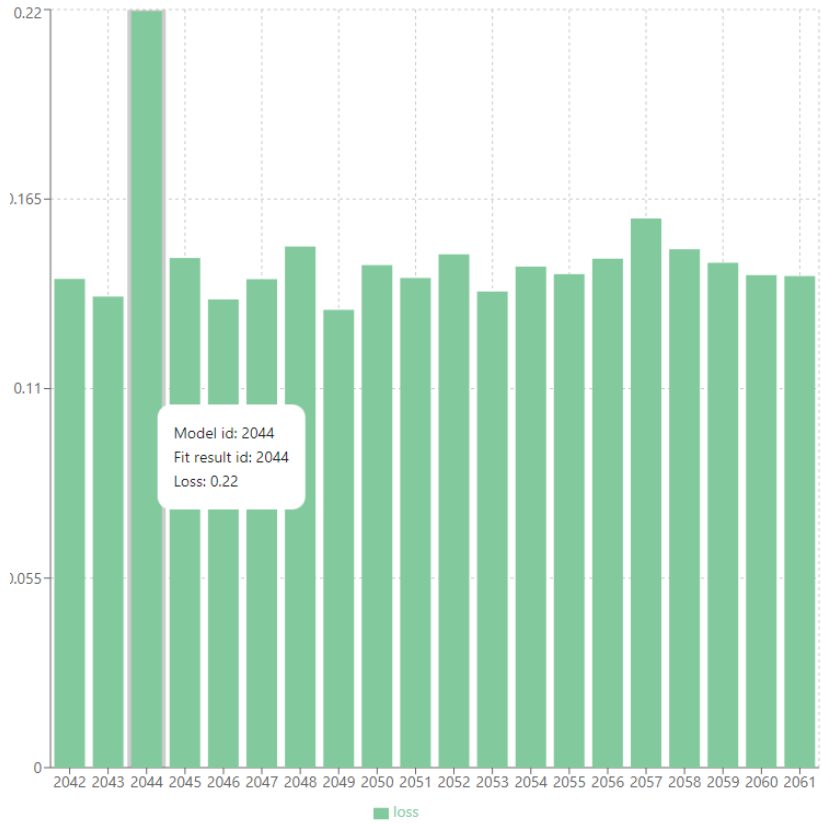

Рисунок 3.16 – Гістограма точності навчених моделей

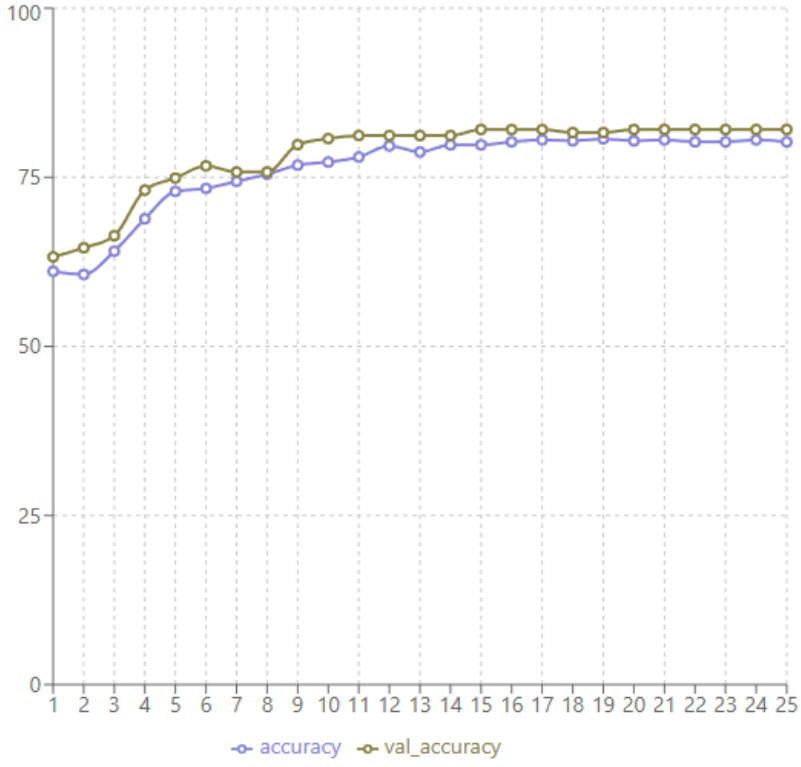

Рисунок 3.17 – Графік точності моделі 2049

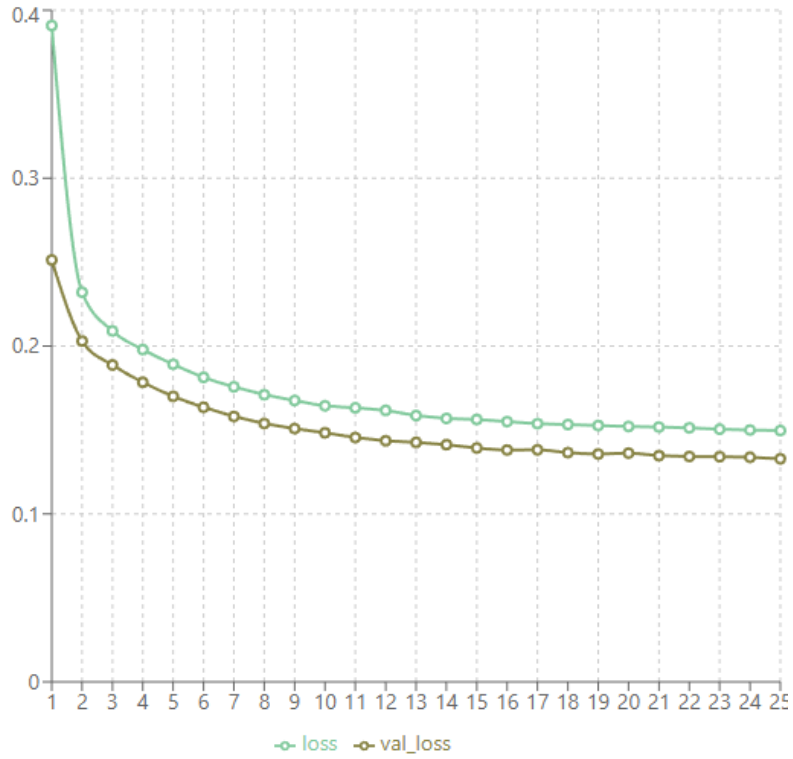

Рисунок 3.18 – Графік втрат моделі 2049

Також проаналізуємо графіки найменш ефективної моделі з ідентифікатором 2044. Як видно з рисунку 3.19, точність з самого початку не змінюється до 25 епохи, навчання не зупиняється після 10 епохи без покращення, тому що умова зупинки по втратам не спрацювала до того моменту. Після 25 епохи, точність трохи зросла, а потім почала різко падати, після чого навчання було автоматично зупинено. Це можна побачити по графіку втрат моделі 2044, який зображено на рисунку 3.20. Хоча втрати зменшуються, але дуже повільно, зрештою різниця між втрат між епохами досягає граничного значення 0.01 і на вчання зупиняється.

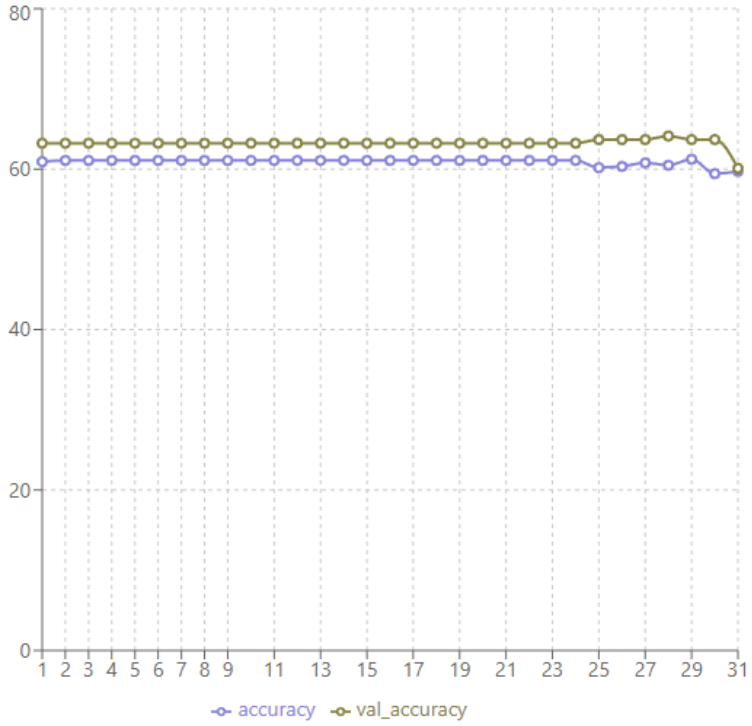

Рисунок 3.19 – Графік точності моделі 2044

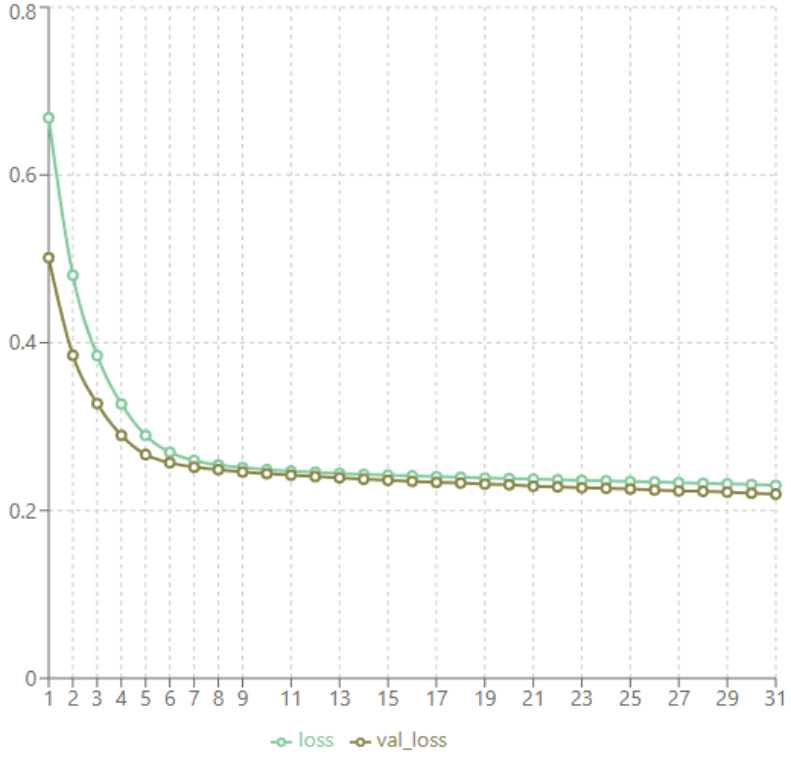

Рисунок 3.20 – Графік втрат моделі 2044

Проведений експеримент з генерації та навчання 20 моделей на задачі про титанік показав, що модель з ідентифікатором 2049 має найвищу точність (82.6%), тоді як модель 2044 є найменш ефективною (60.09%). Аналіз графіків підтвердив залежність точності від кількості епох навчання. Зауважено, що модель 2049 досягла максимальної точності на 15 епосі, але після 10 епох без покращення навчання було зупинено. У випадку моделі 2044, точність не змінювалася до 25 епохи, після чого різко знизилася. Ці результати можуть вказувати на перенавчання моделі або обмежену кількість даних, проте основною причиною поганих результатів моделі 2044 є невдало зегенорована архітектура.

Таким чином розроблено систему автоматизованого проеткування моделей нейронних мереж. В процесі розробки, програмно реалізовано серверну частину, а саме веб API та алгоритми генерації моделей, інтегровано Keras в .Net, як засіб роботи з нейромережами. Створено базу даних на базі MS SQL Server та створено відповідні таблиці, для підтримки роботи системи. Налаштовано інфраструктуру проекту, а саме роботу з Azure Blob Storage – файлове сховище у хмарі, та сервіси для роботи з планувальником задач – Hangfire. А також розроблено веб клієнт на базі бібліотеки React. Також проведено експериментальне дослідження розробленої системи. Для експерименту обрано просту задачу прогнозування шансу виживання на Титаніку. Загалом згенеровано та навчено 20 моделей. Отримані результати проаналізовано та знайтено найефективнішу та найнеефективнішу модель.

ВИСНОВКИ

Виконано аналіз проблеми проектування моделей нейронних мереж для задач прогнозування, який дозволив встановити основні виклики та недоліки існуючих підходів до проектування моделей. Було встановлено, що одним із головних обмежень є необхідність ручного налаштування архітектури моделі, що вимагає значних зусиль та експертного досвіду.

На основі отриманих результатів аналізу, була розроблена архітектура автоматизованої системи проектування моделей нейронних мереж. Ця система використовує генетичний алгоритм для генерації моделей з різною кількістю нейронів у внутрішніх шарах. Потім ці моделі навчаються на тренувальних даних, і їх ефективність оцінюється за допомогою метрик точності та втрат.

У третьому розділі було реалізовано програмну реалізацію системи та проведено експериментальне дослідження. Генерація та навчання 20 моделей з різною конфігурацією зайняли 54 секунди. З результатів експерименту були побудовані гістограми точності та втрат моделей на тестових даних. З гістограми точності було видно, що найвищу точність має модель з ідентифікатором 2049 - 82.6%, а найнижчу точність - модель 2044 з точністю 60.09%. З графіку втрат було помітно, що навіть після досягнення максимальної точності, втрати продовжували зменшуватися.

Проведене експериментальне дослідження показало високу ефективність розробленої системи в генерації та навчанні моделей нейронних мереж для задач прогнозування. Результати експерименту демонструють, що система здатна забезпечувати високу точність прогнозів на тестових даних. Подальші дослідження та вдосконалення системи можуть привести до ще кращих результатів та розширення її застосування в інших задачах прогнозування.

Результати виконаного проекту підтверджують важливість автоматизації процесу проектування моделей нейронних мереж для задач прогнозування. Розроблена система відповідає вимогам, поставленим у роботі, та демонструє потенціал для подальшого впровадження та застосування в реальних сценаріях.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Neural Network: Architecture, Components & Top Algorithms [Електронний ресурс] – режим доступу до ресурсу: https://www.upgrad.com/blog/neural-network-architecture-components-algorithms/ (дата звернення: 01.04.2023)
2. Машинне навчання простими словами. Частина 1 [Електронний ресурс] – режим доступу до ресурсу: http://www.mmf.lnu.edu.ua/ar/1739 (дата звернення: 12.04.2023)
3. Open Source AutoML Tools: AutoGluon, TransmogrifAI, Auto-sklearn, and NNI [Електронний ресурс] – режим доступу до ресурсу: https://web.archive.org/web/20201202120843/https://www.bizety.com/2020/06/16/open-source-automl-tools-autogluon-transmogrifai-auto-sklearn-and-nni/ (дата звернення: 14.04.2023)
4. Neural Architecture Search: Everything You Need to Know [Електронний ресурс] – режим доступу до ресурсу: https://deci.ai/neural-architecture-search/ (дата звернення: 16.04.2023)
5. 3 NeuroEvolution of Augmenting Topologies (NEAT) [Електронний ресурс] – режим доступу до ресурсу: https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume21/stanley04a-html/node3.html (дата звернення: 01.05.2023)
6. Common web application architectures [Електронний ресурс] – режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата звернення: 03.05.2023)
7. Design a DDD-oriented microservice [Електронний ресурс] – режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/ddd-oriented-microservice (дата звернення: 06.05.2023)
8. Hexagonal Architecture, there are always two sides to every story [Електронний ресурс] – режим доступу до ресурсу: https://medium.com/ssense-tech/hexagonal-architecture-

there-are-always-two-sides-to-every-story-bc0780ed7d9c (дата звернення: 08.05.2023)

9. What Are Single Page Applications and Why Do People Like Them So Much? [Електронний ресурс] – режим доступу до ресурсу: https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application?spz=learn_var (дата звернення: 15.05.2023)
10. SQL Server And Relational Database - Part One [Електронний ресурс] – режим доступу до ресурсу: https://www.c-sharpcorner.com/article/sql-server-and-relational-database-part-one/ (дата звернення: 07.06.2023)
11. Tutorial: Create a web API with ASP.NET Core [Електронний ресурс] – режим доступу до ресурсу: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio (дата звернення: 08.06.2023)
12. Keras.NET [Електронний ресурс] – режим доступу до ресурсу: https://scisharp.github.io/Keras.NET/ (дата звернення: 08.06.2023)
13. What is Microsoft SQL Server and what is it for? [Електронний ресурс] – режим доступу до ресурсу: https://intelequia.com/en/blog/post/what-is-microsoft-sql-server-and-what-is-it-for (дата звернення: 08.06.2023)
14. ReactDOM [Електронний ресурс] – режим доступу до ресурсу: https://uk.legacy.reactjs.org/docs/react-dom.html (дата звернення: 10.06.2023)
15. Keras API [Електронний ресурс] – режим доступу до ресурсу: https://keras.io/api/ (дата звернення: 01.05.2023)
16. How the Titanic helped us think about Explainable AI [Електронний ресурс] – режим доступу до ресурсу: https://www.ibm.com/blog/how-the-titanic-helped-us-think-about-explainable-ai/ (дата звернення: 09.06.2023)

ДОДАТКИ

ДОДАТОК А

SQL Скрипт створення таблиць бази даних

```
CREATE TABLE [dbo].[Sessions](
	[id] [int] NOT NULL PRIMARY KEY IDENTITY(1,1),
	[session_name] [nvarchar](32) NOT NULL,
	[model_count] [int] NOT NULL,
	[is_trained] [bit] NOT NULL,
	[dataset_path] [nvarchar](128) NULL
) ON [PRIMARY]

GO

CREATE TABLE [dbo].[Models](
	[id] [int] NOT NULL PRIMARY KEY IDENTITY(1,1),
	[model_config] [nvarchar](2048) NOT NULL,
	[model_config_path] [nvarchar](100) NULL,
	[session_id] [int] FOREIGN KEY REFERENCES [dbo].[Sessions]([id])
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[FitResults](
	[id] [int] NOT NULL PRIMARY KEY IDENTITY(1,1),
	[accuracy] [float] NULL,
	[loss] [float] NULL,
	[history_path] [nvarchar](100) NOT NULL,
	[weight_path] [nvarchar](100) NOT NULL,
	[model_id] [int] FOREIGN KEY REFERENCES [dbo].[Models]([id])
) ON [PRIMARY]

GO
```

ДОДАТОК Б

Реалізація інтерфейсу IKerasCore

```
using KerasLib = Keras;
using Keras.Callbacks;
using Keras.Layers;
using Keras.Models;
using ModelService.Model.Keras;
using ModelService.Model.Models;
using ModelService.Model.ResultModels;
using Numpy;
using Python.Runtime;
using Keras;

namespace ModelService.Domain.Keras;

public class KerasCore : IKerasCore
{
   public KerasCore()
   {
      Python.Runtime.PythonEngine.Initialize();
      Python.Runtime.PythonEngine.BeginAllowThreads();

      using (Py.GIL())
      {
         PythonEngine.RunSimpleString("tf.config.list_physical_devices('GPU')");
      }
   }

   public Result<string> CompileModel(string sequential, CompileConfig compileConfig)
   {
      var jsonModel = string.Empty;
      using (Py.GIL())
      {
         var model = BaseModel.ModelFromJson(sequential);
         model.Compile(
            optimizer: compileConfig.Optimizer,
            loss: compileConfig.LossFunc);

         jsonModel= model.ToJson();
      }

      return Result<string>.FromSuccess(jsonModel);
   }

   public Result<string> CreateModel(ModelConfig modelConfig)
   {
      var jsonModel = string.Empty;

      using (Py.GIL())
      {
         var model = new Sequential();

         model.Add(new Input(
            shape: new Shape(modelConfig.InputShape),
            name: "input"));

         foreach (var layer in modelConfig.InternalLayers)
```

```
        {
            model.Add(new Dense(
                units: layer.NeuronsCount,
                use_bias: layer.UseBias,
                activation: layer.ActivationFunc));
        }

        model.Add(new Dense(
            units: modelConfig.OutputLayer.NeuronsCount,
            use_bias: modelConfig.OutputLayer.UseBias,
            activation: modelConfig.OutputLayer.ActivationFunc));

        KerasLib.Utils.Util.PlotModel(model, "model.png", true, true);

        jsonModel = model.ToJson();
      }

      return Result<string>.FromSuccess(jsonModel);
    }

    public Result<FitHistory> FitModel(string modelJson, FitConfig config, CompileConfig compileConfig, TrainData
trainData)
    {
      var fitHistory = new FitHistory();

      using (Py.GIL())
      {
        var ndArrayX = np.array<float>(trainData.XTrain).reshape(trainData.CountOfLines, trainData.CountOfInputs);
        var ndArrayY = np.array<float>(trainData.YTrain).reshape(trainData.CountOfLines, trainData.CountOfOutputs);

        var trainX = ndArrayX;
        var trainY = ndArrayY;
        NDarray valX = null;
        NDarray valY = null;

        Callback[] callbacks = null;

        if (config.UseEarlyStopping)
        {
          callbacks = new Callback[]
          {
            new KerasLib.Callbacks.EarlyStopping(
              min_delta: config.MinDelta.Value,
              patience: config.Patience.Value,
              monitor: config.IsLearnWithValidation ? "val_loss" : "loss")
          };
        }

        float valSplitCoeficient = 0.75f;

        if (config.IsLearnWithValidation)
        {
          trainX = np.split(ndArrayX, new int[] { Convert.ToInt32(ndArrayX.len * valSplitCoeficient) }).First();
          valX = np.split(ndArrayX, new int[] { Convert.ToInt32(ndArrayX.len * valSplitCoeficient) }).Last();
          trainY = np.split(ndArrayY, new int[] { Convert.ToInt32(ndArrayY.len * valSplitCoeficient) }).First();
          valY = np.split(ndArrayY, new int[] { Convert.ToInt32(ndArrayY.len * valSplitCoeficient) }).Last();
        }

        var model = BaseModel.ModelFromJson(modelJson);
```

```
            model.Compile(
                optimizer: compileConfig.Optimizer,
                loss: compileConfig.LossFunc,
                metrics: new string[] { "accuracy" });

            var history = model.Fit(
                x: trainX,
                y: trainY,
                validation_data: valX == null || valY == null ? null : new NDarray[] { valX, valY },
                epochs: config.Epochs,
                batch_size: config.BatchSize,
                callbacks: callbacks);

            var result = model.Evaluate(
                x: config.IsLearnWithValidation ? valX : trainX,
                y: config.IsLearnWithValidation ? valY : trainY);

            fitHistory.Loss = Math.Round(result[0], 5);
            fitHistory.Accuracy = Math.Round(result[1], 5);
            fitHistory.History = history.HistoryLogs;
            fitHistory.Epoches = history.Epoch;

            model.SaveWeight("current_weight.h5");
        }

        return Result<FitHistory>.FromSuccess(fitHistory);
    }
}
```

ДОДАТОК В

Реалізація інтерфейсу IFitFlow

```
using ModelService.Domain.Constants;
using ModelService.Model.Constants;
using ModelService.Model.FitFlow;
using ModelService.Model.FitResult;
using ModelService.Model.Keras;
using ModelService.Model.Model;
using ModelService.Model.Models;
using ModelService.Model.ResultModels;
using ModelService.Model.Services;
using ModelService.Model.Storage;
using Newtonsoft.Json;
using System.Text;

namespace ModelService.Domain.FitFlow;

class FitFlow : IFitFlow
{
   private readonly IAzureBlobProvider _storage;
   private readonly IKerasCore _kerasCore;
   private readonly IModelOrchestrator _modelService;
   private readonly IFitResultOrchestrator _fitResultService;

   public FitFlow(
      IAzureBlobProvider storege,
      IKerasCore kerasCore,
      IModelOrchestrator modelService,
      IFitResultOrchestrator fitResultService)
   {
      _storage = storege;
      _kerasCore = kerasCore;
      _modelService = modelService;
      _fitResultService = fitResultService;
   }

   public async Task<Result<Model.Model.Model>> CreateModelAsync(int sessionId, int modelId,  ModelRangeConfig
modelRangeConfig, Model.Model.Model model)
   {
      try
      {
         var randomModelResult = GenerateRandomModelConfig(modelRangeConfig);

         if (!randomModelResult.IsSuccessful)
         {
            return Result<Model.Model.Model>.FromError(randomModelResult.Message);
         }

         var modelResult = await CreateModelAsync(sessionId, modelId, randomModelResult.Entity, model);

         if (!modelResult.IsSuccessful)
         {
            return Result<Model.Model.Model>.FromError($"Model creation in db error: '{modelResult.Message}'");
         }

         return Result<Model.Model.Model>.FromSuccess(modelResult.Entity);
      }
```

```
        catch (System.Exception e)
        {
            return Result<Model.Model.Model>.FromUnexpectedError(e.Message);
        }
    }

    public async Task<Result<Model.Model.Model>> CreateModelAsync(int sessionId, int modelId, ModelConfig
modelConfig, Model.Model.Model model)
    {
        try
        {
            var createdModel = _kerasCore.CreateModel(modelConfig);

            if (!createdModel.IsSuccessful)
            {
                return Result<Model.Model.Model>.FromError($"Keras model creation error: '{createdModel.Message}'");
            }

            model.ModelId = modelId;
            model.ModelConfig = JsonConvert.SerializeObject(modelConfig);
            model.ModelConfigPath =
$"{GeneralConstants.BaseStorageDir}/session_{sessionId}/created/model_{modelId}.json";

            var modelUpdateResult = await _modelService.UpdateModelAsync(model);

            if (!modelUpdateResult.IsSuccessful)
            {
                return Result<Model.Model.Model>.FromError($"Model creation in db error:
'{modelUpdateResult.Message}'");
            }

            using (var file = new MemoryStream(Encoding.UTF8.GetBytes(createdModel.Entity)))
            {
                await _storage.PutFileAsync("models", model.ModelConfigPath, file, true);
            }

            var modelPngPath = $"{GeneralConstants.BaseStorageDir}/session_{sessionId}/img/model_{modelId}.png";

            using (var file = new StreamReader("model.png"))
            {
                file.BaseStream.Position = 0;
                await _storage.PutFileAsync("models", modelPngPath, file.BaseStream, true);
            }

            return Result<Model.Model.Model>.FromSuccess(modelUpdateResult.Entity);
        }
        catch (System.Exception e)
        {
            return Result<Model.Model.Model>.FromUnexpectedError(e.Message);
        }
    }

    public async Task<Result> FitModelAsync(
        int sessionId,
        int modelId,
        CompileConfig compileConfig,
        FitConfig fitConfig,
        TrainData trainData)
    {
        try
```

```
    {
      var createdModelFile = await _storage.GetFileAsync("models",
$"{GeneralConstants.BaseStorageDir}/session_{sessionId}/created/model_{modelId}.json");
      using var reader = new StreamReader(createdModelFile);
      var modelJson = await reader.ReadToEndAsync();

      var fitHistory = _kerasCore.FitModel(modelJson, fitConfig, compileConfig, trainData);

      if (!fitHistory.IsSuccessful)
      {
        return Result.FromError($"Fit error: '{fitHistory.Message}'.");
      }

      var fitResult = new FitResult
      {
        ModelId = modelId,
        Accuracy = fitHistory.Entity.Accuracy,
        Loss = fitHistory.Entity.Loss,
        HistoryPath =
$"{GeneralConstants.BaseStorageDir}/session_{sessionId}/fit_results/fit_results_{Guid.NewGuid()}.json",
        WeightPath = $"{GeneralConstants.BaseStorageDir}/session_{sessionId}/trained/weight_{Guid.NewGuid()}.h5",
      };

      await _fitResultService.CreateFitResultAsync(fitResult);

      using (var file = new StreamReader("current_weight.h5"))
      {
        await _storage.PutFileAsync("models", fitResult.WeightPath, file.BaseStream, true);
      }

      using (var file = new MemoryStream(Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(fitHistory.Entity))))
      {
        file.Position = 0;
        await _storage.PutFileAsync("models", fitResult.HistoryPath, file, true);
      }

      return Result.FromSuccess();
    }
    catch (System.Exception e)
    {
      return Result.FromUnexpectedError(e.Message);
    }
  }

  public Result<ModelConfig> GenerateRandomModelConfig(ModelRangeConfig modelRangeConfig)
  {
    var random = new Random(DateTime.UtcNow.Millisecond);

    var internaLayers = new List<Layer>();

    for (int layerNumber = 0; layerNumber < modelRangeConfig.CountOfInternalLayers; layerNumber++)
    {
      var layer = new Layer()
      {
        ActivationFunc = modelRangeConfig.ActivationFunc,
        NeuronsCount = random.Next(modelRangeConfig.NeuronsInLayerMin, modelRangeConfig.NeuronsInLayerMax)
+ 1,
        UseBias = modelRangeConfig.UseBias.Value,
      };
```

```
            internaLayers.Add(layer);
        }

        var createConfig = new ModelConfig
        {
            InputShape = new int[] { modelRangeConfig.InputCount },
            InternalLayers = internaLayers,
            OutputLayer = new Layer
            {
                ActivationFunc = modelRangeConfig.ActivationFunc,
                NeuronsCount = modelRangeConfig.OutputCount,
                UseBias = false,
            },
        };

        return Result<ModelConfig>.FromSuccess(createConfig);
    }
}
```

ДОДАТОК Д

Реалізація планування JobManager.cs

```
using System;
using ModelService.Model.Execution;
using ModelService.Model.JobManager;
using System.Threading.Tasks;
using Hangfire;
using ModelService.Model.FitFlow;
using ModelService.Model.Services;
using ModelService.Model.Models;
using ModelService.Model.ResultModels;
using Newtonsoft.Json;
using ModelService.Model.Session;
using System.Collections.Generic;
using ModelService.Model.Model;

namespace ModelService.Orchestrator.JobManager
{
   public class JobManager : IJobManager
   {
      private readonly IFitFlow _fitFlow;
      private readonly IModelOrchestrator _modelService;

      public JobManager(
         IFitFlow fitFlow,
         IModelOrchestrator modelService)
      {
         _fitFlow = fitFlow;
         _modelService = modelService;
      }

      public async Task<Job> QueueAsync(FitFlowExecution execution, int sessionId)
      {
         var job = await Task.Run(async () =>
         {
            var modelsStack = await CreateAndAddToDbRandomModelAsync(execution.ModelRangeConfig, sessionId);

            var model = modelsStack.Pop();

            var jobId = BackgroundJob.Schedule(() => _fitFlow.CreateModelAsync(sessionId, model.ModelId,
execution.ModelRangeConfig, model), TimeSpan.FromMilliseconds(100));

            for (int i = 1; i < execution.ModelRangeConfig.CountOfModels; i++)
            {
               var model1 = modelsStack.Pop();

               jobId = BackgroundJob.ContinueJobWith(jobId, () => _fitFlow.CreateModelAsync(sessionId, model1.ModelId,
execution.ModelRangeConfig, model1),
               JobContinuationOptions.OnlyOnSucceededState);
            }

            var modelsResult = await _modelService.GetModelBySessionAsync(sessionId);

            foreach (var modelItem in modelsResult.Entity)
            {
               jobId = BackgroundJob.ContinueJobWith(jobId, () =>
```

```
                _fitFlow.FitModelAsync(
                    sessionId,
                    modelItem.ModelId,
                    execution.CompileConfig,
                    execution.FitConfig,
                    execution.TrainData),
                JobContinuationOptions.OnlyOnSucceededState);
            }

            return new Job { JobId = jobId };
        });

        return job;
    }

    private async Task<Stack<ModelService.Model.Model.Model>>
CreateAndAddToDbRandomModelAsync(ModelRangeConfig modelRangeConfig, int sessionId)
    {
        var modelList = new List<ModelService.Model.Model.Model>();

        for (int i = 0; i < modelRangeConfig.CountOfModels; i++)
        {
            var modelCofig = _fitFlow.GenerateRandomModelConfig(modelRangeConfig);

            modelList.Add(new ModelService.Model.Model.Model
            {
                ModelConfig = JsonConvert.SerializeObject(modelCofig),
                SessionId = sessionId,
            });
        }

        var modelResult = await _modelService.CreateModelRangeAsync(modelList);

        if(!modelResult.IsSuccessful)
        {
            throw new System.Exception(modelResult.Message);
        }

        var getModelResult = await _modelService.GetModelBySessionAsync(sessionId);

        if (!getModelResult.IsSuccessful)
        {
            throw new System.Exception(getModelResult.Message);
        }

        return new Stack<ModelService.Model.Model.Model>(getModelResult.Entity);
    }
  }
}
```