

Національний Технічний Університет України
«Київський Політехнічний Інститут імені Ігоря Сікорського»
Міністерство освіти і науки України
Національний Технічний Університет України
«Київський Політехнічний Інститут імені Ігоря Сікорського»
Міністерство освіти і науки України

Кваліфікаційна наукова
праця на правах рукопису

ПАЛАДІЄВ ОЛЕКСАНДР ОЛЕГОВИЧ

УДК 004.65

**МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ ДЛЯ ВИРІШЕННЯ
ЗАДАЧІ КЛАСИФІКАЦІЇ НА ОСНОВІ ТРИВИМІРНИХ
НЕЙРОННИХ МЕРЕЖ**

121 Інженерія програмного забезпечення

12 Інформаційні технології

Подається на здобуття наукового ступеня доктора філософії

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело _____ Паладієв О.О.

Науковий керівник Лісовиченко Олег Іванович, к.т.н., доцент

Київ – 2025

АНОТАЦІЯ

Паладієв О.О. Методи та програмні засоби для вирішення задачі класифікації на основі тривимірних нейронних мереж. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 121 – Інженерія програмного забезпечення з галузі знань 12 – Інформаційні технології. – Національний Технічний Університет України «Київський Політехнічний Інститут імені Ігоря Сікорського», Київ,, 2025.

Дисертаційна робота присвячена розробці та дослідженню спеціалізованих програмних засобів для ефективної класифікації тривимірних зображень з можливістю їх інтеграції в різноманітні автоматизовані процеси. Основною метою дослідження є вдосконалення методів обробки та аналізу тривимірних даних шляхом застосування передових підходів до проектування архітектур нейронних мереж і розробки адаптивних алгоритмів оптимізації. У ході виконання роботи досягнуто наступних результатів:

Вперше запропоновано використання опонентної кольорової системи як методу попередньої обробки даних. Запропонований метод підвищує інформативність ознак і покращує їхню інтерпретацію нейронними мережами, що суттєво зменшує втрати при класифікації. Додатково, досліджено особливості роботи опонентної системи в умовах різного рівня зашумленості вхідних даних, що дозволило розробити стратегії її адаптації до специфічних наборів зображень.

Вперше розроблено топологію нейронних зв'язків яка реалізує локально-обмежені структури зв'язків з протилежними нейронами та їх безпосередніми сусідами в тривимірному просторі. Запропонована топологія поєднує розширювальні та звужувальні шари, що сприяє ефективному вилученню ознак. Особлива увага приділялася оптимізації параметрів цих шарів для забезпечення їхньої гнучкості й адаптивності під різні типи даних, а також визначенню

оптимальної кількості нейронів і типів функцій активації для покращення навчання мережі.

Вперше розроблено методи зміни кількості нейронних зв'язків. Ці підходи забезпечують підвищення швидкодії програмних засобів класифікації, за рахунок видалення слабких зв'язків та підвищення точності класифікації за рахунок генерації нових зв'язків для нейронів з вагами, наближеними до меж функції активації. Розроблено та протестовано кілька методів адаптації, включаючи методи поступового усічення ваг і додавання нових зв'язків для підвищення стійкості моделі до варіативності вхідних даних.

Розроблено програмні засоби для забезпечення доступу до функціоналу класифікації тривимірних зображень, який дозволяє інтегрувати можливості класифікації в різні автоматизовані процеси. Реалізоване рішення підтримує масштабованість обробки великих обсягів даних, включаючи функції попередньої обробки, аналізу результатів і моніторингу продуктивності.

Здійснено детальне тестування продуктивності розроблених програмних засобів на основі двох тривимірних наборів даних. Результати експериментів підтвердили високу точність і продуктивність розробленого підходу, що забезпечує його конкурентоспроможність у вирішенні задач автоматизованої класифікації складних тривимірних структур.

Таким чином, створений метод класифікації та програмні засоби є ефективним рішенням для класифікації тривимірних зображень, поєднуючи новітні архітектури нейронних мереж і алгоритми оптимізації для досягнення високих показників точності та продуктивності. Запропонований підхід демонструє високу гнучкість і адаптивність, що дозволяє його застосування в широкому спектрі прикладних задач, включаючи медичну діагностику, промислову автоматизацію та аналіз наукових даних.

Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 212 сторінок, з яких 168 сторінок основного тексту. Робота містить 50 рисунків, 3 таблиці та 80 літературних джерел.

Ключові слова: інженерія програмного забезпечення, аналіз даних, автоматизація, автоматизація моделювання нейронних мереж, автоматизована обробка, глибоке навчання, класифікація, машинне навчання, мета-навчання, методи машинного навчання, нечітка логіка, оцінка ефективності, попередня обробка, розпізнавання об'єктів, системи нечіткої логіки, нейронні мережі, інтелектуалізація, модель, мікросервіси.

ABSTRACT

Palladiiev O.O. Methods and software for solving the classification problem based on three-dimensional neural networks. – Qualification Research Paper (Manuscript).

Dissertation for the degree of Doctor of Philosophy in the specialty 121 – Software Engineering, in the field of knowledge 12 – Information Technologies. – National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute," Kyiv, 2025.

This dissertation is dedicated to the development and investigation of specialized software tools for effective classification of three-dimensional images with the possibility of integration into various automated processes. The main objective of the study is to improve methods for processing and analyzing three-dimensional data by applying advanced approaches to neural network architecture design and developing adaptive optimization algorithms.

The following results were achieved during the research:

For the first time, the use of an opponent color system as a preprocessing stage for image data was proposed. The suggested technique enhances the informativeness of features and improves their interpretation by neural networks, significantly increasing classification efficiency. Additionally, the behavior of the opponent system under different levels of input data noise was studied, enabling the development of strategies to adapt it to specific image datasets.

A new concept for organizing layers in neural networks with localized connection schemes between neurons was developed. The proposed architecture combines expansion and contraction layers, facilitating effective feature extraction and data dimensionality reduction with minimal information loss. Special attention was paid to optimizing the parameters of these layers to ensure flexibility and adaptability to different data types, as well as determining the optimal number of neurons and activation functions to improve network training.

Adaptive methods for optimizing neural network architecture were implemented for the first time, based on dynamic restructuring of local connections during the training process. This approach improves classification efficiency and accuracy by adapting the network structure to the characteristics of input data. Several adaptation algorithms were developed and tested, including methods for gradual weight pruning and adding new connections to enhance model robustness against data variability.

Algorithms for selecting neurons with low activity (weak connections) were proposed to reduce the computational complexity of the network and minimize the risk of overfitting. This allows for maintaining high accuracy results with optimal resource utilization. Criteria for neuron selection were developed, based on feature importance analysis and their contribution to the final decision.

A tool was developed to provide access to the functionality of three-dimensional image classification, enabling the integration of classification capabilities into various automated processes. The implemented solution supports scalability for processing large datasets, including preprocessing functions, result analysis, and performance monitoring.

Comprehensive performance testing of the developed software was conducted using a spectrum of three-dimensional datasets. Experimental results confirmed the high accuracy and performance of the proposed approach, ensuring its competitiveness in solving automated classification tasks for complex three-dimensional structures.

Thus, the created classification method and software tools represent an effective solution for classifying three-dimensional images, combining advanced neural network architectures and optimization algorithms to achieve high accuracy and performance. The proposed approach demonstrates high flexibility and adaptability, allowing its application in a wide range of practical tasks, including medical diagnostics, industrial automation, and scientific data analysis.

The dissertation consists of an introduction, four chapters, conclusions, a list of sources used and appendices. The total volume of the work is 212 pages, of which 168 pages are the main text. The work contains 50 figures, 3 tables and 80 literary sources.

Keywords: software engineering, data analysis, automation, neural network modeling automation, automated processing, deep learning, classification, machine learning, meta-learning, machine learning methods, fuzzy logic, performance evaluation, preprocessing, object recognition, fuzzy logic systems, neural networks, intellectualization, model, microservices.

Список публікацій здобувача

Наукові праці, в яких опубліковано основні наукові результати дисертації:

- Paladiiev O.O., Lisovychenko O.I. The influence of the opponent's color model on the general capabilities of neural networks // Interdepartmental scientific-technical journal «Adaptive systems of automatic control».-2022.-№ 2 (41).-P. 22-27
DOI: <https://doi.org/10.20535/1560-8956.41.2022.271335>
- Паладієв О.О., Лісовиченко О.І. Вплив зменшення розмірів нейронної мережі на її здатність до узагальнення // Міжвідомчий науково-технічний журнал «Адаптивні системи автоматичного управління».-2023.-№ 2 (43).-С. 124-130
DOI: <https://doi.org/10.20535/1560-8956.43.2023.292262>
- Паладієв О.О., Лісовиченко О.І. Тривимірні нейронні мережі у завданнях кластеризації // Міжвідомчий науково-технічний журнал «Адаптивні системи автоматичного управління».-2024.-№ 1 (44).-С. 166-171
DOI: <https://doi.org/10.20535/1560-8956.44.2024.302431>

Наукові праці, які засвідчують апробацію матеріалів дисертації:

- Паладієв О.О., Лісовиченко О.І. Тривимірні нейронні мережі у завданнях кластеризації // Міжнародна науково-практична конференція SoftTech, 21-23 травня 2024 р., м. Київ, Україна. —Київ: КПІ ім. Ігоря Сікорського, 2024. — С. 109-113.

ЗМІСТ

| | |
|--|----|
| ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ..... | 12 |
| ВСТУП | 13 |
| 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ..... | 19 |
| 1.1 Аналіз програмних рішень | 19 |
| 1.1.1 Аналіз програмних засобів для класифікації тривимірних даних . | 20 |
| 1.1.2 Аналіз бібліотек для класифікації тривимірних даних..... | 22 |
| 1.2 Переваги та недоліки наявних рішень..... | 26 |
| 1.3 Аналіз запропонованих рішень | 29 |
| 1.4 Постановка задачі | 32 |
| Висновки до розділу 1 | 33 |
| 2 РОЗРОБКА МЕТОДІВ ДЛЯ КЛАСИФІКАЦІЇ ТРИВИМІРНИХ ЗОБРАЖЕНЬ | 36 |
| 2.1 Форми представлення тривимірних об'єктів..... | 37 |
| 2.1.1 Точкові хмари (Point Clouds) | 37 |
| 2.1.2 Воксели (Voxels)..... | 39 |
| 2.1.3 Полігональні сітки (Meshes) | 40 |
| 2.2 Наявні методи попередньої обробки даних | 41 |
| 2.3 Методи колориметричної корекції | 52 |
| 2.4 Теоретичні аспекти архітектури класифікатора..... | 53 |
| 2.4.1 Теоретичне обґрунтування тривимірних топологій..... | 54 |
| 2.4.2 Розгортка та згортка в тривимірних шарах | 60 |
| 2.5 Методи оптимізації обчислень..... | 63 |
| 2.5.1 Існуючі методи оптимізації..... | 63 |
| 2.5.2 Метод видалення зв'язків..... | 64 |

| | |
|--|------------|
| | 10 |
| 2.5.3 Метод створення зв'язків | 68 |
| 2.6 Теоретична модель класифікатора..... | 72 |
| Висновки до розділу 2 | 76 |
| 3 АРХІТЕКТУРА ТА КОМПОНЕНТИ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ АВТОМАТИЗАЦІЇ КЛАСИФІКАЦІЇ ТРИВИМІРНИХ ЗОБРАЖЕНЬ | 79 |
| 3.1 Загальна структура програмних засобів..... | 79 |
| 3.2 Використані бібліотеки та інструменти | 93 |
| 3.2.1 ML.NET | 94 |
| 3.2.2 Keras.NET..... | 95 |
| 3.3 Реалізація архітектури прихованих шарів | 100 |
| 3.3.1 Метод розрахунку індексів зв'язків в узгодженій за розміром матриці | 100 |
| 3.3.2 Розширення до обчислення індексів зв'язків в узгодженому тривимірному тензорі | 104 |
| 3.3.3 Узагальнення для знаходження індексів зв'язків в неузгодженому тривимірному тензорі | 107 |
| 3.3.4 Програмна реалізація шару для розріджених зв'язків | 110 |
| 3.4 Реалізація методів оптимізації | 113 |
| 3.4.1 Оптимізація структури мережі методом видалення зв'язків | 113 |
| 3.4.2 Динамічне розширення структури нейронної мережі..... | 115 |
| 3.5 Інтерфейс користувача..... | 118 |
| Висновки до розділу 3 | 122 |
| 4 РЕЗУЛЬТАТИ НАВЧАННЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ | 125 |
| 4.1 Дані для навчання та тестування | 125 |

| | | |
|-------|---|-----|
| 4.2 | Процес тестування програмних засобів | 127 |
| 4.3 | Результати експериментів..... | 132 |
| 4.3.1 | Класифікація воксельних даних | 133 |
| 4.3.2 | Класифікація даних в вигляді сіток..... | 137 |
| 4.4 | Порівняльний аналіз..... | 143 |
| | Висновки до розділу 4 | 156 |
| | ВИСНОВКИ..... | 158 |
| | СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... | 161 |
| | ДОДАТОК А Список публікацій здобувача | 169 |
| | ДОДАТОК Б Частина програмного коду | 170 |
| | ДОДАТОК В Моделі класифікаторів..... | 180 |
| | ДОДАТОК Г Графіки | 186 |
| | ДОДАТОК Д Діаграми | 205 |
| | ДОДАТОК Е Наявні програмні засоби..... | 211 |

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

3DCNN (CNN) – Тривимірна згорткова нейронна мережа.

3DSNN (SNN) – Тривимірна розріджена нейронна мережа.

3DSNNO (SNNO) – Тривимірна розріджена нейронна мережа з оптимізацією.

ReLU (Rectified Linear Unit) – Функція активації з випрямленням.

LiDAR (Light Detection and Ranging) – Лазерне зондування для побудови 3D-даних.

PCA (Principal Component Analysis) – Метод головних компонент для зниження розмірності даних.

LSTM (Long Short-Term Memory) – Довга короткочасна пам'ять, різновид рекурентної нейронної мережі.

GRU (Gated Recurrent Unit) – Рекурентний блок з керованими (затворними) елементами.

Adam (Adaptive Moment Estimation) – Адаптивний метод оптимізації градієнтного спуску.

ВСТУП

Актуальність теми

Сучасні інформаційні технології та методи машинного навчання значно розширили можливості автоматизованої обробки та аналізу даних, зокрема в області класифікації тривимірних зображень. Одним із найперспективніших підходів є використання тривимірних нейронних мереж, які здатні враховувати просторову структуру об'єктів, що є важливим для підвищення точності класифікації [1].

Незважаючи на значний прогрес у цій галузі, існуючі підходи мають певні обмеження. Класичні методи часто не забезпечують достатньої гнучкості та ефективності при роботі з великими обсягами даних або складними структурами [2]. З іншого боку, евристичні методи, такі як штучні нейронні мережі, потребують значних обчислювальних ресурсів і складної параметризації, що ускладнює їх широке практичне застосування [3].

Аналіз існуючих рішень виявляє необхідність розробки нових підходів, які поєднують переваги класичних і сучасних методів. Зокрема, актуальною є розробка алгоритмів і програмних засобів для ефективної класифікації зображень, що базуються на тривимірних нейронних мережах [4]. Такі підходи мають потенціал значно покращити результати класифікації завдяки використанню оригінальних методів виділення характеристик і оптимізації архітектури нейронних мереж [5].

З наукової точки зору, дослідження сприятиме розширенню теоретичних основ побудови тривимірних нейронних мереж та розвитку методів глибокого навчання для аналізу складних багатовимірних структур [6]. Розроблені підходи дозволять формалізувати нові моделі обробки даних, що забезпечить подальше поглиблення наукових знань у галузі комп'ютерного зору та штучного інтелекту.

З прикладної точки зору, результати дослідження можуть бути використані в таких сферах, як медична діагностика (аналіз МРТ та КТ-знімків),

автоматизований контроль якості продукції, розпізнавання об'єктів у тривимірному просторі для робототехніки та систем безпеки. Таким чином, запропоновані алгоритми та програмні засоби матимуть широке практичне застосування.

Таким чином, дослідження, спрямоване на підвищення ефективності класифікації зображень за допомогою тривимірних нейронних мереж та створення спеціалізованих програмних засобів, є надзвичайно важливим як з наукової, так і з прикладної точки зору [7].

Зв'язок роботи з науковими програмами, планами, темами

Зв'язок роботи з науковими програмами, планами, темами. Дослідження за темою дисертаційної роботи проводилось у Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» в рамках виконання держбюджетних науково-дослідних робіт «Створення гібридної обчислювальної технології побудови квазі-формалізованої моделі прогнозування в умовах неоднорідності даних та ненормативних відхилень в системах організаційного управління» (номер державної реєстрації 0117U002448).

Мета дослідження

Метою даної дисертації є підвищення продуктивності та ефективності програмних засобів для класифікації тривимірних зображень на основі тривимірних нейронних мереж.

Завдання дослідження

Для досягнення вказаної мети було сформовано наступні завдання:

1. провести аналіз сучасних методів і підходів до класифікації тривимірних даних на основі нейронних мереж;
2. реалізувати методи попередньої обробки тривимірних даних з урахуванням специфіки їхньої просторової структури;

3. спроектувати та реалізувати архітектуру класифікатора для ефективної класифікації тривимірних даних з можливістю використання методів метанавчання для підвищення точності прогнозування;
4. розробити методи зміни кількості зв'язків у тривимірних нейронних мережах для забезпечення високої продуктивності та узагальнюючої здатності;
5. реалізувати програмні засоби з підтримкою WebAPI для забезпечення інтеграції з іншими інформаційними системами та хмарними платформами;
6. розробити метод тестування та верифікації класифікатора з метою оцінки його точності, продуктивності та масштабованості;
7. провести експерименти для оцінки точності та продуктивності створених програмних засобів.

Результатом виконання завдань стане:

1. підвищення точності класифікації за рахунок оригінальної топології класифікатора;
2. підвищення швидкодії за рахунок зменшення кількості параметрів моделі для класифікації;
3. програмні засоби для автоматизації класифікації тривимірних даних.

Об'єкт дослідження – процес класифікації зображень на базі нейронних мереж.

Предмет дослідження – методи та моделі виділення ключових ознак з тривимірних зображень.

Методи дослідження – методи оптимізації, теорія нейронних мереж, аналіз тривимірних даних, алгоритми машинного навчання, методи чисельного моделювання та обробки великих масивів даних. Для програмної реалізації використано об'єктно-орієнтоване та процедурне програмування, а також технології високопродуктивних і паралельних обчислень.

Наукова новизна отриманих результатів:

1. **вперше розроблено** топологію нейронних зв'язків яка реалізує локально-обмежені структури зв'язків з протилежними нейронами та їх безпосередніми сусідами в тривимірному просторі, що підвищує продуктивність програмних засобів шляхом зменшення кількості обчислень за рахунок меншої кількості з'єднань у шарах нейронних мереж;
2. **вперше розроблено метод** визначення структури зв'язків між шарами нейронної мережі, що забезпечує збереження локальних залежностей між сусідніми елементами у тривимірному просторі та дозволяє ефективно обробляти одномірні вектори шарів, підтримуючи точність і гнучкість моделі за рахунок оптимізованої архітектури програмних засобів;
3. **вперше розроблено методи** зміни кількості нейронних зв'язків, що підвищують швидкодію програмних засобів класифікації, за рахунок видалення слабких зв'язків та підвищують точність класифікації за рахунок генерації нових зв'язків для нейронів з вагами, наближеними до меж функції активації;
4. **запропоновано метод** попередньої колорифікаційної обробки даних для нейронних мереж, заснований на опонентній теорії кольорів, який дозволяє перетворювати кольори в опонентні кольори, що збільшує кореляцію між кольорами, підвищуючи ефективність навчання та зменшуючи втрати нейронної мережі.

Практичне значення отриманих результатів

Розроблено спеціалізовані програмні засоби із підтримкою RESTful API для автоматизованої класифікації тривимірних даних, що забезпечує інтеграцію з сучасними інформаційними системами та хмарними платформами. Програмний комплекс реалізує механізми обробки просторових структур на основі попередньо налаштованих нейронних мереж, доступ до яких здійснюється через інтерфейс прикладного програмування (API) з використанням унікальних ідентифікаторів мереж.

Запропоновані алгоритми включають адаптивну реконфігурацію нейронних зв'язків із вагою, наближеною до порогових значень, визначених активаційними функціями. Крім того, передбачено алгоритмічне доповнення структури новими зв'язаннями для нейронів із високим рівнем активності, що підвищує їхню функціональну роль у процесах обробки даних.

Програмні засоби орієнтовані на використання в задачах класифікації тривимірних даних таких як вокселі та сітки, і може бути інтегрований у масштабовані системи аналізу великих даних. Архітектура рішення передбачає гнучке налаштування параметрів моделей та їх адаптацію до специфіки вхідних даних, що дозволяє ефективно обробляти нові вибірки без необхідності перепроєктування мережі.

Розроблені засоби має потенціал для застосування в різних галузях, включаючи медичну діагностику, системи тривимірного сканування, автономні роботизовані платформи та системи моніторингу навколишнього середовища. Інтеграція з хмарними платформами забезпечує масштабованість і мобільність застосування, розширюючи можливості для аналітичної обробки тривимірних даних.

Особистий внесок здобувача

Усі результати, що виносяться автором до захисту, отримані особисто в процесі науково-дослідницької роботи. У наукових працях, опублікованих у співавторстві, автору належать:

- в роботі [8] обґрунтування впливу зменшення розмірів нейронних мереж на їх здатність до узагальнення, а також формулювання критеріїв вибору оптимальних параметрів архітектури для забезпечення балансу між продуктивністю та узагальнюючими властивостями;

- в роботі [9] розробка процесу попередньої обробки вхідних даних на основі опонентної кольорової моделі, що імітує біологічні принципи сприйняття кольору і підвищує ефективність класифікації зображень;
- в роботі [10] створення архітектури тривимірних нейронних мереж для задач кластеризації, включаючи проектування та реалізацію мережевої структури, що оптимізує просторові залежності між об'єктами, і оцінка її ефективності за допомогою індексу Данна.

Отримані результати сприяють подальшому розвитку методів аналізу даних та оптимізації нейронних мереж у задачах класифікації, кластеризації та розпізнавання образів.

Публікації

За результатами дисертаційних досліджень опубліковано 3 статті у фахових наукових журналах категорії Б.

Структура і обсяг роботи

Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 212 сторінок, з яких 168 сторінок основного тексту. Робота містить 50 рисунків, 3 таблиці та 80 літературних джерел.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Класифікація тривимірних даних становить фундаментальне завдання в галузі обробки інформації, що знаходить широке застосування в таких напрямках, як медична діагностика, промисловий контроль, геодезичні вимірювання та моделювання віртуальних середовищ [11]. Зі збільшенням обсягів та складності даних зростає потреба в розробці високопродуктивних програмних рішень, здатних ефективно обробляти, аналізувати та класифікувати тривимірні об'єкти [12].

Актуальні програмні засоби для роботи з тривимірними даними охоплюють комплексні платформи, спеціалізовані бібліотеки та інтегровані програмні пакети, що забезпечують розширені можливості для аналізу та візуалізації 3D-об'єктів [13].

У цьому розділі представлено аналіз предметної області, що включає огляд програмних рішень і бібліотек для класифікації тривимірних даних [37]. Проведено оцінку їхніх переваг і недоліків, а також розглянуто ключові проблеми, що обмежують ефективність існуючих підходів. Окрему увагу приділено перспективам вдосконалення програмних засобів шляхом оптимізації алгоритмів і застосування сучасних підходів до машинного навчання.

Розділ розпочинається з аналізу програмних рішень і бібліотек для класифікації тривимірних даних (підрозділ 1.1), де детально розглядаються основні інструменти та технології, що використовуються в сучасних системах обробки тривимірної інформації.

1.1 Аналіз програмних рішень

Даний розділ присвячений огляду існуючих програмних засобів і бібліотек, що використовуються для класифікації тривимірних даних. Особливу увагу приділено їхнім перевагам і обмеженням, що дозволяє оцінити поточний стан технологій і виявити напрямки для подальших досліджень та вдосконалень.

1.1.1 Аналіз програмних засобів для класифікації тривимірних даних

Програмні засоби для класифікації тривимірних даних є важливим інструментом у сфері просторового аналізу та обробки великих обсягів інформації. Його основне призначення полягає у вирішенні задач структуризації, сегментації та групування точкових хмар, що забезпечує подальшу ідентифікацію об'єктів. Такі програмні засоби, незважаючи на свою функціональність, здебільшого мають вузьку спеціалізацію та обмежені можливості інтеграції в масштабовані інформаційні системи. Часто їх класифікаційні функції реалізовані за допомогою окремих модулів або плагінів, що ускладнює автоматизацію процесів. Розглянемо детальніше основні програмні рішення, що використовуються в цій області.

CloudCompare

CloudCompare (див. рис. Е.1) є одним із провідних інструментів для обробки та аналізу тривимірних точкових хмар. Ця програма підтримує широкий спектр форматів (LAS, PLY, OBJ) і надає можливості для візуалізації та сегментації даних [14]. Її плагіни дозволяють проводити автоматичну класифікацію об'єктів за геометричними характеристиками. Проте, CloudCompare більше орієнтований на обробку окремих наборів даних і не забезпечує інструментів для інтеграції в комплексні системи аналізу. Додатково варто зазначити, що CloudCompare дозволяє виконувати багаторівневий аналіз класифікованих даних, але обмеження щодо автоматизації створюють труднощі для застосування в реальних сценаріях з великими наборами даних [15].

3DReshaper

3DReshaper є гнучким рішенням для обробки тривимірних моделей і точкових хмар. Його інструментарій включає алгоритми класифікації та сегментації, що дозволяє аналізувати дані в інженерних та промислових застосуваннях [16]. Програма підтримує популярні формати STL, OBJ, PLY і забезпечує високу точність обробки. Вона також має розширені функції

згладжування поверхонь та корекції шумів, що робить її зручною для роботи з даними, отриманими з лазерного сканування. Однак її використання обмежується автономними сценаріями, оскільки вона не має розвинених механізмів інтеграції в масштабовані платформи, що ускладнює автоматизацію процесів аналізу [17].

TerraScan

TerraScan спеціалізується на аналізі даних лазерного сканування (LiDAR). Вона забезпечує автоматизовану класифікацію точок на основі висоти, інтенсивності та інших параметрів [18]. TerraScan підтримує стандартні формати LAS і LAZ, що робить її сумісною з більшістю систем збору LiDAR-даних. Програма включає інструменти для коригування помилок в даних та створення тривимірних моделей місцевості, що є важливим для геодезичних і картографічних задач. Проте її налаштування та використання потребують значного рівня експертизи, а її функціонал спрямований на автономне використання, що обмежує інтеграцію в більші інформаційні системи [19]. Крім того, її алгоритми оптимізовані для певних типів даних і вимагають попереднього калібрування параметрів.

PointCab

PointCab (див. рис. Е.2) надає потужні засоби для обробки 3D-сканів і класифікації точок. Основна сфера застосування включає архітектурне проектування та будівництво [20]. Програма підтримує формати E57, PTS, PTX і дозволяє виконувати попередню обробку даних перед імпортом в інші інструменти. Незважаючи на розширені можливості візуалізації, її класифікаційний функціонал орієнтований на початкову обробку, а не на масштабовані задачі аналізу великих обсягів даних [21]. Додатково PointCab дозволяє експортувати результати в універсальні формати для подальшої обробки іншими системами, проте її автоматизація обмежується локальними сценаріями використання [64, 65].

LiDAR360

LiDAR360 (див. рис. Е.3) є комплексним рішенням для аналізу даних, отриманих за допомогою LiDAR. Програма підтримує автоматизовану класифікацію точок, включаючи виявлення рослинності, будівель і поверхонь [22]. Підтримуються формати LAS, LAZ та TXT. LiDAR360 також забезпечує модулі для виявлення змін в часі, що є корисним для екологічного моніторингу та управління земельними ресурсами [23]. Водночас, LiDAR360 вимагає складного налаштування алгоритмів і не передбачає легкого масштабування для інтеграції в автоматизовані системи обробки, що обмежує її застосування в проектах великого масштабу [70, 71].

Підсумовуючи, наведені програмні засоби демонструють високу ефективність у класифікації тривимірних даних завдяки використанню сучасних алгоритмів машинного навчання та статистичних методів. Проте їхня вузька спрямованість, залежність від плагінів та модульних рішень, а також обмежена інтеграція в складні інформаційні системи створюють значні перешкоди для масштабування та автоматизації. Це вимагає розробки нових підходів і засобів для підвищення ефективності класифікації в умовах автоматизації обчислень.

1.1.2 Аналіз бібліотек для класифікації тривимірних даних

Розвиток обчислювальних технологій та алгоритмів обробки даних створив підґрунтя для появи бібліотек, орієнтованих на класифікацію тривимірних даних [47]. Ці бібліотеки надають інструменти для попередньої обробки, аналізу, сегментації та візуалізації даних, проте їх ефективне використання вимагає комплексного підходу до програмування та інтеграції зовнішніх модулів. Важливим аспектом є гнучкість налаштувань цих бібліотек, яка дозволяє створювати спеціалізовані рішення для конкретних задач, але водночас ускладнює інтеграцію в автоматизовані системи з високою продуктивністю. Нижче розглянуто основні бібліотеки для автоматизації класифікації тривимірних даних, що активно застосовуються у наукових і прикладних дослідженнях.

Open3D

Open3D є однією з провідних бібліотек для роботи з тривимірними даними. Вона підтримує широкий спектр інструментів для реєстрації, фільтрації, класифікації та візуалізації точкових хмар, тривимірних сіток і вокселів [24]. Її інтеграція з алгоритмами машинного навчання дозволяє автоматизувати процеси аналізу даних. Крім базових функцій, Open3D підтримує багатопотокову обробку та роботу з великими наборами даних, що підвищує продуктивність для складних завдань [25]. Проте використання Open3D потребує реалізації зовнішніх модулів, що робить її зручною лише для розробників із високим рівнем програмування. Додатковою перевагою є підтримка роботи з даними у форматах PLY, OBJ та STL, що розширює спектр її застосувань.

PCL

PCL є масштабованою бібліотекою для обробки та аналізу точкових хмар. Вона включає модулі для сегментації, фільтрації та класифікації, а також підтримує реєстрацію об'єктів і роботу з поверхнями [15]. Її глибока інтеграція з OpenCV розширює можливості обробки даних. PCL також підтримує алгоритми вирівнювання та згладжування поверхонь, що дозволяє адаптувати її до завдань із високими вимогами до точності [26]. Однак для автоматизації складних сценаріїв PCL потребує реалізації спеціалізованих алгоритмів і налаштування зовнішніх плагінів, що вимагає досвіду в алгоритмічному проектуванні. Незважаючи на широкий функціонал, обмеження щодо сумісності з великими наборами даних може створювати перешкоди для інтеграції в промислові системи.

Keras

Keras — це високорівнева бібліотека для створення та навчання нейронних мереж, побудована поверх TensorFlow. Вона забезпечує зручний і гнучкий інтерфейс для проектування моделей, підтримує як послідовні, так і функціональні API для створення складних архітектур [1]. Keras широко

використовується для класифікації, регресії, кластеризації, а також для обробки зображень, текстів і 3D-даних.

Основною перевагою Keras є модульність і простота використання, що дозволяє швидко експериментувати з моделями. Бібліотека також підтримує інтеграцію з TensorFlow, що забезпечує масштабованість і високу продуктивність для роботи з великими наборами даних [27]. Завдяки вбудованим інструментам для візуалізації, оптимізації та оцінки результатів Keras підходить як для дослідницьких, так і для прикладних завдань машинного навчання.

Проте для специфічних завдань, таких як обробка LiDAR-даних або 3D-вокселів, може знадобитися створення спеціалізованих шарів і функцій, що вимагає поглиблених знань TensorFlow і Python.

TensorFlow & PyTorch

TensorFlow і PyTorch, як провідні фреймворки для машинного навчання, пропонують спеціалізовані модулі для роботи з тривимірними даними. TensorFlow 3D і PyTorch3D забезпечують інструменти для побудови нейронних мереж, призначених для класифікації, сегментації та виявлення об'єктів у 3D-просторі [28]. Вони підтримують складні архітектури глибоких нейронних мереж і включають готові моделі для вирішення стандартних задач аналізу [29]. Ці бібліотеки відзначаються високою гнучкістю та масштабованістю, проте їх ефективне використання потребує налаштування всіх етапів аналізу через програмний код, включаючи передобробку, навчання та валідацію моделей. Додатково їх інтеграція в промислові середовища може вимагати адаптації інфраструктури для роботи з великими обсягами даних.

VTK

VTK є комплексним інструментом для візуалізації та обробки тривимірних даних. Хоча її основна функція полягає у візуалізації, вона включає алгоритми для аналізу точкових хмар і сіток [30]. Завдяки інтеграції з бібліотекою ІТК вона може бути частиною складних систем обробки. Проте для реалізації задач

класифікації вимагається програмування зовнішніх модулів, що обмежує її застосування для автоматизованих рішень. Її широке використання у візуалізації геометричних структур робить її придатною для попередньої обробки даних перед їх подальшою класифікацією [31].

PyMesh

PyMesh орієнтована на обробку тривимірних сіток і надає функціонал для генерації, редагування та аналізу геометричних структур [32]. Вона ефективно працює з багатогранними моделями та підтримує інтеграцію з іншими бібліотеками для реалізації алгоритмів класифікації. PyMesh також включає інструменти для оптимізації сіток, що є важливим для задач зі складною геометрією [33]. Проте її використання вимагає розробки спеціалізованих модулів машинного навчання для досягнення автоматизації процесів класифікації.

Laspy

Laspy є вузькоспеціалізованою бібліотекою для роботи з даними у форматі LAS, які є стандартом для LiDAR-сканування. Вона забезпечує функціонал для управління, фільтрації та аналізу точкових хмар [34]. Однак її можливості обмежені маніпуляціями з файлами, і для інтеграції алгоритмів класифікації необхідно створювати додаткові зовнішні модулі. Її основне призначення — обробка великих обсягів даних у форматі LiDAR для подальшої передачі в інші інструменти аналізу [35].

Розглянуті бібліотеки є потужними інструментами для обробки та класифікації тривимірних даних. Вони демонструють високу продуктивність та гнучкість завдяки підтримці алгоритмів машинного навчання, проте потребують значних зусиль для програмної інтеграції. Усі описані бібліотеки орієнтовані на спеціалізовані застосування та не є готовими рішеннями для автоматизованої класифікації. Їх використання вимагає глибокого розуміння алгоритмів і

здатності до розробки власних модулів, що підкреслює необхідність подальшого вдосконалення існуючих підходів для масштабованої автоматизації.

1.2 Переваги та недоліки наявних рішень

Розглянуті програмні рішення, такі як CloudCompare, 3DReshaper, TerraScan, PointCab та LiDAR360, пропонують широкий спектр інструментів для аналізу й класифікації тривимірних даних. Їх застосування є ефективним у спеціалізованих прикладних задачах, зокрема в геодезії, картографії, архітектурі та екологічному моніторингу. Однак аналіз виявив як переваги, так і суттєві недоліки, що обмежують можливості цих рішень у масштабованих і автоматизованих системах. Нижче наведено детальний огляд переваг і недоліків.

Переваги

Програми, такі як CloudCompare і TerraScan, оснащені модулями для сегментації, згладжування, класифікації та візуалізації тривимірних даних. Вони дозволяють виконувати широкий спектр операцій із мінімальною необхідністю зовнішніх інструментів, забезпечуючи зручність роботи для користувачів.

Інструменти підтримують роботу з популярними форматами, такими як LAS, PLY, OBJ, STL і E57. Це забезпечує високу сумісність із іншими системами обробки даних та інтеграцію в існуючі робочі процеси.

Такі програми, як PointCab і 3DReshaper, мають зручні графічні інтерфейси, що робить їх доступними для користувачів без глибоких знань програмування. Це спрощує роботу з інструментами і скорочує час навчання нових користувачів.

LiDAR360 та інші програми забезпечують потужні інструменти для тривимірної візуалізації, що сприяє швидкому аналізу даних великих обсягів і дозволяє користувачам оцінювати складні структури в реальному часі.

Програми демонструють високу ефективність у виконанні локальних обчислювальних задач, що робить їх корисними для розв'язання специфічних інженерних і наукових проблем.

Недоліки

Більшість програм оптимізовані для роботи з певними типами даних (наприклад, точковими хмарами чи сітками). Це створює труднощі при необхідності комбінованого аналізу різнорідних даних або інтеграції в складніші процеси.

Інструменти орієнтовані на ручну обробку даних і не підтримують повноцінну автоматизацію процесів класифікації. Це обмежує їхню ефективність у масштабованих проектах з великими обсягами інформації.

Програми працюють як автономні рішення і не передбачають легкої інтеграції в хмарні або масштабовані системи. Це ускладнює використання їх у високопродуктивних середовищах.

Багато програм не оптимізовані для обробки великих обсягів даних за допомогою багато поточності чи розподілених обчислень, що призводить до зниження продуктивності при роботі з великими масивами.

Програми, такі як LiDAR360, вимагають значних обчислювальних ресурсів, що ускладнює їх використання на стандартних комп'ютерах і обмежує доступність для малих підприємств та дослідницьких груп.

На відміну від бібліотек, такі програми, як PointCab і TerraScan, мають закриту архітектуру. Це ускладнює їхню адаптацію під специфічні завдання користувачів і обмежує можливості інтеграції з новими алгоритмами або системами обробки даних.

Існуючі рішення не передбачають легкого додавання нових модулів або інтеграції алгоритмів машинного навчання, що обмежує їхню ефективність у задачах із динамічно змінними вимогами.

Таблиця 1.1 - Порівняння програмних засобів.

| Метрика | CloudCompare | 3DReshaper | TerraScan | PointCab | LiDAR360 |
|--|---|---|--|---|---|
| Основне призначення | Аналіз і візуалізація тривимірних даних; сегментація, класифікація та ін. | Обробка 3D-моделей, створення сіток, аналіз даних для промислового та архітектурного застосування | Спеціалізована класифікація та обробка хмар точок, переважно для геодезії та ГІС | Швидка візуалізація та аналіз даних з лазерного сканування й фотограмметрії | Аналіз і візуалізація хмар точок великих обсягів, зокрема для геопросторових задач |
| Наявність графічного інтерфейсу | Зручний та достатньо гнучкий інтерфейс, орієнтований на візуальний аналіз | Має інтуїтивний графічний інтерфейс, проте потребує певного навчання | Інтерфейс досить технічний, оптимізований під геодезичні завдання | Простий та зручний інтерфейс, орієнтований на швидке освоєння | Має потужні засоби візуалізації, інтерфейс орієнтований на професіоналів |
| Можливості автоматизації | Орієнтований на ручну обробку, має базові скриптові інструменти | Переважно ручний режим, обмежена можливість макросів | Більшість процесів потребує ручної настройки, відсутні повноцінні автоматизовані конвеєри | Здебільшого ручні операції, відсутні механізми розгорнутої автоматизації | Також фокусується на інтерактивній обробці, автоматизація обмежена |
| Інтеграція в масштабовані системи | Може працювати як окремий інструмент, складно інтегрувати у хмарні середовища | Потребує окремого налаштування, орієнтований на офлайн-обробку | Орієнтований на настільні середовища, інтеграція в масштабовані платформи обмежена | Працює як автономне ПЗ, немає розвинених API чи SDK для сторонньої інтеграції | Немає повної хмарної інтеграції, проте працює з великими наборами даних на окремих робочих станціях |
| Оптимізація для великих обсягів даних | Обмежена підтримка багато поточності, але достатньо швидко для невеликих проектів | Орієнтований на середні об'єми, можливе зниження продуктивності на великих масивах даних | Часткова підтримка паралельних обчислень, але не розрахована на дуже великі хмарні рішення | Ефективний для малих і середніх проектів, продуктивність падає на великих наборах | Спроекований для великих обсягів даних, проте потребує потужних ресурсів |

Кінець табл. 1.1.

| Метрика | CloudCompare | 3DReshaper | TerraScan | PointCab | LiDAR360 |
|---|--|---|--|--|---|
| Вимоги до апаратного забезпечення | Порівняно невисокі, можна запускати на середніх ПК | Середні вимоги, залежить від складності сіток і обсягів даних | Вимагає достатньо потужного ПК для великих проектів | Середнього рівня вимоги, можливі обмеження при аналізі великих точкових хмар | Потребує високопродуктивного обладнання, особливо для великих наборів даних |
| Доступність коду та розширюваність | Відкритий код (CloudCompare) | Закритий код, з обмеженою підтримкою сторонніх розширень | Пропрієтарне рішення, відсутній відкритий доступ до коду | Закритий код, кастомізація через обмежений набір інструментів | Комерційне рішення, немає повноцінної підтримки для інтеграції власних алгоритмів |

Як бачимо з таблиці 1.1, програмні рішення для класифікації тривимірних даних демонструють ефективність у вирішенні локальних задач завдяки інтегрованим інструментам, підтримці популярних форматів і зручним графічним інтерфейсам. Проте їх обмеження, такі як відсутність автоматизації, складність інтеграції та високі вимоги до ресурсів, ускладнюють їх використання в масштабованих і автоматизованих системах. Це вказує на необхідність розробки нових підходів, орієнтованих на підвищення продуктивності, універсальності та адаптивності програмних рішень для класифікації тривимірних даних.

1.3 Аналіз запропонованих рішень

Класифікація тривимірних даних є фундаментальним завданням в областях комп'ютерного зору, геоінформаційних систем та промислових додатків. Проте, незважаючи на наявність різноманітних програмних засобів і бібліотек, сучасні підходи не забезпечують достатньої точності, гнучкості та автоматизації, що обмежує їхнє застосування в складних сценаріях. Аналіз попередніх розділів

дозволив ідентифікувати ключові проблеми, які перешкоджають досягненню оптимальних результатів класифікації та масштабованості систем обробки даних.

Обмежена точність класифікації

Сучасні програмні рішення демонструють лише помірний рівень точності, що є прийнятним для загальних задач, але недостатнім для складних структур. Наявні алгоритми не завжди враховують локальні варіації в даних, що призводить до втрати інформації про деталі та погіршення якості результатів класифікації. Особливо це помітно при обробці шумових наборів даних або комплексних об'єктів з високим рівнем неоднорідності. Крім того, методи навчання моделей часто базуються на спрощених підходах, які не враховують складні просторові взаємозв'язки між сітками чи вокселями, що обмежує можливості для підвищення точності.

Відсутність автоматизації та інтеграції з зовнішніми системами

Розглянуті програмні рішення, такі як CloudCompare, 3DReshaper, TerraScan, PointCab і LiDAR360, орієнтовані на автономне використання. Відсутність інструментів для автоматизації процесів класифікації ускладнює їх інтеграцію в комплексні обчислювальні системи та хмарні платформи. Це створює бар'єри для використання в системах реального часу, де потрібна постійна обробка даних із сенсорів або аналіз великих потоків інформації. Низький рівень інтеграції також обмежує можливості комбінування цих програм з алгоритмами машинного навчання або інструментами штучного інтелекту.

Залежність від класичних моделей

Більшість програмних рішень базуються на традиційних методах аналізу даних, які, хоча й забезпечують базову функціональність, не відповідають сучасним вимогам до продуктивності та гнучкості. Класичні моделі не враховують можливості глибокого навчання для підвищення точності та адаптації. Відсутність підтримки архітектур глибоких нейронних мереж, таких

як 3D-CNN або трансформери, обмежує здатність алгоритмів до аналізу складних залежностей між даними та автоматичного покращення моделей у процесі навчання.

Специфічність архітектур для різних типів даних

Існуючі рішення орієнтовані на обробку окремих форматів даних, таких як точкові хмари, сітки або вокселі. Це вимагає застосування специфічних алгоритмів і архітектур для кожного типу, що ускладнює інтеграцію різнорідних наборів даних. Подібна фрагментованість підходів призводить до необхідності створення окремих моделей для кожного формату, збільшуючи витрати на розробку, тестування та оптимізацію. У випадках комбінованих форматів або даних з різною топологією, наявні інструменти стають ще менш ефективними [77].

Високі обчислювальні витрати та вимоги до ресурсів

Програми, такі як LiDAR360, демонструють значне навантаження на апаратні ресурси, особливо при роботі з великими наборами даних. Відсутність підтримки паралельних обчислень і масштабованих хмарних рішень обмежує продуктивність при роботі в реальному часі. Це ускладнює використання таких рішень у великих проєктах і знижує їхню ефективність у розподілених системах обробки. Крім того, потреба у високопродуктивних графічних процесорах і серверних платформах підвищує витрати на впровадження таких рішень.

Закритість архітектури та обмежені можливості адаптації

Більшість програм мають закриту архітектуру, що унеможливорює додавання нових алгоритмів і моделей машинного навчання без значних модифікацій. Це ускладнює адаптацію інструментів під специфічні сценарії використання, зокрема у випадках потреби в налаштуванні параметрів для специфічних форматів даних. Відсутність підтримки для налаштування архітектур обмежує можливості для експериментів і впровадження нових методик, що є критичним для дослідницьких застосувань.

Аналіз проблем класифікації тривимірних даних показав, що наявні рішення мають низку значних обмежень. Зокрема, це недостатня точність алгоритмів, обмежена інтеграція з іншими системами, залежність від класичних моделей, висока ресурсомісткість і вузька спеціалізація архітектур. Подолання цих бар'єрів вимагає створення універсальних платформ, що підтримують автоматизацію, інтеграцію сучасних методів глибокого навчання та оптимізовану обробку різнорідних даних. Майбутні дослідження повинні зосереджуватися на розробці архітектур із гнучкими параметрами налаштування та алгоритмами самооптимізації для підвищення продуктивності та точності класифікації тривимірних даних.

1.4 Постановка задачі

Метою даної дисертації є підвищення продуктивності та ефективності програмних засобів для класифікації тривимірних зображень на основі тривимірних нейронних мереж. Розроблені засоби повинні інтегрувати сучасні алгоритми обробки та оптимізації навчання нейронних мереж, забезпечуючи високу точність та продуктивність при аналізі тривимірних даних. Крім того, програмні засоби мають підтримувати автоматизовану обробку різнорідних вхідних даних та бути адаптивним до нових архітектур нейронних мереж.

Для реалізації зазначеної мети необхідно вирішити такі завдання:

1. провести аналіз сучасних методів і підходів до класифікації тривимірних даних на основі нейронних мереж;
2. реалізувати методи попередньої обробки тривимірних даних з урахуванням специфіки їхньої просторової структури;
3. спроектувати та реалізувати архітектуру класифікатора для ефективної класифікації тривимірних даних з можливістю використання методів мета-навчання для підвищення точності прогнозування;

4. розробити методи зміни кількості зв'язків у тривимірних нейронних мережах для забезпечення високої продуктивності та узагальнюючої здатності;
5. реалізувати програмні засоби із підтримкою WebAPI для забезпечення інтеграції з іншими інформаційними системами та хмарними платформами;
6. розробити метод тестування та верифікації класифікатора з метою оцінки його точності, продуктивності та масштабованості;
7. провести експерименти для оцінки точності та продуктивності створених програмних засобів.

Запропоновані завдання спрямовані на створення універсального інструменту для класифікації тривимірних даних, який відповідатиме сучасним вимогам до ефективності, масштабованості та інтеграції з іншими системами. Реалізація цих завдань забезпечить основу для подальшого розвитку програмних засобів та вдосконалення алгоритмів аналізу тривимірних структур.

Висновки до розділу 1

Проведений аналіз сучасних методів і програмних засобів для класифікації тривимірних даних дозволив ідентифікувати критичні аспекти, що визначають поточний рівень розвитку цієї галузі. Розгляд існуючих підходів продемонстрував їхню ефективність у вузькоспеціалізованих завданнях, проте виявив значні обмеження щодо масштабованості, автоматизації та інтеграції в розподілені обчислювальні середовища. Дослідження висвітлило потребу в комплексному підході до оптимізації алгоритмів і архітектур, що мають забезпечити більшу ефективність при роботі з великими масивами даних і в реальних часових умовах.

Дослідження доступних програмних рішень, таких як CloudCompare, 3DReshaper, TerraScan, PointCab і LiDAR360, підкреслило їхню придатність для вирішення конкретних задач аналізу тривимірних даних. Водночас їхня

орієнтація на локальні сценарії та залежність від ручного налаштування обмежують можливості для інтеграції в сучасні автоматизовані системи обробки даних. Аналіз бібліотек Open3D, PCL, TensorFlow 3D і PyTorch3D підтвердив їхню придатність до реалізації методів глибокого навчання, проте вказав на необхідність значних програмних і обчислювальних ресурсів для їх ефективного застосування. Зокрема, ці бібліотеки потребують детального налаштування параметрів і створення модулів для адаптації під специфічні сценарії використання, що утворює певну складність їх інтеграції в масштабовані системи.

У ході дослідження виявлено, що існуючі підходи недостатньо враховують специфіку обробки різнорідних форматів тривимірних даних і демонструють обмежену адаптивність до нових архітектур машинного навчання. Це створює потребу у формуванні нових методів, здатних забезпечити оптимізацію обчислювальних витрат і підвищити точність класифікації. Також було виявлено, що перспективні підходи до обробки даних вимагають розробки алгоритмів, орієнтованих на спрощення процесу інтеграції різних типів даних та автоматизацію аналізу в реальному часі. Впровадження таких методів дозволить адаптувати архітектури глибоких нейронних мереж до специфічних умов і типів вхідних даних, розширюючи їхні функціональні можливості.

Аналіз результатів дозволив сформулювати наукове підґрунтя для подальшого розвитку спеціалізованих програмних засобів, що будуть орієнтовані на інтеграцію архітектур глибоких нейронних мереж, методів метанавчання та механізмів оптимізації параметрів. Особлива увага буде приділена створенню алгоритмів, здатних до динамічного оновлення параметрів моделі на основі потокових даних і підтримки високої продуктивності в умовах розподілених обчислень. Такі рішення відкривають перспективи для реалізації автоматизованих платформ, які забезпечують ефективну класифікацію даних у масштабованих і гібридних інформаційних системах.

Результати цього розділу стали відправною точкою для визначення напрямків подальших досліджень, спрямованих на створення масштабованих, продуктивних та адаптивних рішень для автоматизованої класифікації тривимірних даних. Подальша робота буде сфокусована на розробці алгоритмів, що підтримують глибокі нейронні мережі з урахуванням оптимізації параметрів і адаптивних підходів до аналізу. Планується також створення інтегрованих середовищ для тестування й оптимізації моделей у режимі реального часу, що підвищить точність і швидкодію класифікації при збереженні високої гнучкості системи.

2 РОЗРОБКА МЕТОДІВ ДЛЯ КЛАСИФІКАЦІЇ ТРИВИМІРНИХ ЗОБРАЖЕНЬ

У цьому розділі представлено комплексний підхід до розробки методу класифікації тривимірних зображень, що базується на використанні нейронних мереж із вдосконаленою архітектурою та оптимізаційними методами. Основна увага приділяється побудові ефективної системи обробки даних, яка враховує особливості роботи з тривимірними форматами, включаючи точкові хмари, вокселі та полігональні сітки. Здійснюється аналіз методів попередньої обробки, принципів організації прихованих шарів нейронних мереж, підходів до оптимізації обчислень і формування теоретичної моделі класифікатора.

Розробка методу класифікації починається з аналізу підготовки даних, що включає нормалізацію, видалення шумів і корекцію кольорових характеристик для підвищення інформативності вхідних даних. Далі розглядається архітектура прихованих шарів, зокрема застосування механізмів інфляції та дефляції для масштабування обчислень і побудови адаптивних просторових зв'язків між нейронами.

Особливу увагу приділено методам оптимізації обчислювальних процесів, включаючи обмеження на зв'язки в мережі, динамічне зв'язування, само редукацію нейронів і алгоритми міграції зв'язків. Ці підходи спрямовані на підвищення продуктивності та адаптивності моделі, що забезпечує її ефективність у масштабованих системах.

Завершальною частиною розділу є опис теоретичної моделі класифікатора, яка інтегрує розроблені методи та алгоритми в єдину структуру. Модель підтримує адаптивну оптимізацію параметрів і ефективно працює з великими обсягами інформації, гарантуючи точність і продуктивність у задачах класифікації тривимірних даних.

У результаті розгляду запропонованих методів і моделей обґрунтовано їхню доцільність та ефективність для обробки тривимірних зображень, а також

визначено перспективи подальшого вдосконалення архітектури та її програмної реалізації.

2.1 Форми представлення тривимірних об'єктів

Тривимірні дані, що застосовуються для класифікації, представлені у трьох найпоширеніших форматах (див. рис. 2.1):

- точкові хмари (point clouds);
- вокселі (voxels);
- полігональні сітки (meshes).



Рисунок 2.1 - Зображення тривимірного об'єкта в різних представленнях.

Кожен із цих форматів має свої переваги та обмеження, які впливають на вибір методів їх обробки. Наприклад, точкові хмари забезпечують високу деталізацію, але не мають явних топологічних зв'язків між точками, що створює труднощі при реконструкції поверхонь. Вокселі дозволяють моделювати внутрішню структуру об'єктів, однак вимагають значних обчислювальних ресурсів. Полігональні сітки забезпечують точне геометричне представлення поверхонь, проте потребують оптимізації для роботи з великими обсягами даних.

У цьому розділі розглянуто основні типи даних що використовуються в задачах класифікації тривимірних об'єктів.

2.1.1 Точкові хмари (Point Clouds)

Точкові хмари є множиною дискретних точок (див. рис. 2.2), кожна з яких описується координатами і додатковими атрибутами, такими як колір або

інтенсивність сигналу [59, 60]. Вони формуються на основі даних, отриманих із лазерних сканерів (LiDAR) або фотограмметричних реконструкцій [62].

$$P = \{p_i = (x_i, y_i, z_i, a_i)\}, i = 1, 2, \dots, N, \quad (2.1)$$

де:

P — множина точок, кожна з яких представлена як $p_i = (x_i, y_i, z_i, a_i)$;

p_i — окрема точка у просторі, задана координатами та додатковим атрибутом;

x_i, y_i, z_i — координати точки p_i у тривимірному просторі (для осей x , y та z);

a_i — додатковий атрибут точки, наприклад, колір або інтенсивність;

i — індекс точки, який змінюється від 1 до N ;

N — загальна кількість точок у множині P .



Рисунок 2.2 - Зображення тривимірного об'єкта в представленні точкових хмар.

Основна перевага точкових хмар — висока деталізація і простота збору даних, що робить їх ідеальними для моделювання поверхонь складних об'єктів. Проте цей формат не зберігає явних топологічних зв'язків між точками, що ускладнює подальшу обробку. Для вирішення цієї проблеми застосовуються методи згладжування (наприклад, k -NN), кластеризації для виділення груп точок і алгоритми побудови трикутних сіток, зокрема метод Делоне.

Крім того, для підвищення ефективності обробки використовуються методи зменшення розмірності, такі як PCA (Principal Component Analysis), які дозволяють спрощувати представлення точкових хмар без втрати ключових характеристик об'єкта.

2.1.2 Вокселі (Voxels)

Вокселі є регулярною тривимірною решіткою, де кожен елемент (воксель) зберігає інформацію про наявність об'єкта або його властивості, такі як щільність чи колір (див. рис. 2.3). Воксельне представлення використовується для внутрішнього моделювання об'єктів, наприклад, у медичних зображеннях або фізичних симуляціях [60].

$$V(x, y, z) = \{0, 1\}, \quad (2.2)$$

де:

$V(x, y, z)$ — значення вокселя (0 або 1);

(x, y, z) — координати точки по осям x , y та z .



Рисунок 2.3 - Зображення тривимірного об'єкта в представленні вокселів.

Головною перевагою вокселів є здатність моделювати внутрішню структуру об'єкта, що недоступно для інших форматів. Проте їх використання потребує значних обчислювальних ресурсів, особливо для обробки великих обсягів даних

[63]. Для зменшення обчислювальної складності застосовуються алгоритми октодерев, які ієрархічно представляють вокселі, і методи квантування для зменшення кількості елементів.

Також використовуються морфологічні операції, такі як ерозія та дилатація, для згладжування поверхонь і усунення шумів. Застосування процедур інтерполяції забезпечує реконструкцію неповних даних і підвищує якість моделі.

2.1.3 Полігональні сітки (Meshes)

Полігональні сітки створюють поверхню об'єкта на основі вершин (vertices), ребер (edges) і граней (faces). Кожна грань зазвичай представлена у вигляді трикутника або багатокутника, що дозволяє точно описати форму об'єкта (див. рис. 2.4). Цей формат широко використовується в комп'ютерній графіці, 3D-моделюванні та симуляціях.

$$M = (V, E, F), \quad (2.3)$$

де:

$V = \{v_1, v_2, \dots, v_n\}$ — набір вершин, кожна з яких визначається координатами (x, y, z) ;

$E = \{e_1, e_2, \dots, e_n\}$ — набір ребер, що з'єднують вершини;

$F = \{f_1, f_2, \dots, f_n\}$ — набір граней, які формують поверхню об'єкта, кожна з яких описується як набір вершин.

Полігональні сітки

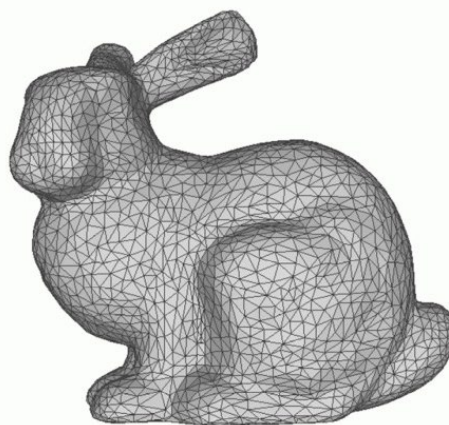


Рисунок 2.4 - Зображення тривимірного об'єкта в представленні полігональних сіток.

Основна перевага полігональних сіток полягає у високій точності геометричного представлення поверхонь. Проте для роботи з великими моделями потрібно знижувати обчислювальні витрати за допомогою редукції полігонів і адаптивного згладжування. Нормалізація нормалей використовується для забезпечення коректного відображення текстур і освітлення.

Серед алгоритмів обробки особливе місце займають методи реконструкції поверхонь, такі як Marching Cubes, які дозволяють генерувати полігональні сітки з воксельних даних, забезпечуючи точне моделювання складних форм. Інші підходи, такі як метод Лапласового згладжування, використовуються для підвищення якості відтворення геометрії.

2.2 Наявні методи попередньої обробки даних

Попередня обробка тривимірних даних відіграє важливу роль у забезпеченні ефективності алгоритмів класифікації та точності кінцевих результатів. Відповідна обробка дозволяє усунути шуми, нормалізувати масштаби, заповнити пропуски та адаптувати дані для подальшого використання в глибоких нейронних мережах.

Методи попередньої обробки точкових хмар

Фільтрація шумів

Фільтрація шумів є важливим елементом попередньої обробки даних, що забезпечує підвищення якості вхідних даних для подальшої обробки. У процесі роботи з тривимірними нейронними мережами особливо актуальним є усунення випадкових відхилень, які можуть призводити до значних похибок у результатах класифікації. Одним з ефективних методів зменшення шумів є використання алгоритму k-ближчих сусідів (k-NN) [66, 67].

Алгоритм k-NN дозволяє обчислити усереднене положення точок у локальному оточенні, що сприяє зниженню флуктуацій поверхні та створює ефект згладжування. Це особливо важливо при аналізі тривимірних структур, де шуми можуть з'являтися внаслідок помилок вимірювань або перешкод у процесі сканування [73]. Формально нове положення точки визначається за формулою:

$$p_i^{new} = \frac{1}{k} \sum_{j=1}^k p_j, \quad (2.4)$$

де:

p_i^{new} — нове положення точки після згладжування;

k — кількість найближчих сусідів, що враховуються під час усереднення;

p_j — координати сусідів.

Основні переваги фільтрації шумів за допомогою k-NN:

- зменшення локальних флуктуацій: усереднення координат точок дозволяє згладжувати дрібні нерівності поверхні;
- простота реалізації: алгоритм легко реалізується та адаптується до різних типів даних;

- гнучкість параметрів: параметр k може бути налаштований відповідно до конкретних вимог дослідження, забезпечуючи компроміс між ступенем згладжування та збереженням деталей структури;
- стійкість до викидів: метод знижує вплив окремих аномальних точок на загальну форму об'єкта.

Оптимальне значення параметра k залежить від густини точок у досліджуваній області. Занадто мале значення може призвести до недостатнього згладжування, тоді як занадто велике значення може викликати втрату деталей у структурі. Практичний підхід до вибору параметра включає тестування декількох значень на навчальних даних та оцінку результатів за допомогою метрик якості.

Нормалізація координат

Нормалізація координат є одним із ключових елементів попередньої обробки даних, що спрямований на усунення впливу масштабу, розмірів і орієнтації об'єкта [69]. Вона забезпечує приведення значень координат до єдиного діапазону, зазвичай $[0,1]$ або $[-1,1]$. Такий підхід підвищує стабільність алгоритмів машинного навчання, забезпечуючи коректність порівняння даних та прискорюючи процес навчання моделей.

Формально нормалізація виконується за допомогою наступного перетворення:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}, \quad (2.5)$$

де:

x' — нормалізоване значення координати;

x — початкове (ненормалізоване) значення координати;

x_{min} — мінімальне значення координати у вибірці;

x_{max} — максимальне значення координати у вибірці.

Переваги нормалізації координат:

- усунення масштабних відмінностей: Перетворення дозволяє ефективно обробляти дані різних розмірностей, зводячи їх до одного масштабу;
- покращення збіжності алгоритмів: Уніфікація діапазону значень координат знижує чутливість моделей до початкових умов та прискорює оптимізацію параметрів;
- зменшення впливу аномалій: Перетворення обмежує вплив викидів, що мають надто великі або малі значення, знижуючи спотворення в навчальних даних;
- сумісність із різними методами класифікації: Деякі алгоритми, зокрема ті, що базуються на евклідовій відстані (наприклад, k-NN або SVM), чутливі до різниці масштабів, і нормалізація допомагає забезпечити коректність обчислень.

У випадку обробки тривимірних даних нормалізація допомагає узгодити масштаб координат уздовж усіх осей (x , y , z). Це дозволяє нейронним мережам зосередитися на виявленні патернів у структурі об'єктів, а не на абсолютних значеннях координат [75]. Нормалізація також важлива при роботі з даними, отриманими з різних джерел, де масштабування може відрізнятися.

Видалення аномалій

Видалення аномалій також є важливим елементом підготовки даних, який спрямований на виявлення та усунення точок, що суттєво відрізняються від основного розподілу даних. Аномалії можуть виникати внаслідок похибок вимірювань, шумів або унікальних, нестандартних властивостей об'єкта. Наявність таких точок може значно знижувати ефективність класифікації та роботи тривимірних нейронних мереж.

У цьому розділі використовується кластерний аналіз для ідентифікації ізольованих точок, які знаходяться на значній відстані від своїх сусідів.

Критерієм для видалення є перевищення порогового значення відстані, що формалізується наступною умовою:

$$D(p_i, p_j) > \tau \rightarrow \text{видалення}, \quad (2.6)$$

де:

$D(p_i, p_j)$ — відстань між точками p_i та p_j ;

τ — порогове значення, яке визначає максимальну допустиму відстань між точками, щоб вони не вважалися ізольованими.

Методика ідентифікації аномалій:

- обчислення відстаней: Для кожної точки обчислюється відстань до найближчих сусідів (наприклад, за евклідовою метрикою);
- порівняння з порогом: Якщо відстань до найближчих сусідів перевищує заданий поріг τ , точка класифікується як аномальна;
- видалення або маркування: Аномальні точки можуть бути видалені або марковані для подальшої обробки.

Порогове значення визначається експериментально або на основі статистичних характеристик даних. Один із підходів це використання середнього значення відстаней плюс кілька стандартних відхилень. Альтернативно, можна застосовувати алгоритми кластеризації, такі як DBSCAN, які автоматично виявляють аномалії, враховуючи щільність точок.

Використання кластерного аналізу для виявлення аномалій дозволяє ефективно зменшити шум у вхідних даних, що сприяє підвищенню точності моделей машинного навчання. Такий підхід є особливо корисним для роботи з тривимірними нейронними мережами, де стабільність і точність даних є критично важливими.

Методи попередньої обробки вокселів

Фільтрація шумів

Обробка тривимірних даних, представлених у вигляді вокселів, вимагає ефективних методів фільтрації шумів для збереження структури об'єкта. Одним із підходів є використання морфологічних операцій, таких як ерозія та дилатація. Ці операції застосовуються безпосередньо до воксельних даних, дозволяючи видаляти артефакти та відновлювати геометричні особливості об'єктів.

Ерозія — це морфологічна операція, яка зменшує область об'єкта шляхом виключення вокселів, що не задовольняють структурному елементу. Вона ефективно видаляє невеликі шумові точки та роз'єднані компоненти.

Формально, ерозія визначається як:

$$A \ominus B = \{z | B_z \subseteq A\}, \quad (2.7)$$

де:

A — початковий набір вокселів (об'єкт);

B — структурний елемент;

B_z — зсув структурного елемента B до позиції z ;

z — позиція, до якої зсувається структурний елемент B .

Операція виконується таким чином, що B повинен повністю поміститися всередині A для збереження вокселя z .

Ерозія використовується для видалення шумів, згладжування країв і зменшення розмірів об'єкта, особливо для усунення окремих ізольованих вокселів.

Дилатація, навпаки, є операцією розширення, яка додає вокселі до меж об'єкта, заповнюючи пропуски або дрібні отвори. Це дозволяє відновлювати структуру після ерозії або заповнювати дрібні порожнини.

Формально, дилатація визначається як:

$$A \oplus B = \{z | B_z \cap A \neq \emptyset\}, \quad (2.8)$$

де:

A — початковий набір вокселів (об'єкт);

B — структурний елемент;

B_z — зсув структурного елемента B до позиції z ;

z — позиція, до якої зсувається структурний елемент B .

Додаткові вокселі додаються в місця, де структурний елемент B перетинається з об'єктом A .

Дилатація використовується для відновлення геометрії об'єктів, заповнення прогалин і посилення контурів після етапу ерозії.

Нормалізація щільності

Нормалізація щільності вокселів (2.5) є важливим етапом попередньої обробки даних у тривимірних нейронних мережах. Її основне призначення полягає в уніфікації просторового розподілу точок, що забезпечує масштабованість, ефективність і коректність роботи алгоритмів машинного навчання.

Основними завданнями нормалізації є усунення неоднорідностей, вирівнювання щільності точок і спрощення обчислень. Дані, отримані зі сканерів або сенсорів, часто мають варіації в щільності в різних регіонах. Приведення щільності до фіксованого рівня дозволяє розв'язати цю проблему та полегшити інтеграцію з іншими форматами даних.

Повторна вибірка вокселів застосовується для зміни розміру або щільності шляхом інтерполяції або вибірки, що оптимізує обсяг даних без втрати критичної інформації. Уніфікація масштабу зменшує кількість вокселів у ділянках з

надмірною щільністю, тоді як заповнення прогалин додає вокселі в області з низькою щільністю.

Після нормалізації щільності часто застосовуються морфологічні операції, такі як ерозія та дилатація. Ерозія видаляє ізольовані точки та дрібні артефакти, а дилатація відновлює з'єднання між частинами об'єкта, заповнює порожнини та відновлює структуру. Комбіновані операції, такі як відкриття та закриття, дозволяють зберігати важливі деталі об'єктів, одночасно усуваючи небажані шуми.

Нормалізація щільності разом із морфологічною обробкою підвищує якість вхідних даних для нейронних мереж, забезпечує точність сегментації та класифікації, а також полегшує обробку великих обсягів інформації.

Реконструкція поверхонь

Реконструкція поверхонь є важливим етапом аналізу тривимірних даних, що передбачає побудову геометричних моделей об'єктів на основі воксельних представлень. Одним із найпоширеніших алгоритмів для цієї задачі є метод Marching Cubes.

Метод Marching Cubes генерує полігональні сітки шляхом інтерполяції значень вокселів, створюючи згладжені поверхні, які можуть бути використані для візуалізації, аналізу або подальшої обробки. Основна ідея алгоритму полягає в аналізі локальних кубічних комірок, сформованих групами з восьми вокселів, і побудові трикутних полігонів, що апроксимують поверхню.

Інтерполяція значень використовується для точнішого визначення положення вершин полігонів. Наприклад, для двох сусідніх вокселів значення на поверхні обчислюється за формулою:

$$v(p) = \frac{p_1 + p_2}{2}, \quad (2.9)$$

де:

$v(p)$ — інтерпольоване значення в точці p , яке визначається як середнє між значеннями сусідніх вокселів;

p_1, p_2 — значення двох сусідніх вокселів.

Це дозволяє згладити переходи між вокселями, забезпечуючи плавність відновленої поверхні.

Основними перевагами Marching Cubes є висока точність, можливість працювати з великими наборами даних і підтримка складних тривимірних структур. Метод широко використовується в комп'ютерній графіці, медичній візуалізації та науковому моделюванні.

Реконструкція поверхонь відіграє критичну роль у тривимірних нейронних мережах, оскільки забезпечує зв'язок між воксельними даними та полігональними моделями, які використовуються в подальшому аналізі або відображенні.

Методи попередньої обробки полігональних сіток

Фільтрація шумів

Лапласове згладжування усуває нерівності, оптимізуючи положення вершин і створюючи плавні переходи між гранями.

$$v_i^{new} = v_i + \lambda \sum_{j \in N(i)} (v_j - v_i), \quad (2.10)$$

де:

v_i^{new} — нове положення вершини i після згладжування;

v_i — початкове положення вершини i ;

$N(i)$ — набір сусідніх вершин для вершини i ;

v_j — положення сусідньої вершини j , яка належить до набору $N(i)$;

λ — коефіцієнт згладжування, що контролює інтенсивність корекції.

Ця формула реалізує переміщення кожної вершини в напрямку середнього положення її сусідів, що дозволяє зменшити локальні коливання та покращити гладкість поверхні. Значення параметра λ зазвичай обирається в діапазоні від 0 до 1, щоб забезпечити баланс між збереженням деталей і усуненням шумів.

Нормалізація геометрії

Нормалізація геометрії є важливим елементом підготовки полігональних сіток для подальшої обробки та візуалізації. Вона включає вирівнювання довжин ребер і коригування нормалей, що забезпечує стабільність і сумісність з алгоритмами освітлення та текстуровання.

Рівномірний розподіл довжин ребер дозволяє усунути деформації сітки, зберігаючи пропорції та деталі структури. Це покращує точність обчислень під час симуляцій і створює однорідну сітку, яка є більш придатною для алгоритмів обробки.

Коригування нормалей відіграє ключову роль в обчисленні світлових ефектів, таких як відблиски та тіні. Правильна орієнтація нормалей підвищує реалістичність візуалізації й забезпечує коректну інтеграцію текстур.

Дані процеси є критично важливими для забезпечення якості 3D-моделей, що використовуються в машинному навчанні, комп'ютерній графіці та симуляціях фізичних процесів.

Редукція полігонів

Редукція полігонів є важливим елементом оптимізації структури сіток для підвищення продуктивності під час візуалізації та аналізу великих моделей. Основна мета полягає в зменшенні кількості граней без значних втрат деталей, що дозволяє зберегти геометричну точність і візуальну якість об'єкта.

Процес редукції ґрунтується на мінімізації загальної похибки, яка описується наступною формулою:

$$E = \sum_{i=1}^n w_i (d_i)^2, \quad (2.11)$$

де:

E — загальна похибка, що відображає відхилення спрощеної сітки від початкової;

w_i — вагові коефіцієнти, що визначають важливість окремих точок або ділянок;

d_i — відстань від вершини i до наближеної поверхні;

n — загальна кількість точок у множині.

Методика редукції зазвичай включає такі етапи:

- ідентифікація граней або вершин, які можуть бути об'єднані або видалені без суттєвої втрати геометричних особливостей;
- перерахунок положень вершин із врахуванням оптимізації форми поверхні;
- перевірка та оновлення нормалей і текстурних координат для забезпечення збереження візуальної якості.

Алгоритми редукції, такі як Quadric Error Metrics (QEM), використовуються для ефективного балансування між кількістю полігонів і точністю моделі. Вони широко застосовуються в комп'ютерній графіці, візуалізації медичних даних і геометричному моделюванні.

Оптимізація полігональних сіток особливо важлива при роботі з тривимірними нейронними мережами, де швидкість обробки та точність геометрії впливають на якість навчання й результатів класифікації.

2.3 Методи колориметричної корекції

Колориметрична корекція є важливим етапом підготовки даних для аналізу текстур і класифікації зображень [80]. Вона включає перетворення в опонентний простір кольорів, що дозволяє виділити яскравість та хроматичність, підвищуючи інформативність даних.

Перетворення компонентів кольору виконується за наступними формулами:

$$O_1 = \frac{R - G}{\sqrt{2}}, O_2 = \frac{R + G - 2B}{\sqrt{6}}, O_3 = \frac{R + G + B}{\sqrt{3}}, \quad (2.12)$$

де:

O_1 — перший компонент в опонентному просторі, який визначає різницю між червоним (R) та зеленим (G) кольорами;

O_2 — другий компонент в опонентному просторі, що враховує співвідношення між червоним (R), зеленим (G) та синім (B) кольорами;

O_3 — третій компонент в опонентному просторі, який відображає яскравість кольору, використовуючи всі три компоненти: червоний (R), зелений (G) і синій (B);

R, G, B — компоненти кольору у вихідному просторі, що відповідають червоному (R), зеленому (G) та синьому (B) каналам відповідно.

Опонентний простір кольорів відображає інформацію про колірні відмінності й яскравість, що особливо корисно при аналізі текстур або сегментації об'єктів у тривимірних моделях.

Цей підхід дозволяє:

- виділяти кольорові градієнти, покращуючи розпізнавання текстур;
- знижувати вплив освітлення завдяки розділенню інформації про яскравість та кольоровість;

- підвищувати ефективність класифікації, використовуючи колірні особливості для покращення результатів навчання моделей.

Методи попередньої обробки даних, включаючи фільтрацію, нормалізацію, реконструкцію та редукцію полігонів, забезпечують стабільність і адаптацію тривимірних моделей до алгоритмів машинного навчання. Колориметрична корекція доповнює ці методи, розширюючи можливості аналізу текстур і структур. І розрахувавши середні відсоткові покращення кількості втрат (див. рис. Г.37), можна зазначити що використання опонентної колориметричної корекції призводить до зменшення втрат в діапазоні від 8,64% до 53,33%, що в свою чергу веде до покращення точності та стабільності класифікації.

2.4 Теоретичні аспекти архітектури класифікатора

Розробка ефективного класифікатора для тривимірних зображень вимагає детального аналізу архітектури мережі, яка здатна враховувати просторові особливості даних [36]. Побудова такої архітектури базується на принципах організації тривимірних топологій, що забезпечують адаптивність, масштабованість і точність обробки вхідних сигналів [48].

Тривимірні топології відіграють ключову роль у представленні об'єктів із збереженням їхньої геометричної структури. Вони дозволяють формувати моделі, які здатні ефективно працювати з різними форматами даних, такими як вокселі, точкові хмари та полігональні сітки. Завдяки цьому забезпечується можливість використання складних підходів до виявлення ознак і класифікації.

Подальший розгляд зосереджено на теоретичному обґрунтуванні тривимірних топологій, їхніх властивостях та алгоритмах побудови. Цей аналіз є необхідною основою для розробки архітектур нейронних мереж, здатних обробляти великі обсяги інформації з урахуванням просторових взаємозв'язків між елементами даних.

2.4.1 Теоретичне обґрунтування тривимірних топологій

Аналіз існуючих підходів у проектуванні топологій є важливим етапом для визначення їхніх переваг, обмежень та можливостей вдосконалення [38, 43].

Існуючі топології

Нейронні мережі використовують різні топології для обробки даних. Основні типи включають:

- послідовна топологія;
- згорткова топологія;
- рекурентна топологія.

Послідовна топологія

Послідовна топологія є найпростішою та найбільш інтуїтивно зрозумілою структурою для побудови нейронних мереж. У цій архітектурі кожен нейрон одного шару з'єднаний з усіма нейронами наступного шару. Це означає, що кожен вихід одного шару є входом для всіх нейронів наступного шару. Така структура забезпечує повну взаємодію між нейронами та дає змогу мережі навчитися складним нелінійним залежностям у даних [52]. Однак основний недолік цього підходу полягає в обчислювальній складності, яка зростає експоненційно з кількістю нейронів і шарів. Наприклад, якщо в першому шарі 100 нейронів, а в другому – 200, то загалом буде 20 000 з'єднань. Це може призводити до перенавчання і вимагати великих обчислювальних ресурсів для тренування.

У послідовній топології кожен нейрон пов'язаний з усіма нейронами наступного шару. Вихід нейрона обчислюється за формулою:

$$y_i = f \left(\sum_{j=1}^n w_{ij} x_j + b_i \right), \quad (2.13)$$

де:

y_i — вихід i -го нейрона;

x_j — вхідні значення від j -го нейрона попереднього шару;

w_{ij} — ваги зв'язків між нейронами;

b_i — зміщення для i -го нейрона;

f — функція активації;

n — кількість нейронів у попередньому шарі.

Згорткова топологія

Згорткова топологія використовується в згорткових нейронних мережах (CNN), які спеціально розроблені для обробки даних із просторовою структурою, таких як зображення [44]. Основна ідея полягає в тому, щоб мережа навчалася виявляти локальні ознаки, такі як текстури, краї, кути або кольорові градієнти. Згорткова топологія працює за рахунок застосування фільтрів (ядер згортки), які ковзають по зображенню і створюють карти ознак. Кожен фільтр шукає специфічний шаблон у зображенні, зберігаючи локальність зв'язків. Потім отримані карти ознак проходять через шари підвибірки (пулінгу), які зменшують розмірність даних, роблячи мережу менш чутливою до зсувів або масштабів об'єктів [68]. Перевагами цього підходу є значне зменшення кількості параметрів у порівнянні з щільними мережами, підвищена ефективність і масштабованість для великих вхідних даних, а також можливість автоматичного виділення ознак без необхідності ручної інженерії ознак.

Згорткова топологія обробляє локальні області вхідних даних за допомогою згорткових фільтрів. Вихід нейрона для кожного елемента згортки розраховується так:

$$y_{i,j} = f \left(\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} w_{m,n} x_{i+m,j+n} + b \right), \quad (2.14)$$

де:

$y_{i,j}$ — вихідне значення в позиції (i, j) на карті ознак після згортки;

$x_{i+m,j+n}$ — значення вхідних даних у відповідному вікні згортки для позиції (i, j) ;

$w_{m,n}$ — ваги ядра згортки розміром M на N ;

b — зміщення (bias), яке додається до результату згортки;

f — функція активації, яка застосовується до результату для нелінійного перетворення;

M, N — розміри ядра згортки (висота M та ширина N);

m — вертикальний індекс елемента в ядрі згортки, що змінюється в межах $m=0,1,\dots,M-1$;

n — горизонтальний індекс елемента в ядрі згортки, що змінюється в межах $n=0,1,\dots,N-1$.

Рекурентна топологія

Рекурентні нейронні мережі (RNN) призначені для роботи з послідовними або часовими даними, такими як текст, аудіо або відео [40]. Головною особливістю цієї топології є наявність зворотних зв'язків, які дозволяють зберігати інформацію про попередні елементи послідовності. У традиційних RNN вихід кожного нейрона може бути повернений на попередні шари, забезпечуючи пам'ять про попередній стан. Це дозволяє враховувати контекст і обробляти залежності в послідовностях даних. Наприклад, аналіз тексту для прогнозування наступного слова в реченні, розпізнавання мови або музики на основі попередніх звуків, прогнозування часових рядів, таких як фінансові дані [54]. Основними недоліками рекурентних мереж є проблема зникнення або вибуху градієнтів під час тренування, особливо для довгих послідовностей, а також високі обчислювальні витрати при роботі з великими наборами даних. Для вирішення цих проблем були розроблені вдосконалені архітектури, такі як LSTM (Long

Short-Term Memory) і GRU (Gated Recurrent Unit), які краще зберігають довгострокові залежності.

Рекурентні нейронні мережі (RNN) мають зворотні зв'язки для збереження інформації про попередні стани. Формула розрахунку виходу виглядає так:

$$h_t = f(W_h h_{t-1} + W_x x_t + b), \quad (2.15)$$

де:

h_t — прихований стан у момент часу t ;

h_{t-1} — прихований стан у попередній момент часу ($t-1$);

x_t — вхідні дані в момент часу t ;

W_h, W_x — матриці ваг для прихованих станів і входів відповідно;

b — зміщення;

f — функція активації.

Локальні мережі

Локальні зв'язки дозволяють зосереджуватись на певних областях вхідних даних [45]. Наприклад, згорткові мережі використовують фільтри для виявлення ключових особливостей, таких як контури або текстури [61]. Після цього дані стискаються за допомогою пулінгу, що допомагає зменшити їх розмірність і підвищити стійкість до зміщень [79].

Однак такий підхід не враховує зв'язки між віддаленими частинами зображення, що обмежує його застосування для складних задач. Це вирішується за допомогою тривимірних топологій.

Тривимірна топологія

Тривимірна топологія враховує просторову структуру вхідних даних. Основна ідея полягає в тому, щоб представити дані у вигляді багатовимірного масиву (кубічної матриці) і зберегти їх просторові зв'язки.

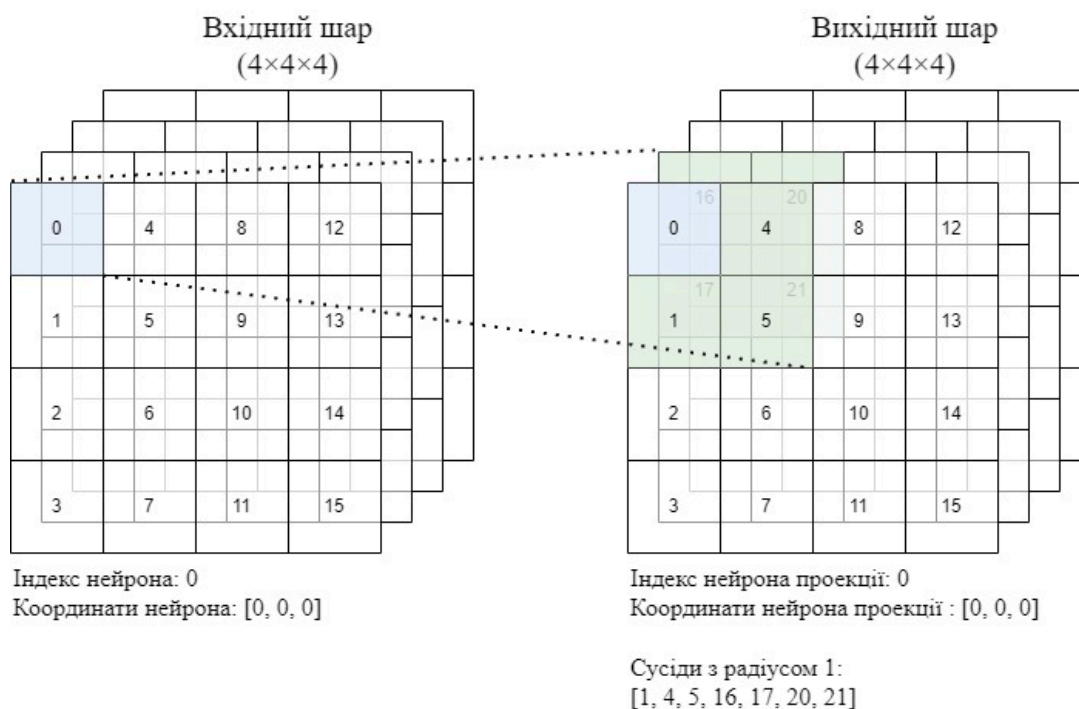


Рисунок 2.5 - Проекція нейрона на вихідний шар та сусіди в ньому.

Для нейрона з індексом 0 і координатами (0, 0, 0), проекцією в вихідний шар такого ж розміру буде такий самий нейрон з координатами (0, 0, 0) (див. рис. 2.5) а індекс розрахуємо за формулою:

$$index = x + y * n + z * n^2, \quad (2.16)$$

де:

$index$ — індекс нейрона в одномірному масиві;

x — координата нейрона вздовж осі X ;

y — координата нейрона вздовж осі Y ;

z — координата нейрона вздовж осі Z ;

n — розмір сітки по одній осі (оскільки сітка кубічна, її розмірність $n \times n \times n$).

Підставимо елементи в формулу і отримаємо індекс.

$$index = 0 + 0 * 4 + 0 * 4^2 = 0, \quad (2.17)$$

Далі через зміщення розраховуються координати сусідів проекції нейрона в вихідному шарі, і для кожного розраховується індекс нейрона в одномірному векторі.

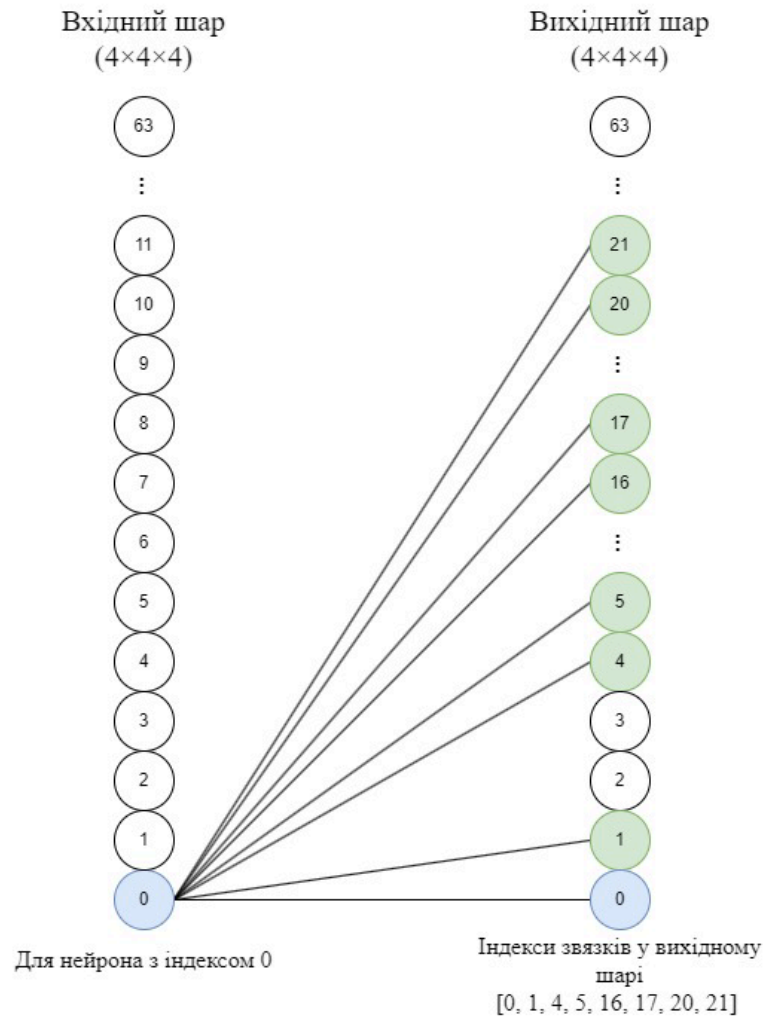


Рисунок 2.6 - Локальні зв'язки нейрона між шарами.

Зв'язки будуються (див. рис. 2.6) тільки з нейроном проекцією (синій колір) та його сусідами (зелений колір).

У тривимірній топології враховується просторовий зв'язок між елементами вхідних даних, організованих у вигляді тривимірної матриці. Нейрон пов'язаний лише з локальними сусідами у своїй околиці, що враховує його положення у тривимірному просторі.

$$y_{i,j,k} = f \left(\sum_{p=-P}^P \sum_{q=-Q}^Q \sum_{r=-R}^R w_{p,q,r} x_{i+p,j+q,k+r} + b \right), \quad (2.18)$$

де:

$y_{i,j,k}$ — вихідне значення нейрона в позиції (i, j, k) у тривимірному об'ємі;

$x_{i+p,j+q,k+r}$ — вхідне значення сусіда в околиці розміром $(2P + 1)(2Q + 1)(2R + 1)$;

$w_{p,q,r}$ — ваги зв'язків між нейронами для всіх локальних сусідів у тривимірному просторі;

b — зміщення;

f — функція активації;

P, Q, R — радіуси околиці у кожному з трьох вимірів (відповідно вздовж осей x, y, z);

p, q, r — індекси, що змінюються в межах $[-P, P], [-Q, Q], [-R, R]$ відповідно, щоб охопити всі сусідні елементи у тривимірній околиці.

Тривимірні топології є перспективними для аналізу складних структурованих даних, таких як медичні знімки (КТ, МРТ) або 3D-моделі об'єктів. Вони забезпечують масштабованість і ефективність обчислень навіть при великих обсягах інформації.

2.4.2 Розгортка та згортка в тривимірних шарах

Розгортка та згортка є ключовими операціями в нейронних мережах, особливо в архітектурах, призначених для обробки тривимірних даних, таких як 3D-зображення, томографічні дані або відео потоки. Ці операції дозволяють змінювати розмірність вхідних даних для досягнення різних цілей: підвищення деталізації (розгортка) або зменшення обчислювальних витрат і узагальнення ознак (згортка) [76].

Розгортка

Розгортка використовується для збільшення розмірності даних, коли потрібно відновити або поліпшити деталізацію, наприклад, у завданнях сегментації чи генерації зображень. Найпростішим методом є "найближчий сусід", при якому значення сусідів повторюються для збільшення розмірності

[78]. Він є ефективним, проте може створювати "драбинчасті" артефакти. Інший підхід — лінійна інтерполяція, що застосовує згладжування між сусідніми значеннями в трьох вимірах (x, y, z).

Більш складним і точним методом є транспонована згортка, яка використовує згорткову операцію для збільшення розмірності. Формула для виходу в цьому випадку виглядає так:

$$O = (I - 1) * S + K - 2P, \quad (2.19)$$

де:

O — розмір вихідного тензора;

I — розмір вхідного тензора;

S — крок, на скільки зміщується вікно згортки під час обчислення;

K — розмір ядра згортки ;

P — заповнення, тобто додавання нулів навколо вхідного тензора для контролю розміру виходу.

Транспонована згортка забезпечує високу якість відновлення деталей, однак вимагає більше обчислювальних ресурсів. Апсемплінг дозволяє відновлювати високу роздільну здатність, покращувати деталізацію для генеративних задач і підтримувати сумісність з архітектурами, що вимагають симетричної обробки даних.

В запропонованій топології розгортка реалізується за рахунок проектування нейрона з вхідного шару меншого розміру до шару більшого розміру. А також за рахунок сусідів нейрона проекції.

Згортка

Згортка спрямована на зменшення розмірності просторових даних. Її основна мета — зберегти найбільш важливі ознаки вхідних даних, водночас зменшивши їхню кількість для спрощення обчислень.

Зменшення розмірності також може виконуватися за допомогою згорткових операцій з кроком більше одиниці, що дозволяє зберігати найбільш значущі ознаки. Згортка забезпечує зменшення кількості параметрів і обчислень, підвищує стійкість до шумів і дозволяє виділяти основні особливості без втрати значної інформації.

У розроблюваному класифікаторі розгортка та згортка реалізовані на основі тривимірної топології. Кожен нейрон пов'язаний із певною групою сусідніх нейронів у наступному або попередньому шарі, що дозволяє зберігати просторові зв'язки.

При розгортці (див. рис. 2.7) кожен нейрон породжує множину нових нейронів, які розташовуються в локальній околиці, зберігаючи структуру вихідних даних. Це забезпечує плавне масштабування та деталізацію.

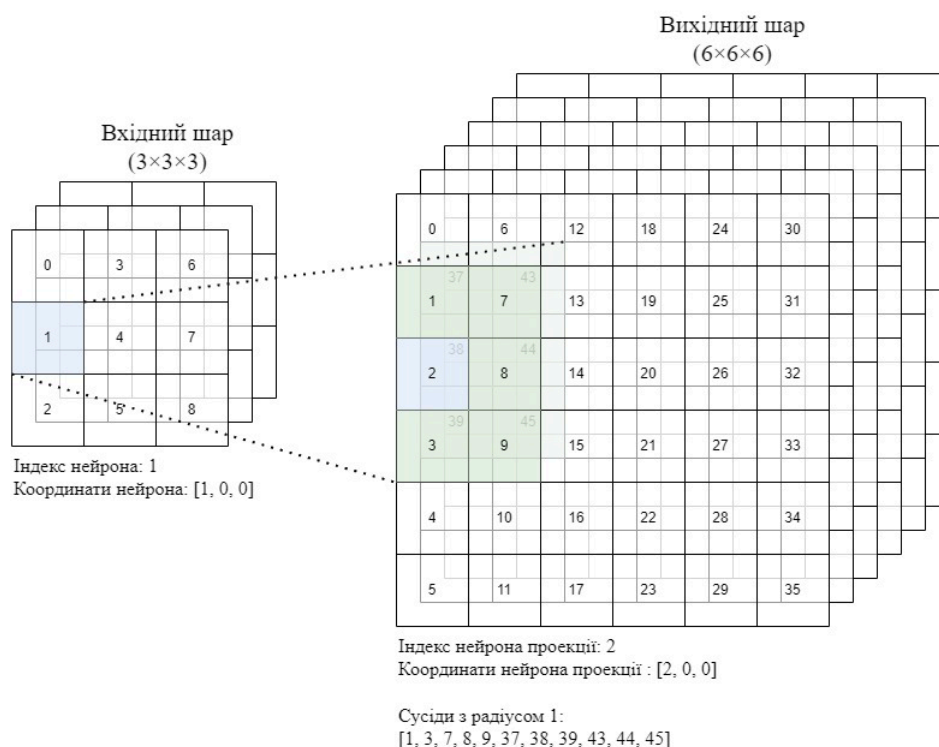


Рисунок 2.7 - Представлення розгортки в тривимірній топології.

Зворотній процес це згортка (див. рис. 2.8), вона об'єднує кілька нейронів у один, зберігаючи головні ознаки. Наприклад, для нейрона вищого рівня

використовується вікно сусідів у попередньому шарі, щоб узагальнити інформацію.

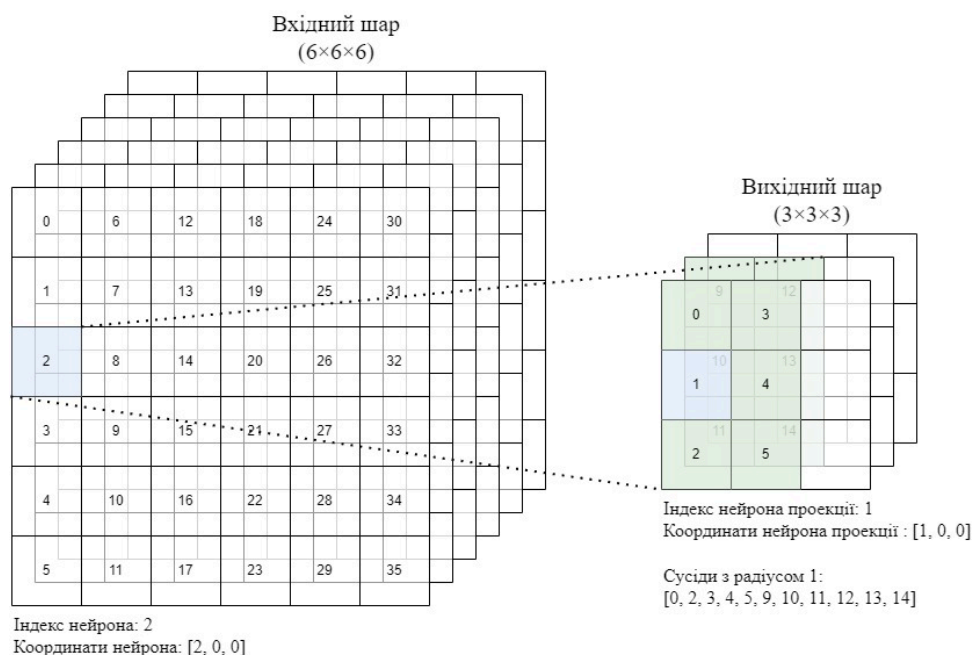


Рисунок 2.8 - Представлення згортки в тривимірній топології.

Така організація дозволяє досягти гнучкого масштабування розмірностей, зберігаючи важливі зв'язки в даних, що робить її ефективною для обробки 3D-структур.

2.5 Методи оптимізації обчислень

2.5.1 Існуючі методи оптимізації

Оптимізація нейронних мереж є критично важливою складовою для підвищення ефективності та точності моделей класифікації зображень [39, 57]. Серед існуючих методів оптимізації виділяються стохастичний градієнтний спуск, адаптивна оптимізація, регуляризація та проріджування ваг.

Стохастичний градієнтний спуск (SGD) є одним із найпоширеніших методів оптимізації. Він оновлює ваги моделі на основі обчислених градієнтів функції втрат. Його перевага полягає в простоті реалізації та ефективності для великих наборів даних. Водночас, недоліками є повільна збіжність та можливість потрапляння в локальні мінімуми.

Адаптивні методи оптимізації, такі як Adam [42], RMSprop та Adagrad, використовують адаптивні коефіцієнти навчання для прискорення збіжності:

- Adam поєднує переваги двох методів — моментів першого та другого порядків, забезпечуючи швидшу збіжність навіть для нерівномірно масштабованих даних;
- RMSprop стабілізує навчання шляхом масштабування градієнтів, що робить його ефективним для нерівномірних розподілів даних;
- Adagrad автоматично налаштовує швидкість навчання для кожного параметра, що особливо корисно для розріджених даних.

Регуляризація спрямована на зменшення перенавчання моделей [41]:

- Dropout випадково вимикає нейрони під час навчання, що змушує модель краще узагальнювати результати;
- L2-регуляризація додає штраф за великі ваги до функції втрат, обмежуючи їх зростання та запобігаючи перенавчанню.

Проріджування ваг передбачає видалення малозначущих зв'язків у нейронній мережі, що зменшує розмір моделі та підвищує її обчислювальну ефективність. Такий підхід забезпечує кращу узагальнювальну здатність моделі за рахунок спрощення її структури.

Для розроблюваного класифікатора з локальними зв'язками ми використаємо та оптимізуємо методи регуляризації та проріджування.

2.5.2 Метод видалення зв'язків

Основна ідея методу видалення слабких зв'язків базується на припущенні, що ваги нейронних зв'язків із низькими абсолютними значеннями не роблять значущого внеску в результат роботи мережі [58]. З математичної точки зору слабкі зв'язки є такими, для яких абсолютні значення ваг нижчі за певний поріг, визначений на основі середньої активності нейрона.

Процес оптимізації виконується ітеративно між епохами навчання і включає кілька етапів:

- обчислення середнього абсолютного значення вагових коефіцієнтів для кожного нейрона;
- встановлення індивідуального порогу для видалення ваг на основі отриманого середнього значення;
- формування маски збереження зв'язків;
- динамічне коригування порогових значень у процесі навчання для забезпечення адаптивності алгоритму.

Обчислення середнього абсолютного значення вагових коефіцієнтів

Для кожного нейрона i визначається середнє абсолютне значення вагових коефіцієнтів зв'язків:

$$\bar{w}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} |w_{ij}|, \quad (2.20)$$

де:

\bar{w}_i — середнє абсолютне значення вагових коефіцієнтів нейрона i ;

n_i — кількість вихідних зв'язків для нейрона i ;

w_{ij} — вага зв'язку між нейронами i та j .

Встановлення порогового значення для видалення ваг

Для кожного нейрона встановлюється поріг видалення вагових коефіцієнтів на основі коефіцієнта пропорційності T :

$$\tau_i = T * \bar{w}_i, \quad (2.21)$$

де:

τ_i — порогове значення для нейрона i ;

T — параметр регулювання, що визначає рівень чутливості до видалення слабких зв'язків;

\bar{w}_i — середнє абсолютне значення вагових коефіцієнтів нейрона i .

Параметр T є гіперпараметром і вибирається на етапі попередньої валідації для забезпечення оптимального балансу між продуктивністю та точністю [49].

Формування маски для збереження вагових коефіцієнтів

На основі порогового значення створюється маска, що визначає, які зв'язки залишити:

$$M_{ij} = \begin{cases} 1, \text{ якщо } |w_{ij}| \geq \tau_i \\ 0, \text{ в іншому випадку} \end{cases} \quad (2.22)$$

де:

M_{ij} — маска збереження ваг для зв'язку між нейронами i та j ;

w_{ij} — вага зв'язку між нейронами i та j ;

τ_i — порогове значення для нейрона i , яке визначає межу для збереження ваги зв'язку.

Оновлення вагових коефіцієнтів

Залишені зв'язки оновлюються за формулою:

$$\bar{w}_{ij} = M_{ij} * w_{ij}, \quad (2.23)$$

де:

\bar{w}_{ij} — оновлене значення ваги після застосування маски;

M_{ij} — маска збереження ваг для зв'язку між нейронами i та j ;

w_{ij} — вага зв'язку між нейронами i та j .

Адаптивне коригування порогів

Для уникнення надмірного видалення ваг на ранніх етапах навчання параметр T може бути скоригований відповідно до метрик втрат:

$$T_{new} = T_{prev} * (1 - \eta * \Delta L), \quad (2.24)$$

де:

T_{new} — нове значення порогового коефіцієнта;

T_{prev} — попереднє значення порогового коефіцієнта до коригування;

η — швидкість корекції;

ΔL — зміна функції втрат за останню епоху навчання.

Переваги алгоритму

Завдяки динамічному скороченню кількості зв'язків алгоритм зменшує обчислювальні витрати, що робить його придатним для масштабованих задач і великих наборів даних.

Застосування порогового аналізу зменшує ймовірність перенавчання за рахунок виключення шумових зв'язків, які можуть перешкоджати узагальненню результатів.

Використання динамічного коригування порогів забезпечує баланс між збереженням інформативних зв'язків і зниженням надмірності моделі.

Алгоритм можна застосовувати до будь-яких топологій нейронних мереж, зокрема до тривимірних структур, де зв'язки відображають просторові взаємозв'язки між елементами даних.

Запропонований алгоритм динамічного видалення слабких зв'язків у тривимірних нейронних мережах забезпечує оптимізацію їхньої структури на основі аналізу вагових коефіцієнтів. Його математичне обґрунтування базується

на концепціях вагової регуляризації та порогового відбору, що дозволяє зберігати високу точність класифікації при зниженні обчислювальної складності.

Завдяки використанню адаптивних порогів алгоритм демонструє гнучкість і ефективність у задачах з різноманітними типами вхідних даних і є перспективним для подальших досліджень і вдосконалень у галузі тривимірного машинного навчання.

2.5.3 Метод створення зв'язків

Процес створення нових зв'язків ґрунтується на припущенні, що нейрони з високою середньою вагою (сильними зв'язками) мають високий потенціал до подальшого розвитку [50]. Тому для таких нейронів додаються нові з'єднання, що дозволяють розширити їхню обчислювальну здатність.

Процедура включає наступні кроки:

- визначення середньої ваги для кожного нейрона;
- порівняння отриманого значення із заданим порогом створення зв'язків;
- генерація нових з'єднань у локальному просторі за допомогою асиметричних тривимірних індексів;
- встановлення нових ваг на основі поточної активності нейрона.

Обчислення середньої ваги нейрона

Для нейрона з індексом i обчислюється середнє абсолютне значення ваг:

$$\bar{w}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} |w_{ij}|, \quad (2.25)$$

де:

\bar{w}_i — середнє абсолютне значення вагових коефіцієнтів нейрона i ;

n_i — кількість вихідних зв'язків для нейрона i ;

w_{ij} — вага зв'язку між нейронами i та j .

Перевірка порогового значення для створення зв'язків

Вага порівнюється із порогом Θ_{create} :

$$\bar{w}_i > \Theta_{create} \quad (2.26)$$

де:

\bar{w}_i — середнє абсолютне значення вагових коефіцієнтів нейрона i ;

Θ_{create} — порогове значення, що визначає необхідність створення нових з'єднань.

Перетворення індексу у тривимірні координати

Нехай n — розмірність вхідного шару, а m — розмірність вихідного шару. Індекс нейрона у вхідному шарі переводиться у тривимірні координати:

$$(z, y, x) = \left(\frac{i}{n^2}, \frac{i \% n^2}{n}, i \% n \right), \quad (2.27)$$

де:

i — індекс нейрона у вхідному шарі;

n — розмірність вхідного шару (кількість нейронів на одну сторону);

(z, y, x) — тривимірні координати нейрона у вхідному шарі.

Масштабовані координати в вихідному шарі:

$$(c_z, c_y, c_x) = \left(\left\lfloor \frac{z * m}{n} \right\rfloor, \left\lfloor \frac{y * m}{n} \right\rfloor, \left\lfloor \frac{x * m}{n} \right\rfloor \right), \quad (2.28)$$

де:

(z, y, x) — тривимірні координати нейрона у вхідному шарі;

n — розмірність вхідного шару;

m — розмірність вихідного шару;

(c_z, c_y, c_x) — масштабовані координати у вихідному шарі.

Генерація зсувів у радіусі r

Формується множина всіх можливих зсувів у заданому радіусі:

$$\Delta = \{(dz, dy, dx) \mid -r \leq dz, dy, dx \leq r\}, \quad (2.29)$$

де:

Δ — множина всіх можливих зсувів у радіусі r ;

dz, dy, dx — зсуви по осях z, y, x ;

r — радіус, у межах якого генеруються зсуви.

Для кожного зсуву обчислюються нові координати:

$$(n_z, n_y, n_x) = (c_z + dz, c_y + dy, c_x + dx), \quad (2.30)$$

де:

(c_z, c_y, c_x) — координати центра околиці у вихідному шарі;

(dz, dy, dx) — зсуви по осях z, y, x ;

(n_z, n_y, n_x) — нові координати після застосування зсувів.

Перевіряється умова виходу за межі:

$$0 \leq n_z < m, \quad 0 \leq n_y < m, \quad 0 \leq n_x < m, \quad (2.31)$$

де:

(n_z, n_y, n_x) — нові координати після застосування зсувів;

m — розмірність вихідного шару.

Обчислюється індекс у вихідному шарі:

$$out_{idx} = n_z * m^2 + n_y * m + n_x, \quad (2.32)$$

де:

(n_z, n_y, n_x) — нові координати після застосування зсувів;

m — розмірність вихідного шару;

out_{idx} — індекс нейрона у вихідному шарі, обчислений на основі тривимірних координат.

Відбір нових зв'язків

Для виключення дублікатів порівнюються індекси нових з'єднань із вже існуючими, і формується маска для відбору унікальних зв'язків:

$$M_{new} = out_{idx} \notin existing_connections, \quad (2.33)$$

де:

M_{new} — маска, яка визначає нові зв'язки, що не містяться у множині існуючих зв'язків;

out_{idx} — індекс нового з'єднання у вихідному шарі;

$existing_connections$ — множина вже існуючих зв'язків.

Формується набір нових з'єднань:

$$C_{new} = \{out_{idx} \in M_{new}\}, \quad (2.34)$$

де:

C_{new} — набір нових з'єднань, що були відібрані за допомогою маски M_{new} ;

M_{new} — маска, яка вказує на унікальні нові зв'язки.

Ініціалізація ваг нових зв'язків

Ваги нових зв'язків ініціалізуються на основі середньої ваги:

$$w_{new} = \bar{w}_i, \quad (2.35)$$

де:

w_{new} — початкова вага для нового з'єднання;

\bar{w}_i — середнє абсолютне значення ваг існуючих зв'язків для нейрона i .

Переваги алгоритму

- структура мережі змінюється залежно від навчальних даних, забезпечуючи ефективну обробку складних завдань;
- створення нових зв'язків враховує топологію просторових координат, що зберігає локальні взаємозв'язки між елементами;
- метод підтримує різні масштабування між вхідним і вихідним шарами, дозволяючи застосовувати його в задачах з різними розмірами даних;
- завдяки локалізованим радіусам пошуку нових зв'язків алгоритм забезпечує обмежену кількість обчислень навіть для великих мереж.

Запропонований алгоритм створення нових зв'язків і ваг у тривимірних нейронних мережах забезпечує адаптивне збільшення обчислювальної ємності моделі. Його математичне обґрунтування базується на геометричній трансформації координат та аналізі існуючих зв'язків. Завдяки динамічному регулюванню порогових значень алгоритм демонструє високу ефективність у задачах класифікації складних тривимірних даних.

Подальші дослідження можуть зосередитися на вдосконаленні механізму ініціалізації ваг для зниження ризику перенавчання та покращення адаптивності алгоритму до змін у вхідних даних.

2.6 Теоретична модель класифікатора

Загальний опис моделі класифікатора

Архітектура класифікатора, що розробляється, побудована з урахуванням тривимірної структури обробки даних, яка ідеально підходить для роботи з 3D-зображеннями або воксельними форматами [51]. Теоретична модель цієї нейронної мережі (див. рис. 2.9), розділена на кілька логічних частин: вхідний шар, тривимірні шари, повнозв'язні шари і вихідний шар [72, 74].

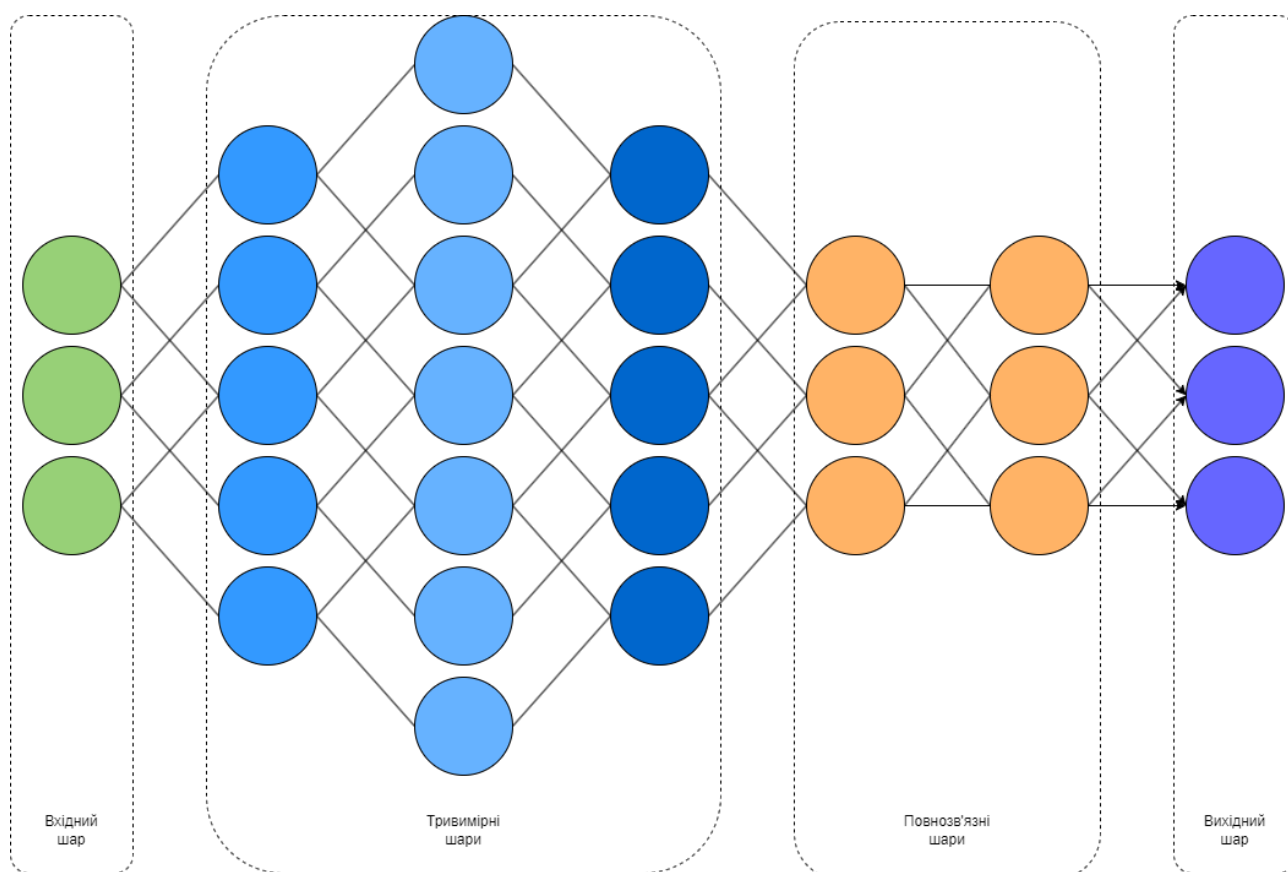


Рисунок 2.9 - Теоретична модель класифікатора.

Вхідний шар

Перший елемент архітектури — вхідний шар, представлений нейронами, позначеними зеленим кольором. Він відповідає за прийом вхідного вектора даних, який кодує тривимірну інформацію, наприклад, у вигляді вокселів або 3D-координат точок. Кожен нейрон цього шару відповідає окремому параметру або значенню вхідного зображення.

Основна функція: підготовка даних до подальшої обробки в прихованих шарах.

Тривимірні шари та їх структура

Ключовим компонентом моделі є приховані тривимірні шари, які складаються з двох основних типів шарів:

- шари збільшення розмірності;
- шари зменшення розмірності.

Шари збільшення розмірності (сині нейрони світлого кольору)

Шари збільшення розмірності виконують попередню обробку вхідних даних, знижуючи їхню роздільну здатність і підвищуючи ефективність обчислень. Це дозволяє моделі концентрувати увагу на основних особливостях зображення, ігноруючи незначні деталі.

Основна функція:

- зменшення щільності даних;
- зниження надмірності інформації для уникнення перенавчання;
- підготовка даних до виявлення ознак.

У цьому шарі нейрони аналізують локальні області вхідних даних, формуючи узагальнену інформацію для наступних етапів.

Шари зменшення розмірності (сині нейрони темного кольору)

Після етапу збільшення розмірності відбувається перехід до шарів зменшення розмірності, які виконують функцію витягування ознак зі спрощених даних.

Основна функція:

- формування нових представлень даних шляхом зменшення розмірності;
- виділення релевантних ознак для класифікації;
- створення векторів ознак із відповідними параметрами.

Шари зменшення розмірності забезпечують ефективне перетворення великого обсягу вхідних даних у компактні, але інформативні представлення.

Повнозв'язний шар

Після обробки даних прихованими шарами мережа переходить до повнозв'язного шару. На малюнку він позначений оранжевими нейронами.

Основна функція:

- виконання класифікації на основі векторів ознак, створених попередніми шарами;
- обробка всіх можливих зв'язків між ознаками для підвищення точності прогнозу.

Повнозв'язний шар моделює нелінійні залежності між виділеними ознаками та передає результати до вихідного шару.

Вихідний шар

Останній етап — вихідний шар, представлений фіолетовими нейронами. Кожен нейрон цього шару відповідає за один клас, який підтримує класифікатор.

Основна функція:

- обчислення ймовірностей приналежності вхідних даних до кожного з класів;
- подача результату у вигляді вектору ймовірностей.

Наприклад, якщо модель має 5 класів, то вихідний шар складатиметься з 5 нейронів, а сума вихідних значень буде дорівнювати 1 (за допомогою функції Softmax).

Оптимізація архітектури

Описана структура класифікатора оптимізується методами адаптивного видалення та додавання зв'язків між нейронами.

Методи оптимізації:

- слабкі зв'язки зі значеннями, що наближаються до нуля, видаляються, зменшуючи складність мережі та підвищуючи ефективність;
- якщо нейрон має сильні зв'язки з усіма доступними сусідами, він отримує можливість створювати нові зв'язки з віддаленішими нейронами (2-го або 3-го порядку).

Переваги оптимізації:

- зниження обчислювальних витрат;
- адаптивна зміна структури мережі відповідно до складності задачі;
- підвищення стійкості до перенавчання.

Взаємодія шарів та навчання

Навчання мережі відбувається за алгоритмом зворотного поширення помилки. Ваги оновлюються за допомогою оптимізаторів, таких як Adam.

Основні етапи навчання:

- пропуск вхідних даних через усі шари;
- обчислення функції втрат, наприклад, крос-ентропії;
- оновлення ваг у зворотному напрямку для мінімізації втрат.

Представлена модель класифікатора з тривимірною архітектурою поєднує ефективність обробки 3D-даних із методами оптимізації структури мережі. Завдяки використанню тривимірних та повнозв'язних шарів, вона демонструє високу гнучкість і адаптивність до вхідних даних. Оптимізація шляхом видалення слабких зв'язків та додавання нових дозволяє підтримувати баланс між продуктивністю та обчислювальною ефективністю, забезпечуючи високу точність класифікації.

Висновки до розділу 2

У другому розділі роботи було детально розглянуто методи класифікації тривимірних зображень на основі нейронних мереж, зокрема архітектури та оптимізаційні підходи, що дозволяють підвищити ефективність аналізу та обробки даних.

Проведений аналіз почався з етапів підготовки вхідних даних. Було показано, що нормалізація, фільтрація шумів і видалення аномалій є критично важливими для забезпечення високої якості даних перед їх подачею в нейронну

мережу. Ці методи дозволяють зменшити вплив шумів і артефактів, що виникають у процесі збору даних, та покращити стабільність результатів класифікації.

Окрема увага була приділена тривимірним форматам представлення даних, включаючи точкові хмари, вокселі та полігональні сітки. Було визначено переваги та обмеження кожного з них. Точкові хмари відзначаються високою деталізацією, проте мають слабо виражені топологічні зв'язки. Вокселі забезпечують внутрішнє моделювання структури об'єкта, але вимагають значних обчислювальних ресурсів. Полігональні сітки гарантують точне відтворення геометрії, однак потребують оптимізації для роботи з великими наборами даних.

Було описано застосування методів розгортки та згортки, що дозволяють зберігати важливі ознаки під час зміни розмірності вхідних даних. Ці підходи особливо важливі для тривимірних даних, оскільки забезпечують узагальнення особливостей об'єктів без втрати ключової інформації.

Окремо було досліджено алгоритми оптимізації нейронної мережі, включаючи методи видалення та додавання зв'язків між нейронами, а також елімінацію неактивних нейронів. Такий підхід дозволяє динамічно змінювати структуру моделі, забезпечуючи баланс між продуктивністю та точністю. Оптимізаційні алгоритми сприяють зменшенню обчислювальних витрат та адаптивності класифікатора до нових даних, що є важливим для масштабованих систем.

Було також розроблено теоретичну модель класифікатора з урахуванням архітектурних особливостей. Вона включає вхідний шар, тривимірні приховані шари для виявлення ознак, повнозв'язний шар для класифікації та вихідний шар для представлення результатів. Запропонована модель поєднує методи адаптивної оптимізації, забезпечуючи високу точність та ефективність обробки великих масивів тривимірних даних.

У підсумку, проведені дослідження підтвердили доцільність використання тривимірних нейронних мереж і методів оптимізації для класифікації тривимірних зображень. Це відкриває перспективи для подальшого вдосконалення архітектури та її програмної реалізації, що сприятиме підвищенню ефективності розв'язання прикладних задач у різних галузях, включаючи медицину, робототехніку та доповнену реальність.

3 АРХІТЕКТУРА ТА КОМПОНЕНТИ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ АВТОМАТИЗАЦІЇ КЛАСИФІКАЦІЇ ТРИВИМІРНИХ ЗОБРАЖЕНЬ

У сучасному світі обробка тривимірних зображень набуває дедалі більшої актуальності, особливо в медицині, промисловості та сфері тривимірного моделювання. Завдяки розвитку фреймворків машинного навчання розробники отримали потужні інструменти, які дають змогу вирішувати складні завдання класифікації, сегментації та аналізу 3D-об'єктів. У цій частині дослідження пропонується розглянути архітектуру програмних засобів та їх компонентів, що поєднує ASP.NET WebAPI (для організації вебсервісу), ML.NET (для класичних методів машинного навчання) та Keras.NET (для глибоких нейронних мереж). Такий інтегрований підхід надає можливість масштабувати рішення та впроваджувати його в середовищах з різним рівнем навантаження.

3.1 Загальна структура програмних засобів

Загальна діаграма побудови програмних засобів (див. рис. 3.1 або Д.1) ґрунтується на багатошаровій архітектурі, де кожен шар має чітко визначену роль у загальному конвеєрі обробки даних. Цей підхід сприяє підтримуванню, розширенню та легкій адаптації до нових вимог. На діаграмі, наведені окремі модулі, що розміщені в межах декількох логічних шарів: шар представлення, шар бізнес-логіки, шар машинного навчання, шар даних та інфраструктурний шар. Кожен із них робить свій внесок у загальний процес, починаючи від приймання й розбору запиту, до збереження та аналізу результатів класифікації.

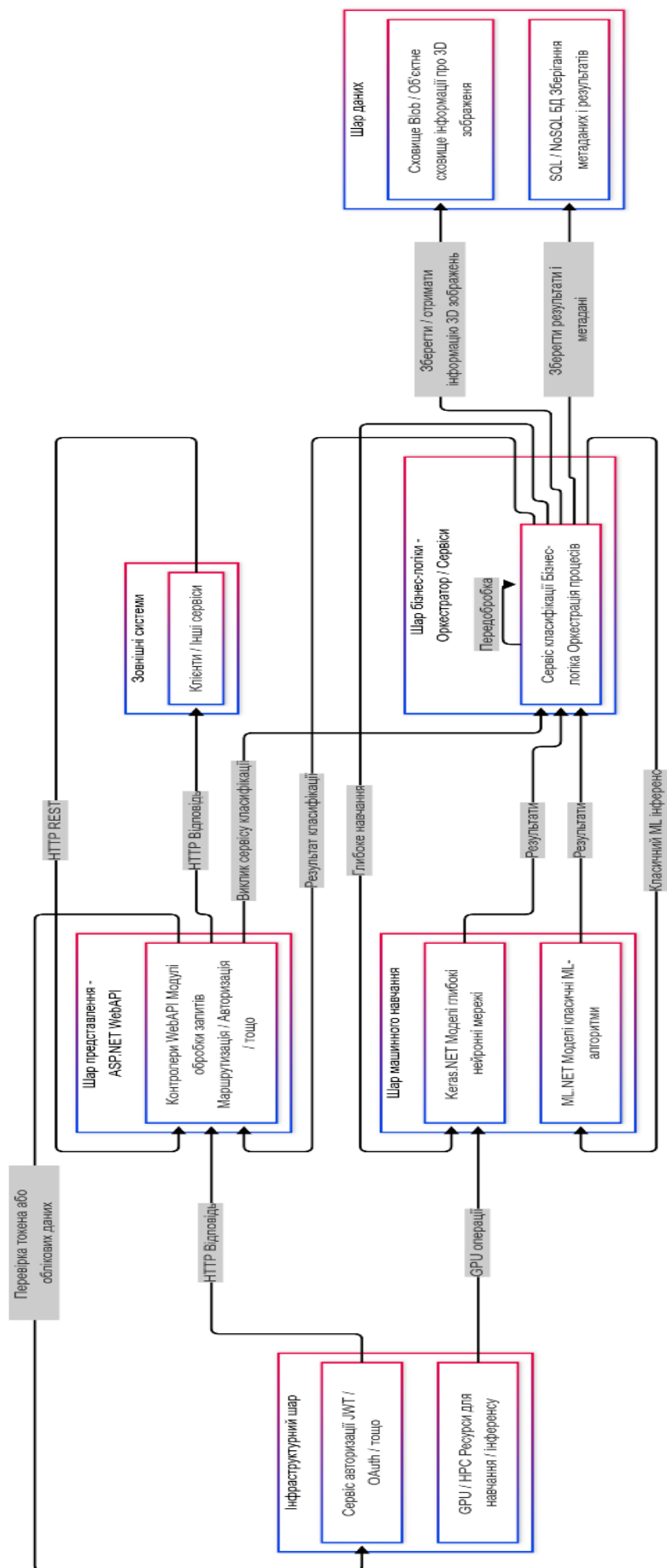


Рисунок 3.1 - Діаграма побудови програмних засобів.

Шар представлення

У даній системі за шар представлення відповідають контролери ASP.NET WebAPI. Вони обробляють вхідні HTTP-запити, розподіляють їх відповідно до маршрутів і виконують початкові операції з авторизації чи валідації даних. У разі успішної авторизації (яка реалізована через токени JWT) контролери викликають сервіси бізнес-логіки. Важливою особливістю цього шару є його універсальність: розробники та зовнішні системи можуть використовувати єдиний REST-інтерфейс для взаємодії, що полегшує інтеграцію програмних засобів в інші проекти чи мікросервісну інфраструктуру.

Шар бізнес-логіки

Серце системи становить шар бізнес-логіки, який реалізує основну оркестрацію процесу класифікації. Тут розміщено сервіс класифікації, що відповідає за керування послідовністю дій: від початкової передобробки тривимірних даних до збереження результатів. Якщо потрібно нормалізувати, згладжувати чи змінювати формат вихідного 3D-файлу, ці етапи відбуваються саме в межах бізнес-логіки. Далі сервіс визначає, який саме інструментарій машинного навчання варто залучити, залежно від вимог до точності, доступних ресурсів та специфіки вхідних даних. Саме цей шар виконує також додаткові задачі перевірки та контролю коректності, зберігаючи послідовність і цілісність усієї процедури.

Шар машинного навчання

Шар машинного навчання представлений двома основними компонентами: ML.NET Models та Keras.NET Models. Перший компонент зосереджений на реалізації класичних алгоритмів (лінійні методи, дерева рішень, прискорення), що можуть бути ефективними для початкових або менш складних завдань. Другий компонент залучає потужні глибинні нейронні мережі, здатні працювати з великою кількістю параметрів і виявляти складні закономірності в 3D-даних. Використання Keras.NET дає змогу підключати мережі, попередньо

навчені в середовищах Python/TensorFlow, або навіть розгортати власне навчання безпосередньо в межах .NET-екосистеми.

Важливо зауважити, що в разі опрацювання великих обсягів тривимірних даних або складних мультимедійних колекцій, часто виникає потреба в обчислювальних ресурсах на рівні GPU/НРС. Це виправдано, коли час відповіді має бути мінімальним (наприклад, у реальних медичних системах) або коли обчислення настільки масштабні, що традиційні CPU-ресурси не забезпечують належної продуктивності.

Шар даних

Безпосереднє збереження та керування даними здійснюється у шарі даних, який розділено на дві частини. Перша – це SQL або NoSQL база даних для управління метаданими, історіями класифікацій, логами та іншою структурованою інформацією, що дозволяє гнучко виконувати пошук чи агрегації. Друга частина – сховище Blob (або інший тип об'єктного сховища), призначене для зберігання інформації про тривимірні зображення. Це можуть бути будь-які великі ресурси, яким потрібна спеціалізована інфраструктура для швидкого читання, запису та захисту від втрати даних. Відокремлення «важких» об'єктів від метаданих і результатів аналітики забезпечує більшу масштабованість та підвищує ефективність системи в цілому.

Інфраструктурний шар

Для комплексної реалізації подібних засобів, важливим є також інфраструктурний шар. Він зосереджує питання автентифікації та авторизації користувачів (через сервіси Auth Service, IdentityServer4) та керування обчислювальними ресурсами, включно з GPU або цілими НРС-кластерами. В межах великої корпорації або дослідницької установи це може бути окремий Kubernetes-кластер, що розгортає обчислювальні ноди з підтримкою CUDA.

Наявність такого шару на діаграмі дає змогу підкреслити критично важливу роль безпеки та керованості в створенні продуктивної і надійної системи.

Процес взаємодії

Процес взаємодії між шарами починається з того, що зовнішня система або користувач надсилає запит до ASP.NET WebAPI. Після успішної перевірки авторизації шар представлення переспрямовує управління у бізнес-логіку. Сервіс класифікації виконує внутрішню попередню обробку, визначає тип моделі (класичної або глибокої) та передає дані до відповідного компонента шарів ML.NET Models чи Keras.NET Models. Після завершення обчислень (інференсу) отриманий результат або зберігається в базі даних (якщо вимагають умови використання), або відразу повертається у вигляді HTTP-відповіді. За потреби, попередньо або паралельно звертаються до blob сховища для отримання початкової або проміжної інформації про 3D зображення.

Уся послідовність дій відбувається таким чином, щоб у кінцевому підсумку клієнт отримав максимально точний результат класифікації, використовуючи при цьому всі можливості сучасної інфраструктури та фреймворків машинного навчання. При збільшенні навантаження система може бути масштабована горизонтально, додаючи додаткові сервери WebAPI чи розміщуючи додаткові ML-сервіси в окремих контейнерах. Цей підхід гарантує стабільність і дозволяє обробляти великий потік 3D-зображень у режимі реального часу.

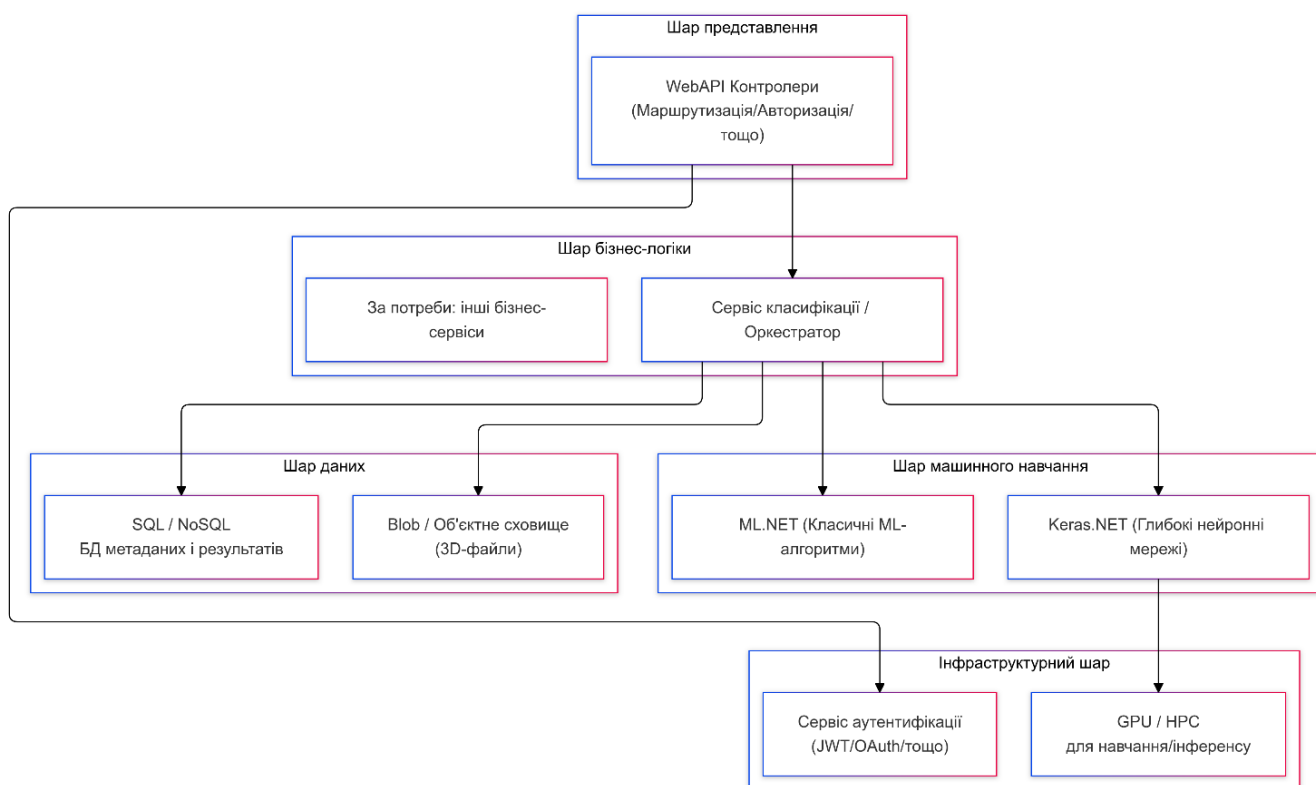


Рисунок 3.2 - Діаграма компонентів програмних засобів.

У межах наведеної діаграми компонентів (див. рис. 3.2) можна виокремити п'ять основних шарів, які утворюють загальну структуру програмних засобів: шар представлення, шар бізнес-логіки, шар машинного навчання, шар даних та інфраструктурний шар. Кожен із цих сегментів виконує цілком визначену функцію та відповідає за окрему ланку конвеєра обробки запитів і даних. Такий розподіл дає змогу забезпечити високу гнучкість, масштабованість і підтримку засобів, а також дає можливість вносити зміни в конкретний шар без значного втручання в роботу інших компонентів.

На діаграмі (див. рис. 3.2) чітко видно, що початковим пунктом обробки запитів виступає шар представлення, позначений блоком WebAPI. Цей компонент відповідає за маршрутизацію вхідних HTTP-запитів, їхню авторизацію (якщо така передбачена в рамках загальної системи) та передачу необхідних даних іншим підсистемам. WebAPI може бути реалізований на базі ASP.NET Core, що дає широкий вибір засобів налаштування контролерів, фільтрів, middleware та механізмів безпеки. З діаграми видно, що WebAPI має

прямий зв'язок із сервісом аутентифікації (AuthService), аби перевірити повноваження користувачів або зовнішніх сервісів. Крім того, WebAPI надсилає запит на оркестратор (Orchestrator), який належить до шару бізнес-логіки.

Шар бізнес-логіки, окреслений блоком Orchestrator і додатковими бізнес-сервісами (OtherServices), є центральною ланкою системи, де зосереджена основна функціональна логіка програмних засобів. Саме тут приймаються рішення щодо того, які саме операції слід виконати над вхідними даними, у якому порядку відбуватимуться виклики компонентів машинного навчання та як будуть організовані процеси збереження результатів. Зв'язок Orchestrator із шаром машинного навчання (MachineLearning) свідчить, що для класифікації та обробки тривимірних зображень можуть використовуватися як класичні ML-алгоритми (через ML.NET), так і глибокі нейронні мережі (через Keras.NET). Проте роль Orchestrator не обмежується маршрутизацією даних: він також може виконувати загальну координацію, керування транзакціями чи логування з метою відстеження проміжних і фінальних результатів.

Шар машинного навчання включає два ключові блоки: ML.NET і Keras.NET. На діаграмі помітно, що вони безпосередньо взаємодіють із шаром бізнес-логіки, отримуючи відповідні дані для обчислень. ML.NET забезпечує засоби класичного машинного навчання, де можуть бути використані моделі регресії, дерев рішень, метод опорних векторів чи інші підходи. Ці інструменти, як правило, зручні, коли обсяг даних або їхня складність не потребує глибоких нейронних мереж. У той самий час Keras.NET дає змогу підключати потужні згорткові мережі та інші архітектури глибинного навчання. Завдяки цьому користувач може використати інфраструктурні переваги GPU або HPC-ресурсів, адже з діаграми видно зв'язок Keras.NET з блоком GPU. Така взаємодія означає, що саме у компоненті глибинних нейронних мереж виконується складне паралельне обчислення, необхідне для високоточних моделей класифікації чи сегментації 3D-зображень.

Наступним критично важливим складником загальної структури є шар даних (Data), представлений блоками DB та BlobStorage. DB орієнтований на зберігання структурованих даних, зокрема метаданих і результатів обчислень, які можна оперативно вибирати, аналізувати та поєднувати з іншими записами в рамках системи. Така база може бути реляційною (SQL) або нереляційною (NoSQL), залежно від конкретних вимог до масштабованості, структури даних та продуктивності. BlobStorage забезпечує збереження великих об'єктів, а саме тривимірних файлів, які можуть мати значний розмір і вимагати спеціалізованого підходу до зберігання. Завдяки цьому логіка програмних засобів розділена таким чином, що метадані й аналітичні записи не переплітаються з «важкими» мультимедійними чи науковими даними. Оркестратор має прямий зв'язок із цими двома блоками: він може завантажувати або зчитувати 3D-файли з BlobStorage, а також записувати або читати інформацію про класифікацію чи результати навчання з DB.

Завершує загальну діаграму інфраструктурний шар, що охоплює компоненти AuthService та GPU. AuthService відіграє центральну роль у забезпеченні безпеки та дотриманні політик доступу. Він може реалізовувати аутентифікацію за допомогою JWT, OAuth 2.0 чи інших протоколів, аби гарантувати, що лише санкціоновані користувачі або сервіси мають доступ до операцій класифікації. GPU або HPC-ресурси необхідні для оперативного виконання трудомістких обчислень, зокрема при застосуванні глибинних нейронних мереж. На діаграмі помітно, що цей блок напряму пов'язаний із Keras.NET, оскільки саме бібліотека глибинного навчання найчастіше використовує паралельні операції матричного множення і конволюцій, які можуть бути прискорені за допомогою графічних процесорів.

Таким чином, наведена діаграма компонентів відображає інтегровану структуру, де шар представлення передає вхідні дані в шар бізнес-логіки, а той, своєю чергою, залучає інструменти машинного навчання і спирається на відповідні сховища даних. Інфраструктурний шар слугує центральним вузлом

безпеки та високопродуктивних обчислень, що має критичне значення для успішної реалізації функціоналу класифікації тривимірних зображень у реальному чи дослідницькому середовищі. Такий багатoshаровий підхід гарантує чіткий поділ обов'язків, надійність у масштабуванні й можливість підтримки сучасних фреймворків ML.NET та Keras.NET у рамках розроблюваних програмних засобів.

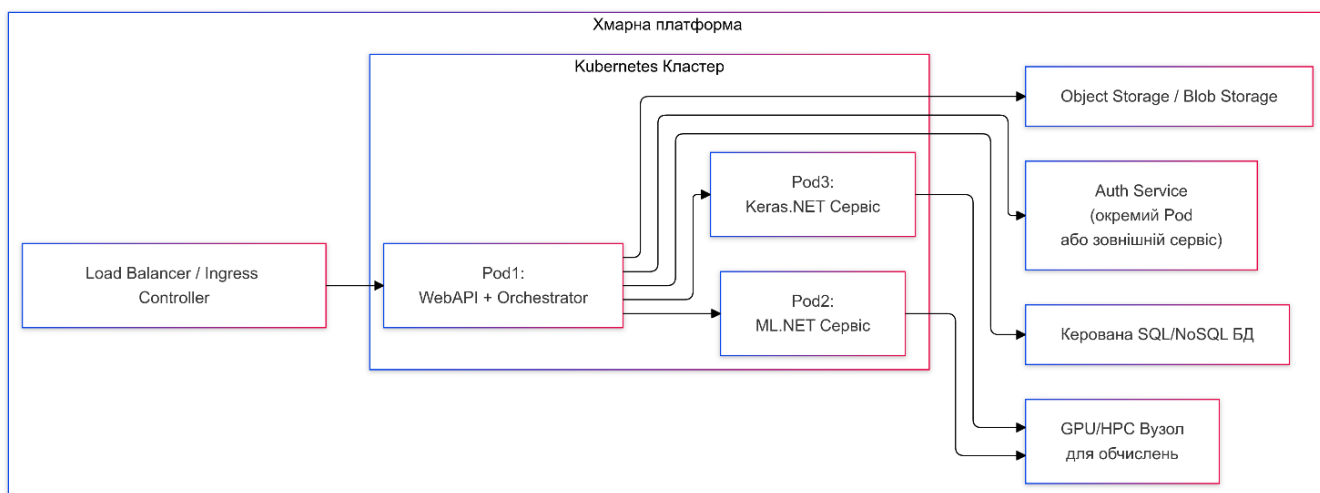


Рисунок 3.3 - Діаграма розгортання.

На запропонованій діаграмі (див. рис. 3.3) зображено типовий сценарій розгортання програмних засобів для класифікації тривимірних зображень у хмарній платформі з використанням Kubernetes. У межах Kubernetes-кластера створено декілька Pod-ів, кожен із яких виконує окреме призначення. Pod1 містить WebAPI та Orchestrator, що відповідають за приймання запитів від користувачів (через балансувальник навантаження) та керування процесом класифікації. Pod2 реалізує ML.NET Service, призначений для виконання класичних алгоритмів машинного навчання та конвеєрів обробки, а Pod3 – Keras.NET Service, що здатен обробляти глибокі нейронні мережі, включно з 3D-згортками.

Важливою складовою є Load Balancer / Ingress Controller, через який спрямовуються вхідні HTTP-запити. Завдяки цьому рішення отримує гнучкість у маршрутизації та масштабуванні, адже можна динамічно створювати нові Pod-и й розподіляти навантаження між ними. AuthService, виділений як окремий Pod

або зовнішній сервіс, забезпечує автентифікацію й авторизацію, даючи змогу контролювати доступ до API на рівні кластера.

Для зберігання та обробки даних діаграма передбачає дві ключові компоненти: керовану SQL/NoSQL БД (DB) для метаданих і результатів класифікації та Object Storage / Blob Storage (Blob) для великих 3D-файлів. Pod1 (WebAPI + Orchestrator) взаємодіє з ними, керуючи завантаженням, оновленням чи читанням необхідних даних. У ситуаціях, коли потрібна значна обчислювальна потужність, Pod2 і Pod3 можуть скористатися GPU/HPC-вузлом (GPUNode), що підвищує швидкість тренування та інференсу глибоких моделей.

Загалом така архітектура забезпечує високу масштабованість, гнучкість конфігурації та розділення відповідальності. Контейнери з WebAPI й оркестратором можуть бути масштабовані залежно від кількості вхідних запитів, тоді як ML.NET Service і Keras.NET Service з GPU-ресурсами масштабуються за потреби в інтенсивних обчисленнях. Завдяки цим програмним засобам зберігає ефективність при обробці великої кількості 3D-зображень та легко інтегрується з іншими сервісами в хмарному середовищі.

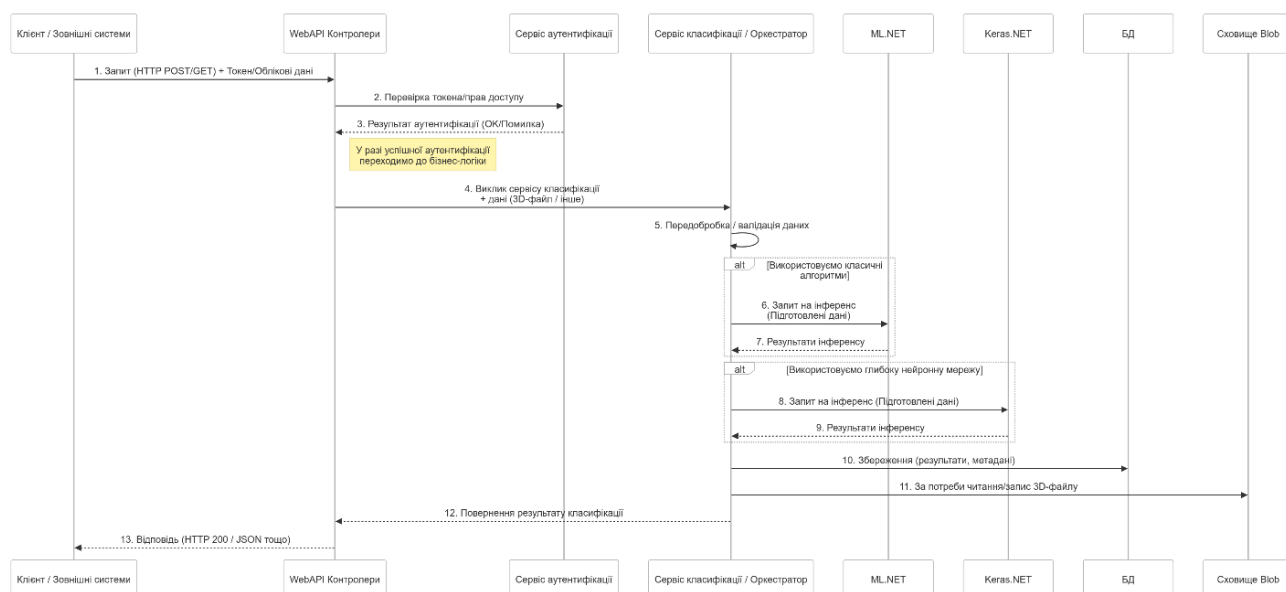


Рисунок 3.4 - Діаграма станів системи.

Послідовність взаємодії між компонентами системи, відображена на діаграмі (див. рис. 3.4 або Д.5), демонструє типовий сценарій роботи програмних засобів для класифікації тривимірних зображень у багаторівневому середовищі. На початку Клієнт (або зовнішня система) надсилає HTTP-запит із токеном чи реєстраційними даними до WebAPI Контролерів (крок 1). WebAPI виконує перевірку повноважень, звертаючись до Сервісу аутентифікації (крок 2). Якщо сервіс авторизації повертає позитивний результат (крок 3), надходить дозвіл на подальшу обробку даних. У протилежному випадку запит завершується повідомленням про помилку доступу.

Після підтвердження авторизації WebAPI передає керування на Сервіс класифікації / Оркестратор (крок 4). Оркестратор виконує внутрішню передобробку та валідацію (крок 5), перевіряючи цілісність отриманих 3D-даних. Далі приймається рішення щодо того, яку технологію машинного навчання застосувати. Якщо достатньо класичних алгоритмів, відбувається виклик ML.NET (крок 6), який після обчислень повертає попередній або фінальний прогноз (крок 7). Якщо ж необхідно розгорнути глибинні нейронні мережі, Orchestrator передає дані до Keras.NET (крок 8). Після виконання інференсу модель повертає результат класифікації (крок 9), який Orchestrator може додатково проаналізувати чи зіставити з іншими метаданими.

На завершальному етапі результат (передбачення або оцінка ймовірностей) зберігається в БД (крок 10), а за потреби також може здійснюватися читання чи запис відповідних 3D-файлів у Сховище Blob (крок 11). Оркестратор повертає зведену відповідь до WebAPI (крок 12). Зі свого боку, WebAPI формує фінальний HTTP-відповідь (крок 13) з кодом успішного виконання (200) або іншим відповідним статусом, а також структурованими даними (JSON/XML). Таким чином, клієнт отримує уніфікований доступ до процесу класифікації тривимірних зображень із можливістю масштабування, безпечного зберігання даних і гнучкого вибору методів машинного навчання. Подібна послідовність дій

забезпечує цілісність і надійність системи, а також дає змогу легко інтегрувати її з іншими корпоративними або дослідницькими платформами.

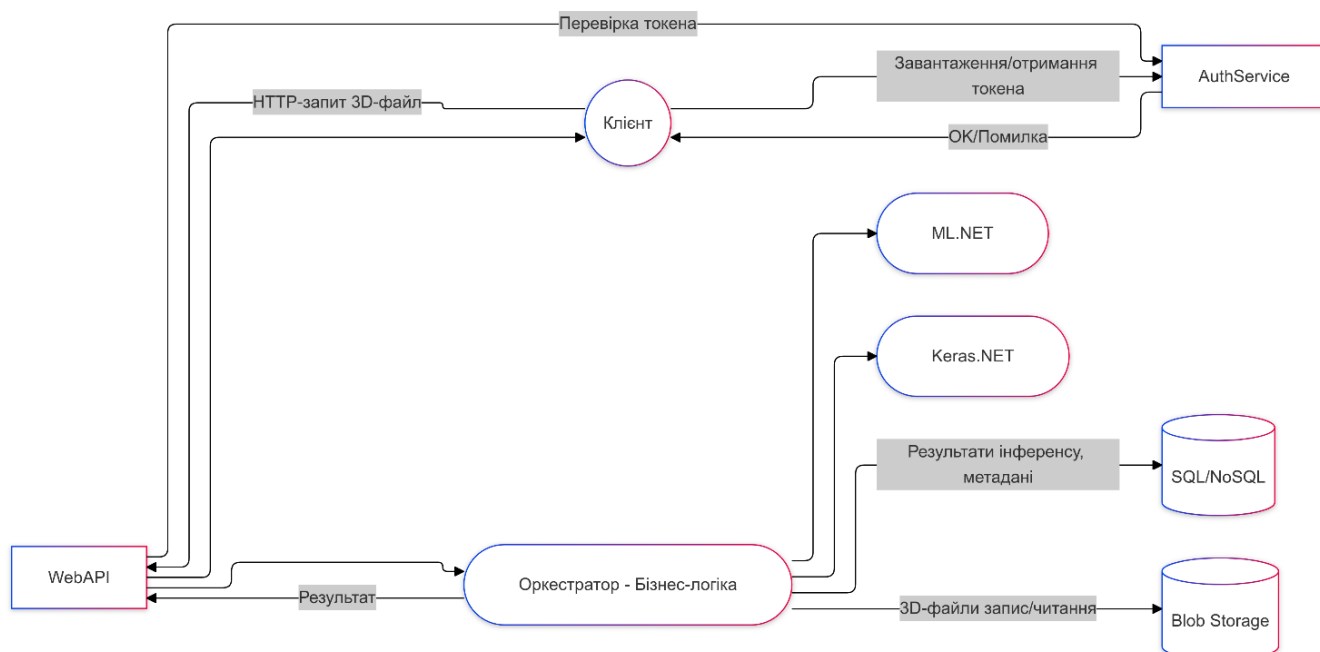


Рисунок 3.5 - Діаграма діяльності.

Наведена діаграма (див. рис. 3.5) ілюструє повний цикл обробки та класифікації тривимірних зображень у межах запропонованих програмних засобів. Процес розпочинається з того, що Клієнт звертається до AuthService задля отримання або верифікації токена доступу. У разі успішної авторизації клієнт може надіслати HTTP-запит, що містить 3D-файл (чи посилання на нього), до WebAPI. Зі свого боку, WebAPI перевіряє отриманий токен, знову звертаючись до AuthService. Якщо перевірка виявляється успішною, запит передається в Оркестратор (Business-Logic Layer), який відповідає за координацію та логіку подальшого процесу класифікації.

Усередині Оркестратора визначається, які саме модулі машинного навчання варто залучити: ML.NET або Keras.NET. ML.NET переважно використовується для класичних алгоритмів (лінійні методи, дерева рішень, регресія), тоді як Keras.NET забезпечує доступ до глибоких нейронних мереж, що ефективні для складних 3D-задач із великими масивами даних. Під час інференсу Оркестратор може збирати чи зберігати проміжні та підсумкові результати в сховищах даних.

Для структурованих метаданих (результати обчислень, історії класифікацій, записи логів) використовується SQL/NoSQL база даних (DS1), а великі 3D-файли зберігаються у Blob Storage (DS2). Оркестратор звертається до цих сховищ у процесі читання чи запису, зокрема якщо потрібно зберегти результати інференсу або отримати додаткові 3D-файли з минулих сесій.

Після завершення класифікації Оркестратор повертає підсумковий результат запиту назад до WebAPI. WebAPI формує уніфіковану HTTP-відповідь, яку надсилає клієнту. Завдяки цьому клієнт отримує можливість у реальному часі ініціювати процес класифікації, передавати 3D-зображення, а також оперативно отримувати прогнози та супутню інформацію (наприклад, імовірності, метадані чи часові мітки). Така архітектура забезпечує чіткий поділ відповідальності між компонентами, захищеність доступу (через AuthService), гнучке використання різних інструментів машинного навчання та надійну роботу з великими обсягами даних у хмарних або локальних середовищах.

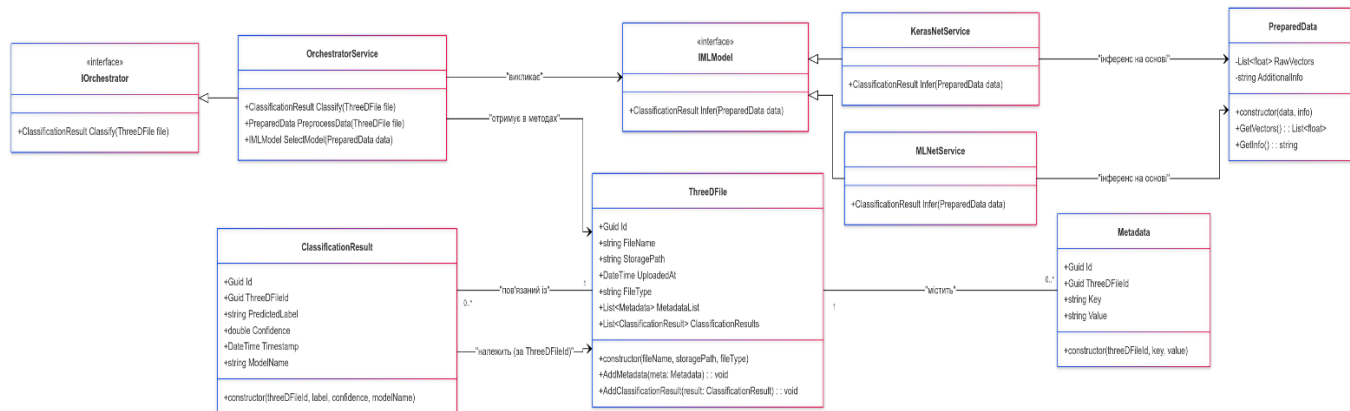


Рисунок 3.6 - Діаграма класів програмних засобів.

У наведеній діаграмі класів (див. рис. 3.6) показано логічну структуру програмних компонентів, що взаємодіють у процесі обробки та класифікації тривимірних зображень. Вона ілюструє, як об'єкти різних типів пов'язані між собою, які інтерфейси вони реалізують і яким чином відбувається обмін даними на об'єктному рівні. Такий підхід дає змогу забезпечити чітке розмежування відповідальності між бізнес-логікою (OrchestratorService), моделями машинного

навчання (MLNetService, KerasNetService), а також структурами даних (ThreeDFile, PreparedData, Metadata, ClassificationResult).

Інтерфейс IOrchestrator визначає контракт для оркестратора та містить метод Classify. Від нього успадковується клас OrchestratorService, який реалізує бізнес-логіку програмних засобів. У межах цього сервісу передбачено методи Classify, PreprocessData і SelectModel. Завдяки такому розподілу обов'язків OrchestratorService може спочатку підготувати дані (наприклад, виконати нормалізацію чи вилучення ознак), а потім, використовуючи внутрішній метод SelectModel, визначати, який саме алгоритм чи фреймворк машинного навчання застосувати. Це надає гнучкість вибору між класичними моделями ML.NET або глибинними мережами, що підключаються через Keras.NET.

Інтерфейс IMLModel задає метод Infer, який має реалізовувати будь-яка модель машинного навчання. Класи MLNetService та KerasNetService успадковують IMLModel, пропонуючи власні варіанти реалізації методу Infer. MLNetService застосовує класичні підходи (лінійні методи, дерева рішень, прискорення тощо) і зазвичай є ефективним для завдань із відносно невеликою кількістю параметрів або менш складними даними. Водночас KerasNetService реалізує глибинні нейронні мережі (зокрема згорткові 3D-архітектури) і може використовувати GPU-акселерацію для пришвидшеного обчислення. Завдяки інтерфейсу IMLModel OrchestratorService викликає узгоджений метод Infer, не переймаючись конкретною реалізацією, що полегшує підтримку та розширюваність системи.

Клас PreparedData уособлює результат попередньої обробки вхідного файлу. Він містить поле RawVectors для зберігання числових векторів (наприклад, вокселів або характеристичних ознак), а також додаткову інформацію в полі AdditionalInfo. Набір відповідних методів (GetVectors, GetInfo) полегшує доступ до внутрішніх структур даних. Подібний підхід дозволяє чітко відокремити сирі 3D-файли від їхніх векторизованих чи агрегованих представлень.

Клас `ThreeDFile` містить метадані про файл, такі як його унікальний ідентифікатор (`Id`), назва (`FileName`), шлях до сховища (`StoragePath`), дата завантаження (`UploadedAt`) і тип файлу (`FileType`). Поля `MetadataList` та `ClassificationResults` забезпечують логічний зв'язок із колекціями об'єктів `Metadata` та `ClassificationResult`. Така структура спрощує відстеження всіх операцій, пов'язаних з обробкою файлу, включно з інформацією про застосовані методи класифікації, історію змін та вхідні дані (ключі, описові значення тощо).

Клас `Metadata` призначений для зберігання довільних ключ/значення пар (`Key`, `Value`), які можна додавати до `ThreeDFile` з метою збагачення опису або проведення додаткових аналітичних операцій. Клас `ClassificationResult` містить підсумки інференсу, включно з передбаченою міткою (`PredictedLabel`), впевненістю передбачення (`Confidence`), назвою моделі (`ModelName`) та часовою міткою (`Timestamp`). Це дає змогу системі фіксувати, коли саме було виконано класифікацію, які результати отримано й за допомогою якого алгоритму.

Таким чином, діаграма чітко демонструє, як окремі класи та інтерфейси взаємодіють між собою в процесі класифікації 3D-файлів. Вона забезпечує модульний, розширюваний дизайн: `OrchestratorService` виступає координаційним центром, взаємодіючи з `MLNetService` або `KerasNetService` через уніфікований інтерфейс `IMLModel`, тоді як `PreparedData`, `ThreeDFile`, `Metadata` і `ClassificationResult` структурують дані для ефективного зберігання та аналізу в межах системи.

3.2 Використані бібліотеки та інструменти

У підрозділі 1.1.2 ми здійснили детальний аналіз теоретичних можливостей і функціональних особливостей різних бібліотек для реалізації нейронних мереж. У даному розділі буде проведено комплексне дослідження практичних аспектів використання зазначених бібліотек. Зокрема, розглядатимуться приклади реалізації архітектур та алгоритмічних топологій нейронних мереж, орієнтованих на задачі глибокого навчання. Також буде висвітлено

методологічні підходи до створення індивідуалізованих архітектурних конфігурацій і оптимізаційних стратегій для підвищення продуктивності моделей. Особливу увагу приділено адаптації алгоритмів під специфічні вимоги задач класифікації та обробки тривимірних даних, що забезпечить всебічне розуміння процесів побудови та налаштування нейронних мереж.

3.2.1 ML.NET

ML.NET є фреймворком машинного навчання від компанії Microsoft, орієнтованим на розробників у середовищі .NET. Основна ідея полягає у спрощенні процесу побудови, тренування та розміщення моделей машинного навчання, не покидаючи межі екосистеми .NET. За допомогою ML.NET можна виконувати широкий спектр завдань: класифікацію, регресію, рекомендаційні системи, аналіз тексту, виявлення аномалій та навіть роботу з часовими рядами.

Однією з ключових концепцій ML.NET є DataView – внутрішній формат зберігання та обробки даних, що дозволяє ефективно працювати з великими наборами даних у потоковому режимі. Це означає, що бібліотека може поетапно читати частини даних для обчислень, не зберігаючи весь набір даних у пам'яті. Така архітектура дає змогу масштабувати обчислення та полегшує інтеграцію з різними джерелами, як-от файли CSV, SQL-бази чи будь-які інші структуровані формати.

У ML.NET передбачено конвеєри (pipelines), які складаються з послідовності перетворень (transformations) та тренувань моделі. Для задачі класифікації можна вибудувати конвеєр, що включає кілька кроків: спочатку виконується нормалізація або кодування ознак, далі – можливі «feature engineering» кроки (наприклад, текстова обробка чи вилучення певних статистик), а потім – виклик навчального алгоритму (наприклад, LightGBM, кросплатформні реалізації лінійних моделей чи методи на основі дерев рішень). Розробник має гнучкі можливості налаштування кожного кроку. Окрім того, бібліотека пропонує вбудовані методи для розрахунку метрик (Accuracy, F1-score, RMSE, Precision, Recall), що спрощує аналіз результатів.

У контексті тривимірних зображень ML.NET може використовуватися для організації класичних сценаріїв обробки. Наприклад, коли розміри чи особливості 3D-даних дають змогу їх «згортати» у набір статистичних ознак або коли застосовується підхід до вилучення ознак за допомогою інших інструментів, а сам конвеєр машинного навчання зосереджується на прийнятті рішення. Цей фреймворк також пропонує механізми для автоматичного машинного навчання (AutoML), котрі можуть спростити вибір оптимальної моделі та налаштувань гіперпараметрів.

Завдяки сумісності ML.NET із .NET Core, програмні засоби можуть бути розгорнуті на різноманітних платформах та операційних системах, включно з Windows, Linux чи macOS. Така кросплатформеність важлива, коли потрібно забезпечити універсальний доступ або інтегрувати систему з іншими сервісами, що працюють у хмарі чи у контейнерах Docker/Kubernetes.

3.2.2 Keras.NET

Keras.NET – це бібліотека-обгортка, що надає змогу .NET-розробникам взаємодіяти з фреймворком Keras, популярним у середовищі Python. Keras сам по собі є високорівневим API для глибинного навчання, що використовується поверх таких обчислювальних середовищ, як TensorFlow чи Theano. У контексті тривимірних зображень Keras.NET виявляється незамінним, оскільки дозволяє створювати, налаштовувати та тренувати складні згорткові нейронні мережі (CNN), зокрема 3D-варіації, призначені для роботи з об’ємними даними.

Ключова перевага Keras.NET полягає в можливості створення користувацьких шарів і параметрів без необхідності залишати простір .NET. Наприклад, розробник може налаштувати мережу, додаючи специфічні шари (Conv3D, MaxPooling3D, LSTM, Dense), змінюючи функції активації (ReLU, Tanh, LeakyReLU тощо) або навіть створюючи власні кастомні шари на основі існуючих. У глибинних нейронних мережах, що мають велику кількість параметрів, часто виникає потреба в більш тонкій оптимізації, яка включає зміну

кількості ядер згортки (filters), розмірів «вікна» (kernel_size) чи методів ініціалізації ваг (HeNormal, XavierUniform тощо).

Розглянемо як можна розширити наявні класи бібліотеки.

```

1 // -----
2 // 1. Приклад кастомного шару (Layer)
3 // -----
4 public class CustomLayer : Layer
5 {
6     private int _units;
7     private string _activation;
8
9     public CustomLayer(int units, string activation = "relu", string name = "CustomLayer")
10        : base(name: name, trainable: true)
11     {
12         _units = units;
13         _activation = activation;
14     }
15
16     protected override void build(Shape input_shape)
17     {
18         // При бажанні можна ініціалізувати власні ваги, наприклад:
19         // this.AddWeight(shape: new Shape(input_shape[1], _units),
20         //                 initializer: "glorot_uniform",
21         //                 name: this.Name + "_weights");
22         base.build(input_shape);
23     }
24
25     protected override Keras.Tensor call(Keras.Tensor inputs, bool is_training = false)
26     {
27         // У цьому прикладі продемонструємо найпростішу ідею:
28         // Помножимо вхідні дані на деякий коефіцієнт, а потім застосуємо активацію.
29         // За потреби можна отримати доступ до попередньо доданих ваг:
30         // var W = this.GetWeights()[0];
31         // var z = K.dot(inputs, W);
32
33         // Для простоти зробимо псевдооперацію: y = inputs * 2
34         var scaledInput = Keras.Backend.K.Mul(inputs, 2.0);
35
36         // Потім застосуємо вбудовану активацію (наприклад ReLU)
37         Keras.Tensor activated;
38         switch (_activation.ToLower())
39         {
40             case "relu":
41                 activated = Keras.Backend.K.Relu(scaledInput);
42                 break;
43             case "sigmoid":
44                 activated = Keras.Backend.K.Sigmoid(scaledInput);
45                 break;
46             // Можна додати інші варіанти.
47             default:
48                 activated = scaledInput;
49                 break;
50         }
51
52         return activated;
53     }
54 }

```

Рисунок 3.7 - Приклад коду оригінального шару.

Даний шар (див. рис. 3.7) наслідує базовий клас Layer з Keras.NET і перевизначає методи build та call. У методі build можна (за потреби) оголосити та ініціалізувати власні параметри (ваги, зміщення), які будуть навчатись під час зворотного розповсюдження помилки. У методі call виконується основна логіка перетворення вхідних тензорів. У прикладі для наочності показано, як масштабувати значення та застосовувати активаційну функцію “ReLU” з бібліотеки TensorFlow. У реальному проєкті ви можете реалізувати більш складну математику, наприклад, власні згортки чи спеціалізовані функції.

```

1 // -----
2 // 2. Приклад кастомного колбеку (Callback)
3 // -----
4 public class CustomCallback : Callback
5 {
6     // Викликається на початку епохи
7     public override void OnEpochBegin(int epoch, Dictionary<string, NArray> logs = null)
8     {
9         Console.WriteLine($"[CustomCallback] Початок епохи {epoch}");
10        base.OnEpochBegin(epoch, logs);
11    }
12
13    // Викликається в кінці кожної епохи
14    public override void OnEpochEnd(int epoch, Dictionary<string, NArray> logs = null)
15    {
16        if (logs != null && logs.ContainsKey("loss"))
17        {
18            var lossValue = logs["loss"];
19            Console.WriteLine($"[CustomCallback] Кінець епохи {epoch}, loss: {lossValue}");
20        }
21        base.OnEpochEnd(epoch, logs);
22    }
23 }

```

Рисунок 3.8 - Приклад коду оригінального зворотного виклику.

Однією з особливостей Keras.NET є можливість підключення оригінальних зворотних викликів (див. рис. 3.8), наприклад, для автоматичного регулювання швидкості навчання або збереження кращих ваг моделі. У сценаріях, де важливо отримувати проміжні результати (наприклад, у медичних дослідженнях для сегментації зображення), зворотні виклики спрощують моніторинг показників якості в процесі навчання. Також можна реалізувати власні функції втрат чи метрики, що дає повну гнучкість у налаштуванні нейронних мереж під специфічні задачі.

```

1 // -----
2 // 3. Приклад кастомної 3D-моделі з використанням нашого шару
3 // -----
4 public class Custom3DModel
5 {
6     public static void BuildAndTrainModel()
7     {
8         var model = new Sequential();
9
10        // Припустимо, ми маємо 3D-вхідні дані розміром 64 x 64 x 64 з одним каналом
11        // (наприклад, сірий колір або однокомпонентні воксели).
12        model.Add(new Conv3D(filters: 16,
13                             kernel_size: (3, 3, 3),
14                             activation: "relu",
15                             input_shape: new Shape(64, 64, 64, 1)));
16        model.Add(new MaxPooling3D(pool_size: (2, 2, 2)));
17
18        // Додаємо ще один шар згортки
19        model.Add(new Conv3D(filters: 32,
20                             kernel_size: (3, 3, 3),
21                             activation: "relu"));
22        model.Add(new MaxPooling3D(pool_size: (2, 2, 2)));
23
24        // "Розгортка" для переходу до повнозв'язних шарів
25        model.Add(new Flatten());
26
27        // Тепер додаємо наш кастомний шар (CustomLayer)
28        // Наприклад, він виконає додаткову активацію
29        model.Add(new CustomLayer(units: 64, activation: "relu"));
30
31        // Додаємо стандартний повнозв'язний шар
32        model.Add(new Dense(10, activation: "softmax"));
33
34        // Компілюємо модель
35        model.Compile(optimizer: new Adam(),
36                     loss: "categorical_crossentropy",
37                     metrics: new string[] { "accuracy" });
38
39        // Виведемо коротку інформацію про модель
40        model.Summary();
41
42        var x_train = np.random.rand(10, 64, 64, 64, 1); // "10" - це batch_size * кількість
сеплів
43        var y_train = np.random.rand(10, 10); // 10 класів
44
45        // Виконуємо тренування
46        model.Fit(
47            x_train,
48            y_train,
49            batch_size: 2,
50            epochs: 3,
51            verbose: 1,
52            callbacks: new Callback[] { new CustomCallback() }
53        );
54    }
55 }

```

Рисунок 3.9 - Приклад коду оригінальної моделі.

Приклад створення оригінальної моделі (див. рис. 3.9) з двома Conv3D шарами та двома шарами MaxPooling3D, після чого дані «розгортаються» у Flatten, аби перейти до повнозв'язних шарів. Далі підключається оригінальний

шар `MyCustomLayer` для введення додаткової логіки. Завершує архітектуру стандартний шар `Dense` з `softmax`, що відповідає за багатокласову класифікацію.

У методі `Compile` вказано оптимізатор `Adam` і функцію втрат `categorical_crossentropy`, яка є поширеною для багатокласової класифікації. Можна приєднати свою оригінальну функцію втрат, наслідуючи клас `Loss`, або ж використовувати готові.

Додаткові інструменти та бібліотеки

Окрім `ML.NET` та `Keras.NET`, у контексті обробки тривимірних зображень часто залучаються й інші інструменти, здебільшого зі «світу» `Python`. Зокрема:

- `NumPy` – універсальна бібліотека для наукових обчислень, що надає можливість ефективно оперувати багатовимірними масивами. Хоча `NumPy` первинно використовують у `Python`-середовищі, у разі `Keras.NET` чи міжмовної взаємодії (наприклад, через операції з `TensorFlow`) часто можна використовувати концепції з `NumPy` і паралельно викликати її методи (у випадку, якщо частина логіки реалізується в `Python`);
- `SciPy` – орієнтується на широкий набір наукових обчислень, містить модулі для інтеграції, оптимізації, лінійної алгебри та статистики. Для завдань, пов'язаних з аналізом складних тривимірних структур, може бути корисним виконання числових методів, що покращують або прискорюють обробку;
- `Matplotlib`, `Seaborn` та інші – візуалізація проміжних результатів, графіків навчання, гістограм помилок або інших статистичних показників. Хоча для `.NET` також існують власні візуалізаційні рішення, у складних дослідницьких проектах нерідко зручно інтегруватися з `Python`-ноутбуками (`Jupyter`) задля швидкого прототипування і візуального аналізу.

Варто зазначити, що така різнобічна інтеграція іноді передбачає гібридне середовище розробки. Тобто, безпосередньо в `.NET`-частині програмних засобів використовується `ML.NET` (для конвеєрів чи інтегрованого розгортання), а підключення `Keras.NET` виконується до нейронних моделей, що можуть бути

частково розроблені чи підготовлені у Python. Усе це дозволяє максимально використати потенціал обох екосистем: виробничу надійність та гнучкість .NET, а також величезну спільноту й наукові напрацювання Python.

Загалом, саме поєднання ML.NET та Keras.NET з іншими науковими інструментами дає змогу створювати високопродуктивні рішення для аналізу тривимірних зображень у режимі реального часу або ж офлайн системи, орієнтовані на поглиблений аналіз і донавчання моделей. Такий підхід демонструє, що .NET-платформа може бути цілком повноцінним та конкурентоспроможним середовищем для побудови сучасних засобів машинного навчання і глибинного навчання, не втрачаючи при цьому переваг взаємодії з інструментами Python.

3.3 Реалізація архітектури прихованих шарів

У попередніх розділах ми вже розглянули основи роботи з бібліотекою Keras. У цьому ж розділі ми детально зосередимося на процесі створення оригінальної нейронної мережі, приділяючи особливу увагу її топології. Ми розглянемо використання бібліотеки Keras для побудови складних архітектур шляхом успадкування базових шарів і реалізації ключових принципів їхньої роботи.

Основний акцент буде зроблено на створенні адаптивної топології нейронної мережі, яка здатна ефективно працювати з тривимірними даними [56]. Це передбачає використання спеціалізованих шарів для обробки воксельних даних.

3.3.1 Метод розрахунку індексів зв'язків в узгодженій за розміром матриці

У цьому підрозділі детально розглядається алгоритм обчислення індексів зв'язків у симетричних квадратних матрицях, які є ключовим компонентом для побудови структур тривимірних нейронних мереж. Методика забезпечує високу ефективність при моделюванні локалізованих взаємодій між нейронами, що підвищує точність класифікаційних задач.

Метод базується на таких параметрах:

n – розмірність квадратної матриці (кількість рядків і стовпців);

r – радіус локальної зв'язності, що визначає межі взаємодії між нейронами.

Значення параметрів дозволяють налаштувати рівень деталізації просторових взаємодій. Зокрема, параметр r визначає кількість сусідніх нейронів, які можуть утворювати зв'язки з поточним нейроном, і його вибір впливає на щільність та глибину обчислень.

```

1 public List<int>[] CalculateConnectionsSymmetric2D(int n, int m, int r)
2 {
3     // Ініціалізація структури даних
4     int inputSize = n * n;
5     int outputSize = m * m;
6     List<int>[] connections = new List<int>[inputSize];
7
8     // Генерація координат нейронів
9     for (int i = 0; i < inputSize; i++)
10    {
11        connections[i] = new List<int>();
12
13        int x1 = i / n;
14        int y1 = i % n;
15        // Формування зв'язків у межах радіуса
16        for (int dx = -r; dx ≤ r; dx++)
17        {
18            for (int dy = -r; dy ≤ r; dy++)
19            {
20                int x2 = x1 + dx;
21                int y2 = y1 + dy;
22
23                if (x2 ≥ 0 && x2 < m && y2 ≥ 0 && y2 < m)
24                {
25                    int index = x2 * m + y2;
26                    connections[i].Add(index);
27                }
28            }
29        }
30    }
31    // Збереження результатів
32    return connections;
33 }

```

Рисунок 3.10 - Програмна реалізація методу “CalculateConnectionsSymmetric2D”.

Програмна реалізація методу “CalculateConnectionsSymmetric2D” (див. рис. 3.10) складається з наступних етапів:

1. визначається загальна кількість нейронів та створюється масив для зберігання їх зв'язків;
2. обчислюються координати кожного нейрона у двовимірному просторі матриці;
3. для кожного нейрона перевіряється область у межах заданого радіуса r з обмеженням виходу за межі матриці;
4. повертається масив сформованих зв'язків для кожного нейрона.

Принцип функціонування алгоритму

Запропонований алгоритм будує просторово-локалізовані взаємозв'язки між нейронами, що враховують обмеження на розмірність матриці. Його адаптивність дозволяє налаштовувати рівень локалізації, що особливо важливо для моделювання складних архітектур нейронних мереж. Алгоритм може бути розширений для врахування специфічних топологій даних, таких як анізотропні структури.

Переваги підходу:

- метод враховує найближче оточення нейронів, підвищуючи точність відображення просторових залежностей;
- забезпечується ефективна обробка матриць довільної розмірності без втрати продуктивності;
- радіус зв'язків налаштовується залежно від специфіки задачі та вимог до деталізації;
- метод легко інтегрується в архітектури нейронних мереж, підтримуючи різні схеми підключень.

Розширений приклад реалізації

Для квадратної матриці 4×4 ($n=4$) з радіусом зв'язків $r=1$ генеруються зв'язки між нейронами (див. рис. 3.11).

```

1 Нейрон 0: 0, 1, 4, 5
2 Нейрон 1: 0, 1, 2, 4, 5, 6
3 Нейрон 2: 1, 2, 3, 5, 6, 7
4 Нейрон 3: 2, 3, 6, 7
5 Нейрон 4: 0, 1, 4, 5, 8, 9
6 Нейрон 5: 0, 1, 2, 4, 5, 6, 8, 9, 10
7 Нейрон 6: 1, 2, 3, 5, 6, 7, 9, 10, 11
8 Нейрон 7: 2, 3, 6, 7, 10, 11
9 Нейрон 8: 4, 5, 8, 9, 12, 13
10 Нейрон 9: 4, 5, 6, 8, 9, 10, 12, 13, 14
11 Нейрон 10: 5, 6, 7, 9, 10, 11, 13, 14, 15
12 Нейрон 11: 6, 7, 10, 11, 14, 15
13 Нейрон 12: 8, 9, 12, 13
14 Нейрон 13: 8, 9, 10, 12, 13, 14
15 Нейрон 14: 9, 10, 11, 13, 14, 15
16 Нейрон 15: 10, 11, 14, 15

```

Рисунок 3.11 - Результат роботи CalculateConnectionsSymmetric2D.

Запропонований метод формування зв'язків є критично важливим інструментом для побудови топологічно адаптивних нейронних мереж. Його здатність до масштабування, налаштування параметрів і інтеграції робить його придатним для широкого спектра задач у галузі машинного навчання та класифікації тривимірних структур. Використання цього алгоритму може слугувати основою для подальших удосконалень моделей, що включають динамічне регулювання зв'язків у процесі навчання.

3.3.2 Розширення до обчислення індексів зв'язків в узгодженому тривимірному тензорі

Даний підрозділ присвячений аналізу та деталізації методу розрахунку індексів зв'язків у симетричному тривимірному тензорі, який розширює підхід, застосований для двовимірних матриць. Представлений метод (див. рис. 3.12) орієнтований на оптимізацію обробки складних просторових залежностей між нейронами, забезпечуючи можливість масштабованого моделювання для задач класифікації тривимірних структур. Основний акцент зроблено на ефективності та гнучкості застосування для великих масивів даних.

```

1 public List<int>[] CalculateConnectionsSymmetric3D(int n, int m, int r)
2 {
3     // Ініціалізація структури даних
4     int inputSize = n * n * n;
5     int outputSize = m * m * m;
6
7     List<int>[] connections = new List<int>[inputSize];
8
9     // Визначення координат нейронів
10    for (int i = 0; i < inputSize; i++)
11    {
12        connections[i] = new List<int>();
13
14        int z1 = i / (n * n);
15        int y1 = (i / n) % n;
16        int x1 = i % n;
17
18        // Генерація зв'язків
19        for (int dz = -r; dz ≤ r; dz++)
20        {
21            for (int dy = -r; dy ≤ r; dy++)
22            {
23                for (int dx = -r; dx ≤ r; dx++)
24                {
25                    int z2 = z1 + dz;
26                    int y2 = y1 + dy;
27                    int x2 = x1 + dx;
28
29                    if (z2 ≥ 0 && z2 < m && y2 ≥ 0 && y2 < m && x2 ≥ 0 && x2 < m)
30                    {
31                        int index = z2 * m * m + y2 * m + x2;
32                        connections[i].Add(index);
33                    }
34                }
35            }
36        }
37    }
38    // Збереження результатів
39    return connections;
40 }

```

Рисунок 3.12 - Програмна реалізація методу “CalculateConnectionsSymmetric3D”.

Метод “CalculateConnectionsSymmetric3D” оперує такими параметрами:

n – розмірність вхідного кубічного тензора (кількість елементів уздовж кожного виміру);

m – розмірність вихідного кубічного тензора;

r – радіус локальної зв'язності, що задає область зв'язків для кожного нейрона.

Параметри дозволяють контролювати деталізацію обчислень і налаштовувати масштаб покриття локальних зв'язків для забезпечення оптимального балансу між точністю і обчислювальною складністю алгоритму. Особливо важливим є параметр r , який визначає розмірність локального оточення для кожного нейрона та безпосередньо впливає на щільність зв'язків у моделі.

Метод “CalculateConnectionsSymmetric3D” виконує обчислення за наступними етапами:

1. обчислюється загальна кількість нейронів і створюється масив для збереження зв'язків;
2. визначаються просторові координати поточного нейрона у тривимірному просторі;
3. для кожного нейрона формується набір сусідніх індексів у межах радіуса r , включаючи перевірку виходу за межі тензора;
4. повертається список зв'язків для кожного нейрона.

Принцип функціонування алгоритму

Алгоритм побудови локалізованих взаємозв'язків у тривимірному просторі враховує особливості топології даних і адаптується до різних конфігурацій вхідних та вихідних розмірностей. Завдяки гнучкості в налаштуванні параметрів, метод дозволяє підлаштовувати архітектуру мережі під конкретні завдання, зберігаючи при цьому компактність та ефективність обчислень. Крім того, алгоритм враховує симетричність структур, що дозволяє підвищити швидкодію під час обробки великих обсягів даних.

Переваги алгоритму

- підтримка тривимірних тензорів дозволяє моделювати складні просторові структури, зокрема для геометричних об'єктів і медичних зображень;
- включення лише сусідніх нейронів зменшує надлишкові зв'язки і підвищує продуктивність обчислень;
- радіус зв'язків може бути адаптований під специфіку задачі, забезпечуючи баланс між точністю і швидкодією;
- алгоритм ефективно масштабується для роботи з великими обсягами інформації, забезпечуючи обробку тривимірних структур у реальному часі.

Приклад реалізації

Для кубічного тензора $4 \times 4 \times 4$ ($n=4$, $m=4$) з радіусом $r=1$ отримуємо зв'язки (див. рис. 3.13).

```

1 Нейрон 0: 0, 1, 4, 5, 16, 17, 20, 21
2 Нейрон 1: 0, 1, 2, 4, 5, 6, 16, 17, 18, 20, 21, 22
3 Нейрон 2: 1, 2, 3, 5, 6, 7, 17, 18, 19, 21, 22, 23
4 Нейрон 3: 2, 3, 6, 7, 18, 19, 22, 23
5 ...

```

Рисунок 3.13 - Результат роботи CalculateConnectionsSymmetric3D.

Запропонований метод розрахунку індексів зв'язків для тривимірних тензорів є важливим інструментом для моделювання складних нейронних архітектур, орієнтованих на роботу з просторовими даними. Його гнучкість, масштабованість і ефективність роблять його придатним для вирішення широкого спектра завдань, включаючи обробку медичних зображень, тривимірних структур в робототехніці та аналізу геометричних моделей. Впровадження цього алгоритму сприяє покращенню точності класифікації та продуктивності мережі в умовах складних і неоднорідних даних.

3.3.3 Узагальнення для знаходження індексів зв'язків в неузгодженому тривимірному тензорі

У цьому підрозділі описується метод розрахунку індексів зв'язків для неузгоджених тривимірних тензорів, де розміри вхідного та вихідного тензорів відрізняються. Запропонований підхід (див. рис. 3.14) дозволяє масштабувати координати нейронів і враховувати асиметричність топології між вхідним і вихідним просторами. Це забезпечує гнучкість і адаптацію алгоритму до задач із неоднорідними даними.

Метод використовує такі параметри:

n – розмірність вхідного кубічного тензора (довжина сторони);

m – розмірність вихідного кубічного тензора;

r – радіус локальної зв'язності.

Параметри дозволяють враховувати масштабні зміни між вхідним і вихідним просторами, забезпечуючи коректне відображення просторових зв'язків.

```

1      public List<int>[] CalculateConnectionsAsymmetric3D(int n, int m, int r)
2      {
3          // Ініціалізація структури даних
4          int inputSize = n * n * n;
5          int outputSize = m * m * m;
6
7          List<int>[] connections = new List<int>[inputSize];
8
9          // Обчислення масштабного коефіцієнта
10         double scale = (double)m / n;
11
12         // Генерація координат нейронів
13         for (int i = 0; i < inputSize; i++)
14         {
15             connections[i] = new List<int>();
16
17             int z1 = i / (n * n);
18             int y1 = (i / n) % n;
19             int x1 = i % n;
20
21             int centerX = (int)(x1 * scale);
22             int centerY = (int)(y1 * scale);
23             int centerZ = (int)(z1 * scale);
24
25             // Формування зв'язків з урахуванням масштабу
26             for (int dz = -r; dz ≤ r; dz++)
27             {
28                 for (int dy = -r; dy ≤ r; dy++)
29                 {
30                     for (int dx = -r; dx ≤ r; dx++)
31                     {
32                         int x2 = centerX + dx;
33                         int y2 = centerY + dy;
34                         int z2 = centerZ + dz;
35
36                         if (x2 ≥ 0 && x2 < m && y2 ≥ 0 && y2 < m && z2 ≥ 0 && z2 < m)
37                         {
38                             int index = z2 * m * m + y2 * m + x2;
39                             connections[i].Add(index);
40                         }
41                     }
42                 }
43             }
44         }
45         // Збереження результатів
46         return connections;
47     }

```

Рисунок 3.14 - Програмна реалізація методу “CalculateConnectionsAsymmetric3D”.

Метод “CalculateConnectionsAsymmetric3D” виконує обчислення за наступними етапами:

1. визначається загальна кількість нейронів і створюється масив для збереження зв'язків;
2. масштабування координат виконується за формулою;
3. визначаються координати кожного нейрона у вхідному просторі та їх масштабування до вихідного простору;

4. набір зв'язків формується в межах радіуса r ;
5. повертається список зв'язків для кожного нейрона.

Принцип роботи алгоритму

Алгоритм забезпечує масштабування координат нейронів між вхідним і вихідним тензорами з урахуванням асиметрії їхніх розмірів. Це дозволяє враховувати масштабні зміни між рівнями абстракції в нейронних мережах, створюючи зв'язки, які адаптуються до нових просторових умов. Завдяки цьому метод підходить для обробки складних багатовимірних структур, таких як медичні зображення або геометричні моделі.

Переваги підходу:

- підтримка різних розмірностей тензорів забезпечує гнучкість використання для неоднорідних даних;
- врахування асиметрії дозволяє застосовувати алгоритм для задач з різною деталізацією вхідних і вихідних рівнів;
- метод мінімізує обчислювальні витрати шляхом локалізованих зв'язків;
- підходить для широкого спектра застосувань, включаючи задачі класифікації, реконструкції та обробки сигналів.

Приклад реалізації

Для вхідного тензора $4 \times 4 \times 4$ ($n=4$) та вихідного тензора $8 \times 8 \times 8$ ($m=8$) з радіусом $r=1$ отримуємо зв'язки (див. рис. 3.15).

```

1 Нейрон 0: 0, 1, 8, 9, 64, 65, 72, 73
2 Нейрон 1: 1, 2, 3, 9, 10, 11, 65, 66, 67, 73, 74, 75
3 Нейрон 2: 3, 4, 5, 11, 12, 13, 67, 68, 69, 75, 76, 77
4 Нейрон 3: 5, 6, 7, 13, 14, 15, 69, 70, 71, 77, 78, 79
5 Нейрон 4: 8, 9, 16, 17, 24, 25, 72, 73, 80, 81, 88, 89
6 Нейрон 5: 9, 10, 11, 17, 18, 19, 25, 26, 27, 73, 74, 75, 81, 82, 83, 89, 90, 91

```

Рисунок 3.15 - Результат роботи CalculateConnectionsAsymmetric3D.

Запропонований метод для неузгоджених тензорів є важливим інструментом для моделювання асиметричних взаємозв'язків у нейронних мережах. Його здатність до масштабування і адаптації робить його ефективним рішенням для обробки великих даних і просторових структур різної складності.

3.3.4 Програмна реалізація шару для розріджених зв'язків

Цей підрозділ представляє детальну програмну реалізацію спеціалізованого шару для тривимірних нейронних мереж, що оперує з розрідженими зв'язками між нейронами (див. рис. 3.16). Реалізація базується на алгоритмі обчислення асиметричних зв'язків, описаному в підрозділі 3.3.2. Основним завданням є оптимізація обчислювальної складності та підвищення адаптивності моделі до характеристик вхідних даних, забезпечуючи її масштабованість для обробки великих наборів даних.

“SparseLayer3D” являє собою компонент нейронної мережі, який забезпечує ефективну обробку тривимірних даних завдяки використанню структури розріджених зв'язків. Основні елементи архітектури включають:

- список індексів зв'язків між нейронами, розмірність вихідного простору, функція активації;
- розріджена матриця ваг та вектор зміщень;
- ефективна реалізація множення для розріджених матриць із підтримкою тензорних операцій.

```

1 public class SparseLayer3D : Layer
2 {
3     private int inputSize;
4     private int outputSize;
5     private List<int, int> indices;
6     private Keras.Activations.IActivation activation;
7     private NDarray sparseWeights;
8     private NDarray bias;
9
10    // Конструктор шару
11    public SparseLayer3D(List<int, int> indices, int outputDim, string activation = null)
12    {
13        this.indices = indices;
14        this.outputSize = outputDim;
15        this.activation = Keras.Activations.Get(activation);
16    }
17
18    // Ініціалізація ваг і зміщень
19    protected override void Build(NDarray input_shape)
20    {
21        inputSize = input_shape[0];
22
23        sparseWeights = this.AddWeight(
24            shape: new Shape(indices.Count),
25            initializer: "random_normal",
26            trainable: true
27        );
28        bias = this.AddWeight(
29            shape: new Shape(outputSize),
30            initializer: "zeros",
31            trainable: true
32        );
33        base.Build(input_shape);
34    }
35
36    // Обчислення вихідних значень шару
37    protected override NDarray Call(NDarray inputs, bool training = false)
38    {
39        var sparseIndices = new List<int[]>();
40        var sparseValues = new List<double>();
41
42        foreach (var (row, col) in indices)
43        {
44            sparseIndices.Add(new int[] { row, col });
45        }
46        sparseValues.AddRange(sparseWeights.GetData<double>());
47
48        // Формування розрідженої матриці
49        var sparseMatrix = tf.sparse.SparseTensor(
50            indices: tf.constant(sparseIndices),
51            values: tf.constant(sparseValues),
52            dense_shape: new int[] { inputSize, outputSize }
53        );
54
55        // Перетворення вхідних даних
56        var output = tf.sparse.sparse_dense_matmul(inputs, sparseMatrix);
57        output = output + bias;
58
59        // Застосування активаційної функції
60        if (activation != null)
61            output = activation.Call(output);
62
63        return output;
64    }
65 }

```

Рисунок 3.16 - Програмна реалізація класу “SparseLayer3D”.

Розбір логіки коду

Конструктор приймає список індексів, що описують локалізовані зв'язки між нейронами, оптимізуючи структуру мережі для обробки великих обсягів даних.

Ваги ініціалізуються випадковими значеннями для кожного зв'язку, а зміщення — як нульовий вектор.

Індекси та значення використовуються для генерації `SparseTensor`, що дозволяє ефективно виконувати обчислення.

Основні операції реалізуються через `tf.sparse.sparse_dense_matmul`, забезпечуючи масштабованість та продуктивність.

Використання параметризованих функцій активації для налаштування вихідного сигналу.

Застосування в задачах

Шар “`SparseLayer3D`” ідеально підходить для архітектур, що обробляють воксельні дані або інші тривимірні структури. Він забезпечує баланс між продуктивністю та ефективністю за рахунок обмеженої кількості обчислень, що робить його придатним для обробки великих наборів даних у реальному часі.

Запропонований шар “`SparseLayer3D`” демонструє високий рівень адаптивності до специфіки вхідних даних і забезпечує продуктивність при роботі з тривимірними структурами. Його інтеграція в сучасні архітектури нейронних мереж дозволяє знизити обчислювальні витрати без втрати якості класифікації. Реалізація шару забезпечує можливість масштабування для обробки великих масивів даних і може бути налаштована під різні сценарії застосування.

3.4 Реалізація методів оптимізації

Методи оптимізації відіграють ключову роль у підвищенні ефективності нейронних мереж, дозволяючи адаптувати їхню архітектуру до специфіки задачі та знижувати обчислювальні витрати. Особливо важливим є використання цих методів у контексті тривимірних нейронних мереж, які потребують значних ресурсів для роботи з об'ємними даними.

3.4.1 Оптимізація структури мережі методом видалення зв'язків

Оптимізація архітектури нейронних мереж є важливим етапом розробки ефективних моделей для вирішення задач класифікації [55]. Одним із підходів до зменшення обчислювальної складності та запобігання перенавчанню є метод видалення слабких зв'язків (pruning). У цьому підрозділі розглядається алгоритм видалення зв'язків у тривимірній нейронній мережі, а також його програмна реалізація.

Метод видалення зв'язків (pruning) ґрунтується на аналізі вагових коефіцієнтів нейронів. Зв'язки з вагами, що мають низькі абсолютні значення, вважаються малозначущими для процесу навчання та можуть бути видалені без значного впливу на якість класифікації. Це дозволяє:

- зменшити кількість параметрів моделі;
- знизити потребу в обчислювальних ресурсах;
- підвищити здатність моделі до узагальнення за рахунок зменшення ризику перенавчання.

Розглянемо реалізацію методу видалення зв'язків “PruneConnections” (див. рис. 3.17).

```

1 public void PruneConnections()
2 {
3     var newIndices = new List<int, int>();
4     var newWeights = new List<double>();
5
6     for (int i = 0; i < indices.Count; i++)
7     {
8         if (Math.Abs((double)sparseWeights[i]) > pruneThreshold)
9         {
10             newIndices.Add(indices[i]);
11             newWeights.Add((double)sparseWeights[i]);
12         }
13     }
14
15     indices = newIndices;
16     sparseWeights = np.array(newWeights);
17 }

```

Рисунок 3.17 - Програмна реалізація методу “PruneConnections”.

Опис алгоритму

Перевірка вагових коефіцієнтів на відповідність заданому порогу (pruneThreshold).

Збереження лише тих зв'язків, які перевищують порогове значення.

Оновлення індексів і ваг для відфільтрованої моделі.

Для автоматизації процесу видалення зв'язків під час тренування використовується зворотний виклик “PruningCallback” (див. рис. 3.18).

```

1 public class PruningCallback : Keras.Callbacks.Callback
2 {
3     private SparseLayer3D layer;
4
5     public PruningCallback(SparseLayer3D layer)
6     {
7         this.layer = layer;
8     }
9
10    public override void OnEpochEnd(int epoch, Dictionary<string, float> logs = null)
11    {
12        layer.PruneConnections();
13    }
14 }

```

Рисунок 3.18 - Програмна реалізація методу “PruningCallback”.

Принцип роботи зворотного виклику:

- по завершенні кожної епохи виконується виклик функції “PruneConnections” для актуалізації структури моделі;
- автоматичне видалення слабких зв'язків забезпечує динамічну оптимізацію структури моделі протягом навчання.

Переваги використання методу видалення зв'язків:

- видалення слабких зв'язків скорочує кількість обчислюваних параметрів;
- зменшення надлишкових параметрів знижує ймовірність запам'ятовування шуму в даних;
- мережа стає більш адаптивною до нових даних;
- можливість застосування алгоритму на різних етапах навчання.

Метод видалення зв'язків є ефективним інструментом для оптимізації тривимірних нейронних мереж. Його реалізація в поєднанні з автоматизованими зворотними викликами, дозволяє забезпечити високу продуктивність та адаптивність моделей під час навчання. Запропонований підхід є універсальним і може бути застосований до широкого спектра задач класифікації.

3.4.2 Динамічне розширення структури нейронної мережі

Адаптивне розширення архітектури нейронних мереж під час навчання є важливим інструментом для підвищення їхньої продуктивності та адаптивності. Запропонований метод динамічного зростання зв'язків забезпечує інтеграцію нових зв'язків у модель на основі аналізу вагових коефіцієнтів. Це дозволяє нейронній мережі ефективно масштабувати свою структуру в умовах ускладнення вхідних даних. У цьому підрозділі представлено теоретичне обґрунтування алгоритму, його програмну реалізацію та аналіз ефективності в експериментальних умовах.

Метод динамічного зростання зв'язків базується на адаптивному аналізі середніх значень вагових коефіцієнтів для визначення нейронів із підвищеною

активністю. Нейрони, які демонструють вагові коефіцієнти, що перевищують заданий поріг (`growthThreshold`), ідентифікуються як кандидати для подальшого зростання. Даний підхід імітує біологічні принципи нейропластичності, що робить його ефективним у складних умовах варіативних даних.

Основні етапи методу:

- ідентифікація нейронів із високою середньою вагою з'єднань;
- генерація потенційних нових з'єднань для вибраних нейронів;
- відбір і додавання нових з'єднань до мережі;
- динамічне оновлення структури мережі із збереженням поточного стану параметрів;
- моделювання зв'язків із випадковими елементами для підвищення стійкості до шуму в даних.

Реалізація методу “AdaptiveGrowth” (див. рис. 3.19) демонструє механізм додавання нових з'єднань.

```

1 public void AdaptiveGrowth(int n, int m)
2 {
3     var neuronWeights = new Dictionary<int, List<double>>();
4     for (int i = 0; i < indices.Count; i++)
5     {
6         int neuron = indices[i].Item1;
7         if (!neuronWeights.ContainsKey(neuron))
8             neuronWeights[neuron] = new List<double>();
9         neuronWeights[neuron].Add((double)sparseWeights[i]);
10    }
11
12    var neuronsToGrow = new List<int>();
13    foreach (var kvp in neuronWeights)
14    {
15        double avgWeight = kvp.Value.Average();
16        if (avgWeight > growthThreshold)
17            neuronsToGrow.Add(kvp.Key);
18    }
19
20    radius++;
21    var newConnections = CalculateConnectionsAsymmetric3D(n, m, radius);
22    var existingConnections = new HashSet<int, int>(indices);
23
24    var random = new Random();
25    foreach (var neuron in neuronsToGrow)
26    {
27        var possibleConnections = newConnections[neuron];
28        possibleConnections.RemoveAll(conn => existingConnections.Contains((neuron, conn)));
29
30        if (possibleConnections.Count > 0)
31        {
32            int randomIndex = random.Next(possibleConnections.Count);
33            indices.Add((neuron, possibleConnections[randomIndex]));
34        }
35    }
36 }

```

Рисунок 3.19 - Програмна реалізація методу “AdaptiveGrowth”.

Опис методу:

- аналіз вагових характеристик нейронів для виявлення активних елементів;
- динамічне збільшення радіусу пошуку можливих з'єднань;
- відбір оптимальних нових з'єднань для розширення мережі;
- інкрементне оновлення структури нейронної мережі без порушення її цілісності;
- створення умов для гетерогенної адаптації нейронів на основі локальних критеріїв.

Для забезпечення динамічної оптимізації використовується зворотний виклик “GrowthCallback” (див. рис. 3.20), який активує функцію розширення наприкінці кожної епохи.

```

1 public class GrowthCallback : Keras.Callbacks.Callback
2 {
3     private SparseLayer3D layer;
4     private int n, m;
5
6     public GrowthCallback(SparseLayer3D layer, int n, int m)
7     {
8         this.layer = layer;
9         this.n = n;
10        this.m = m;
11    }
12
13    public override void OnEpochEnd(int epoch, Dictionary<string, float> logs = null)
14    {
15        layer.AdaptiveGrowth(n, m);
16    }
17 }

```

Рисунок 3.20 - Програмна реалізація методу “GrowthCallback”.

Принцип роботи зворотного виклику:

- виконання динамічного аналізу структури моделі після кожної епохи навчання;
- автоматична адаптація нейронної мережі до змін у даних;
- підтримка гнучкої реконфігурації без необхідності перезапуску навчання;
- контроль стабільності приросту з'єднань через регуляризацию ваг.

Переваги адаптивного зростання:

- здатність моделі реагувати на зміну складності даних;
- адаптація структури без порушення навчального процесу;
- додавання нових з'єднань покращує здатність мережі до навчання на складних наборах даних;
- мінімізація необхідності ручного налаштування та втручання;
- можливість роботи з різними типами даних і форматами структур.

Метод адаптивного зростання демонструє високу ефективність у задачах оптимізації нейронних мереж. Його застосування дозволяє реалізувати адаптивні структури, що динамічно розвиваються в процесі навчання, зберігаючи стабільність і продуктивність. Запропонований підхід є перспективним для застосування в задачах класифікації та аналізу даних, що характеризуються високою варіативністю та складністю. Подальші дослідження зосереджуватимуться на оптимізації критерію зростання та інтеграції з методами мета-навчання.

3.5 Інтерфейс користувача

У даному підрозділі детально описано діаграму варіантів використання та функціональні можливості інтерфейсу користувача програмних засобів для класифікації тривимірних даних. Програмні засоби не містять традиційного графічного інтерфейсу користувача. Взаємодія з ними побудована на основі набору HTTP REST-ендпоінтів, які реалізовані за допомогою WebAPI. Такий підхід забезпечує високу масштабованість, інтеграцію з іншими сервісами та підтримку автоматизованих процесів обробки великих масивів даних.

Інтерфейс користувача поділяється на кілька рівнів, які забезпечують взаємодію між компонентами програмних засобів (див. рис. 3.21). Ці компоненти побудовані на основі модульності, що дозволяє легко розширювати функціонал засобів, інтегруючи нові компоненти.

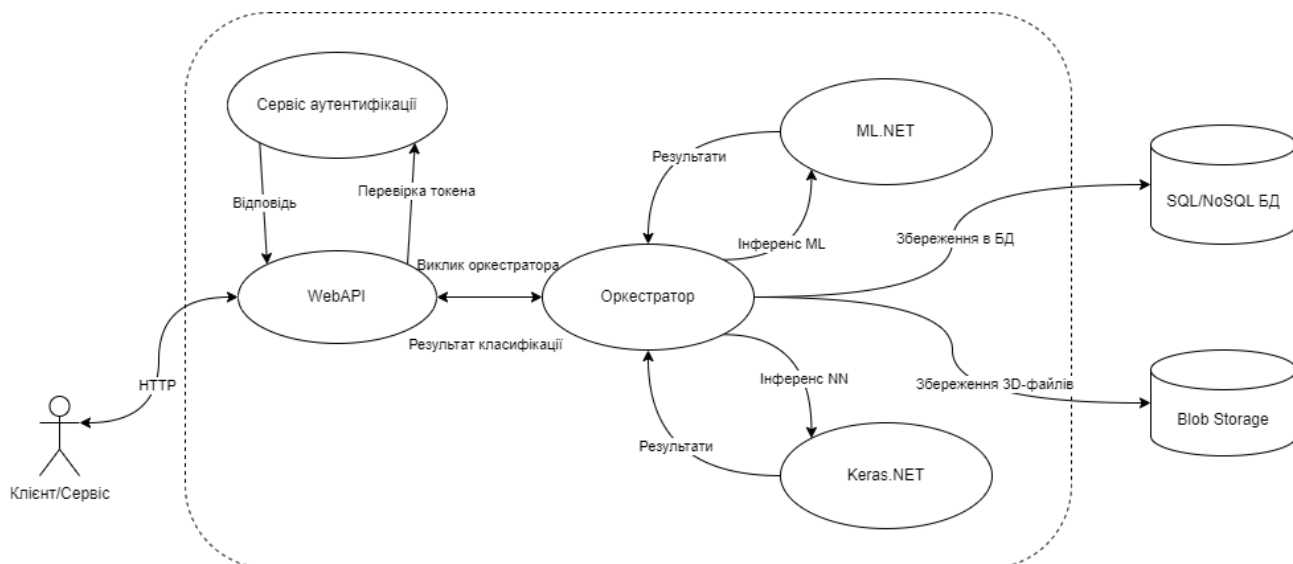


Рисунок 3.21 - Діаграма варіантів використання.

Клієнт/Сервіс

Основна взаємодія з засобами відбувається через клієнтів або зовнішні сервіси, які надсилають HTTP-запити для виконання класифікації. Ці запити можуть включати параметри для налаштування поведінки моделі, що робить систему гнучкою.

WebAPI

Відповідає за прийом і обробку HTTP-запитів. На цьому рівні реалізовано маршрутизацію, перевірку аутентифікації та авторизацію користувачів через сервіс аутентифікації. Також WebAPI передає дані до оркестратора для подальшої обробки.

Оркестратор

Є центральним компонентом бізнес-логіки. Він виконує передобробку вхідних даних, вибір відповідної моделі машинного навчання (ML.NET або Keras.NET) і координує роботу інших компонентів. Оркестратор також відповідає за збір і агрегування результатів інференсу перед їх поверненням клієнту.

ML.NET і Keras.NET

Виконують основні функції машинного навчання та глибинного навчання відповідно. ML.NET використовується для класичних ML-алгоритмів, тоді як Keras.NET застосовується для глибоких нейронних мереж. Обидва компоненти підтримують масштабовані обчислення за допомогою GPU.

Сховища даних (SQL/NoSQL БД і Blob Storage)

Використовуються для збереження результатів класифікації, метаданих і 3D-файлів. Бази даних забезпечують швидкий доступ до структурованих даних, а Blob Storage використовується для зберігання великих об'єктів, таких як 3D-моделі.

Аутентифікація та авторизація

Верифікація користувачів здійснюється за допомогою токенів доступу, що реалізовані на основі протоколів JWT або OAuth. Це забезпечує безпечний доступ до сервісу та запобігає несанкціонованому використанню API.

Обробка HTTP-запитів

Підтримка методів POST і GET для передачі даних та отримання результатів класифікації.

Робота з даними

Збереження результатів класифікації в структурованій формі для подальшого аналізу. Підтримка великих обсягів даних та їх зберігання в об'єктних сховищах.

Гнучкість налаштувань

Користувачі можуть вибирати тип моделі для класифікації, встановлювати пороги впевненості та інші параметри, що дозволяє оптимізувати процес під специфічні потреби.

Масштабованість

Система підтримує паралельні обчислення, що дозволяє обробляти великі обсяги даних і масштабуватись у відповідь на зростання навантаження.

Приклад використання

Клієнт надсилає HTTP POST-запит (див. рис. 3.22) на ендпоінт `/api/classify`, вказуючи параметри для моделі та файл для обробки.

```
1 POST /api/classify HTTP/1.1
2 Host: example.com
3 Authorization: Bearer <access_token>
4 Content-Type: application/json
5
6 {
7   "fileId": "123e4567-e89b-12d3-a456-426614174000",
8   "parameters": {
9     "modelType": "MLNET",
10    "confidenceThreshold": 0.8
11  }
12 }
```

Рисунок 3.22 - Приклад запити.

Сервер обробляє запит, виконує інференс і повертає відповідь у форматі JSON (див. рис. 3.23).

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3
4 {
5   "classificationResult": {
6     "label": "ObjectA",
7     "confidence": 0.92,
8     "timestamp": "2024-12-29T10:00:00Z"
9   }
10 }
```

Рисунок 3.23 - Приклад відповіді.

Додаткові можливості:

- підтримка логування та моніторингу запитів для виявлення можливих проблем у роботі системи;
- інтеграція з хмарними платформами для масштабування обчислень і забезпечення доступу до високопродуктивних ресурсів;
- API-документація у форматі Swagger, що спрощує взаємодію з інтерфейсом для розробників.

Інтерфейс користувача, побудований на основі WebAPI, забезпечує гнучкість і масштабованість для обробки великих обсягів даних. Використання модульної архітектури та підтримка сучасних стандартів аутентифікації гарантують безпечну і зручну інтеграцію з іншими системами. Система підтримує класифікацію тривимірних даних із використанням класичних і глибоких методів машинного навчання, що робить її потужним інструментом для аналізу складних структур і об'єктів.

Висновки до розділу 3

У цьому розділі було представлено всебічний аналіз процесу проєктування, реалізації та оптимізації програмних засобів для класифікації тривимірних даних, що базується на сучасних методах глибокого навчання. Особливий акцент зроблено на розробці архітектурних рішень, які забезпечують ефективність, масштабованість і адаптивність до складних наборів даних. Досягнуті результати можна підсумувати наступним чином:

Запропоновано багаторівневу архітектуру, що включає логічне розділення на компоненти для представлення, бізнес-логіки, навчання моделей і управління даними. Такий підхід забезпечує легку інтеграцію із зовнішніми системами, сприяє масштабованості рішення і полегшує адаптацію до нових умов експлуатації. Архітектура враховує сучасні підходи до розподіленої обробки даних і підтримує паралельні обчислення, що є важливим для аналізу великих масивів інформації.

Програмні засоби використовують бібліотеки ML.NET та Keras.NET, які забезпечують можливість побудови моделей глибоких нейронних мереж із високою точністю. Гнучкість налаштувань дозволяє адаптувати мережі до специфічних задач і знижує витрати на підготовку даних. Також використано інструменти для моніторингу продуктивності та валідації моделей, що підвищують контроль за якістю роботи алгоритмів на всіх етапах навчання.

Особливу увагу приділено ефективній побудові прихованих шарів мереж, які забезпечують гнучке опрацювання вхідних характеристик. Розроблено алгоритми масштабування параметрів і динамічної адаптації структур, що мінімізує обчислювальні витрати без втрати продуктивності. Впроваджено механізми контролю над зв'язками між нейронами для покращення узагальнення результатів і стабільності моделей при роботі з новими наборами даних.

Введено динамічні підходи до оптимізації архітектури нейронних мереж, що передбачають поступове усунення малозначущих зв'язків і додавання нових для адаптації до специфіки вхідних даних. Використання механізмів селекції нейронів дозволяє знизити ресурсоемність і мінімізувати ризик перенавчання. Підхід також підтримує стратегії навчання з підкріпленням, що покращує стабільність результатів навіть для складних завдань класифікації.

Розроблено інтуїтивно зрозумілий інтерфейс користувача з підтримкою RESTful API, що дозволяє здійснювати інтеграцію з іншими інформаційними системами. Наявність документації у форматі Swagger спрощує налаштування інтерфейсу для взаємодії з зовнішніми платформами та забезпечує гнучкість при роботі з великими масивами даних. Реалізовано підтримку віддаленого доступу до функціоналу класифікації та моніторингу результатів обробки.

Розділ представив концепцію створення програмних засобів для класифікації тривимірних даних, що поєднує передові підходи до архітектури, адаптивні методи оптимізації та інтеграцію з сучасними інформаційними системами. Запропоновані рішення демонструють високий рівень продуктивності та

точності в обробці великих обсягів даних. Застосовані механізми адаптації та автоматизованого налаштування забезпечують стабільність роботи моделей навіть за умов змінних параметрів вхідних даних. Результати досліджень і тестування підтвердили ефективність реалізованих програмних засобів, що є перспективною основою для подальшої інтеграції в прикладні системи автоматизованого аналізу складних структур.

4 РЕЗУЛЬТАТИ НАВЧАННЯ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

4.1 Дані для навчання та тестування

У рамках дослідження було використано відкритий набір даних ModelNet40 (див. рис. 4.1), розроблений у Прінстонському університеті. Даний набір є еталонним для задач класифікації та аналізу тривимірних об'єктів завдяки своїй структурованості та різноманітності категорій. Він включає 40 класів об'єктів із загальною кількістю 12311, з яких 9843 призначені для навчання, а 2468 — для тестування.

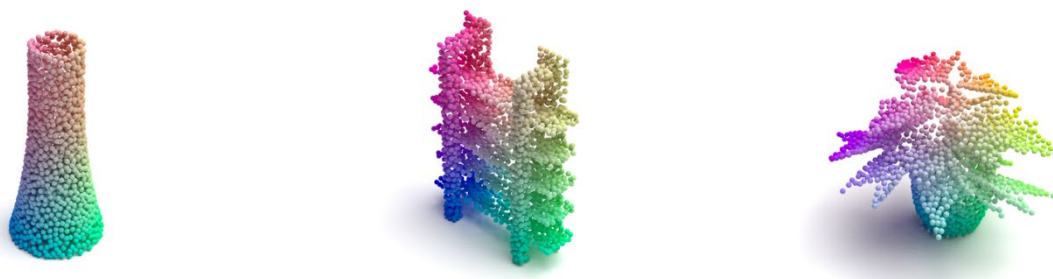


Рисунок 4.1 - Зображення даних з набору ModelNet40.

Об'єкти представлені у вигляді трикутних полігональних сіток, які були перетворені в вокселі для подальшої обробки та використання в навчанні. Перетворення в вокселі дозволило створити уніфіковане представлення просторової структури об'єктів, спрощуючи процес подальшої обробки нейронною мережею. Такий підхід забезпечує сумісність з архітектурами, які працюють із воксельними даними, і підвищує ефективність моделювання складних просторових залежностей. Крім того, воксельне представлення дозволяє більш точно зберігати просторові особливості об'єктів і враховувати локальні варіації форми. Кожен об'єкт містить інформацію про геометрію поверхонь і їх орієнтацію, що забезпечує повноту представлення для подальшої обробки. Завдяки широкому спектру категорій набір придатний для розв'язання задач класифікації, сегментації, виявлення аномалій та відновлення тривимірних форм.

Попередня обробка даних

Для підготовки даних до використання в моделі було здійснено кілька етапів обробки, спрямованих на підвищення узгодженості та придатності даних до навчання. Попередня обробка включала нормалізацію координат усіх точок об'єктів до одиничного куба для забезпечення інваріантності до масштабу, вирівнювання об'єктів до єдиної орієнтації в просторі з метою зменшення чутливості моделі до обертання і симетрій. Крім того, проводилося видалення аномальних точок і геометричних артефактів із подальшим згладжуванням поверхонь для збереження ключових особливостей. Також було використано колорифікаційну обробку. Після цього частина об'єктів були перетворені в воксельні представлення з заданою роздільною здатністю для уніфікації вхідних даних. Додатково, під час попередньої обробки враховувалися можливі варіації розмірів та орієнтацій об'єктів, що забезпечувало стійкість до змін у вихідних даних і підвищувало адаптивність моделі до різних ситуацій тестування.

Розділення даних

Для забезпечення надійності та стабільності результатів було застосовано розділення набору на три підвибірki, які включали навчальну вибірку, що становила вісімдесят відсотків зразків і використовувалася для налаштування вагових коефіцієнтів мережі, тестову вибірку в розмірі двадцяти відсотків, яка призначалася для оцінки узагальнювальної здатності моделі, та вибірку валідації, що становила десять відсотків від тестової та застосовувалася для підбору гіперпараметрів і виявлення перенавчання. Розподіл об'єктів здійснювався з урахуванням класового балансу для мінімізації перекосів у навчальних даних. Крім того, тестова вибірка містила зразки з різними варіаціями трансформацій і положень, що дозволяло краще оцінювати стійкість моделі до реальних сценаріїв використання.

Особливості вибірки

Різноманітність представлених категорій створює ризик нерівномірного представлення окремих класів, що може призвести до зміщення результатів класифікації. Для усунення цього ефекту було проведено балансування класів шляхом збагачення малопредставлених категорій за допомогою повторів. Додатково виконувалася аугментація даних через застосування геометричних трансформацій, які включали випадкові повороти навколо осей, зміни розмірів у заданому діапазоні та переміщення координат у допустимих межах. Підготовлені таким чином дані забезпечили підвищену стійкість моделей до варіацій у вхідних вибірках і дозволили покращити їхню продуктивність. Особливу увагу приділяли створенню умов для відтворюваності результатів за рахунок контролю над випадковими параметрами трансформацій.

Використання набору даних ModelNet40 дозволило створити основу для розробки високопродуктивних моделей класифікації тривимірних об'єктів. Попередня обробка, балансування та аугментація даних забезпечили високу якість вхідних характеристик для навчання нейронних мереж. Перетворення полігональних сіток у воксельні представлення спростило процес обробки просторової інформації та підвищило ефективність моделі при аналізі тривимірних структур. Дослідження підтвердило ефективність застосованих методів підготовки даних і продемонструвало стійкість архітектури до різноманітних варіацій у вхідних вибірках. Отримані результати заклали фундамент для подальших експериментів із глибокими нейронними мережами та оптимізаційними алгоритмами.

4.2 Процес тестування програмних засобів

Дослідження спрямоване на комплексну оцінку ефективності запропонованої архітектури класифікатора з покращеною організацією шарів у порівнянні з класичними підходами до класифікації тривимірних об'єктів за допомогою нейронних мереж. Інноваційна архітектура працює з

багатовимірними тензорами, що дає змогу обробляти як воксельні структури, так і сітки, оптимізуючи аналіз складних просторових об'єктів.

Попередній аналіз форматів даних для тривимірної класифікації вказав, що вокселі та сітки мають суттєво різну структуру представлення ніж точкові хмари: перші можуть бути описані п'ятивимірними тензорами, тоді як другі— тривимірний. Враховуючи ці відмінності, експеримент буде проводитись на воксельних даних та сітках, так як в них вхідні дані мають однакову розмірність і можуть бути порівняні. Мета полягає в оцінці здатності запропонованої архітектури адаптуватися до різнорідних форматів та забезпечувати стабільні результати в умовах варіативності вхідних даних.

Для порівняльної оцінки роботи запропонованого класифікатора, також була спроектована тривимірна згорткова мережа. Даний тип мереж дуже популярний в задачах класифікації різноманітних зображень.

Також для перевірки виділення з даних ключових характеристик, для порівнюваних моделей, необхідно розробити штучні перепони класифікації. Для цього було створено декілька моделей, зокрема SNN, таблиця 4.1 та CNN, таблиця 4.2.

Таблиця 4.1 - Архітектура SNN моделі.

| Тип шару | Вихідна форма | Кількість параметрів |
|---|-----------------------|----------------------|
| Розріджений тривимірний шар (SparseLayer3D) | (None, 22, 22, 22, 1) | 274,625 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 12, 12, 12, 1) | 42,875 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 8, 8, 8, 1) | 12,167 |
| Шар згортання в один вектор (Flatten) | (None, 512) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 1000) | 513,000 |
| Шар виключення нейронів (Dropout) | (None, 1000) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 40,040 |

Таблиця 4.2 - Архітектура CNN моделі.

| Тип шару | Вихідна форма | Кількість параметрів |
|---|------------------------|----------------------|
| Згортковий тривимірний шар (Conv3D) | (None, 28, 28, 28, 26) | 728 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 14, 14, 14, 26) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 12, 12, 12, 51) | 35,853 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 6, 6, 6, 51) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 4, 4, 4, 102) | 140,556 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 2, 2, 2, 102) | 0 |
| Шар згортання в один вектор (Flatten) | (None, 816) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 1000) | 817,000 |
| Шар виключення нейронів (Dropout) | (None, 1000) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 40,040 |

Архітектура побудована таким чином що моделі виокремлюють різноманітні характеристики з вхідних даних та передають їх на повнозв'язну мережу для класифікації. В ході експерименту, ці шари мережі не будуть відрізнятися між порівнюваними моделями.

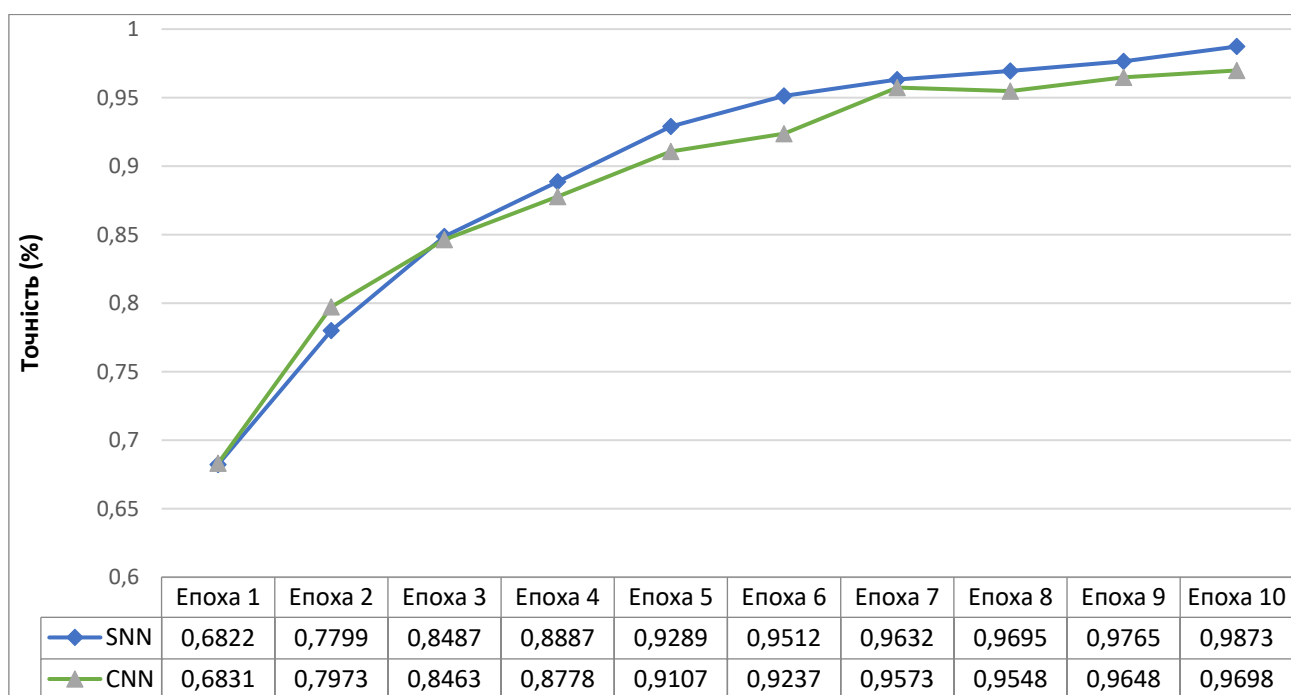


Рисунок 4.2 - Порівняння точності моделей на валідаційних даних (більше - краще).

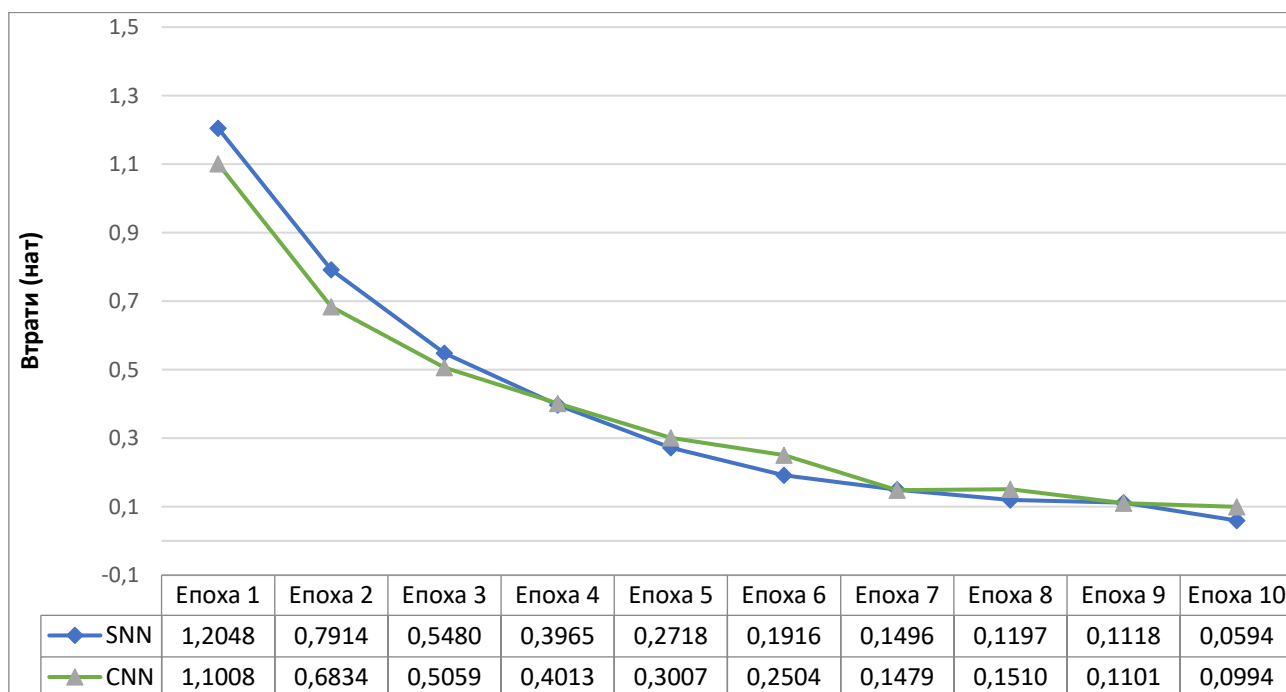


Рисунок 4.3 - Порівняння втрат моделей на валідаційних даних (менше - краще).

Навчивши декілька моделей, з повнозв'язними шарами по 1000 та 40 нейронів (40 це кількість класів в навчальній вибірці) [46], можна побачити (див. рис. 4.2, 4.3) що точність мереж майже не відрізняється. Запропонована модель хоч і має на 151470 параметрів менше, показує більш точний результат з різницею в 2%.

З цього робимо висновок, що повнозв'язний шар на 1000 елементів треба зменшити. Тому для проведення інших експериментів в подальшому будемо використовувати повнозв'язний шар на 500 нейронів, тим самим імітуючи вузьке місце класифікації для проходу якого треба більш чітко виділяти ключові ознаки зображень.

Окрім аналізу архітектур та точності, дослідження охоплює оптимізаційні стратегії. Запропоновані підходи включають динамічне видалення слабких зв'язків та створення нових, що підвищує продуктивність мережі та зменшує обчислювальні витрати. Такі оптимізації сприяють підвищенню узагальнюючої здатності моделей і забезпечують більш ефективне навчання.

Для експериментальної перевірки підготовлено шість моделей з різними конфігураціями:

- 3D_CNN_Vox - згорткова мережа для аналізу вокселів;
- 3D_CNN_Mesh - згорткова мережа для аналізу сіток;
- 3D_SNN_Vox – запропонована топологія для класифікації вокселів без оптимізації;
- 3D_SNN_Mesh - запропонована топологія для класифікації сітки без оптимізації;
- 3D_SNN_O_Vox - оптимізована топологія для класифікації вокселів;
- 3D_SNN_O_Mesh - оптимізована топологія для класифікації сітки.

Для вивчення впливу масштабування було створено п'ять варіантів кожної моделі, з поступовим збільшенням розмірності. Це дозволило простежити тенденції до перенавчання або недонавчання при збільшенні параметрів моделі та виявити граничні конфігурації, які забезпечують найкращі результати.

Кожен варіант моделі навчався 10 разів з подальшим усередненням результатів для мінімізації впливу початкових вагових розподілів. Такий підхід дозволяє забезпечити репрезентативність та стабільність експериментальних даних. Додатково було проаналізовано вплив алгоритмів оптимізації на швидкість збіжності моделей та їх ефективність при роботі з великими обсягами даних.

Оцінка продуктивності здійснювалася шляхом попарного порівняння моделей у двох групах.

Для вокселів:

- 3D_CNN_Vox;
- 3D_SNN_Vox;
- 3D_SNN_O_Vox.

Для сіток:

- 3D_CNN_Mesh;
- 3D_SNN_Mesh;
- 3D_SNN_O_Mesh.

Основні критерії оцінювання включали:

- точність класифікації;
- швидкість збіжності під час навчання;
- кількість навчених параметрів мережі;
- ефективність оптимізаційних алгоритмів;
- масштабованість і схильність до перенавчання;
- продуктивність на великих наборах даних.

Запропонована методологія дозволяє комплексно оцінити продуктивність, точність і адаптивність нових архітектур класифікаторів у порівнянні з класичними підходами. Очікується, що результати експериментів підтвердять переваги оптимізованих топологій щодо узагальнюючої здатності та ефективності обчислень.

Результати аналізу також дозволять визначити критичні параметри, що впливають на стійкість і масштабованість моделей. Це має практичне значення для розробки інтелектуальних систем аналізу тривимірних даних у таких галузях, як комп'ютерний зір, медична діагностика, промислова автоматизація та системи моніторингу.

Таким чином, дослідження робить внесок у розвиток методів і алгоритмів обробки тривимірних даних, підкреслюючи важливість інноваційних архітектур і адаптивних оптимізацій для покращення точності та ефективності класифікації.

4.3 Результати експериментів

Як зазначалося раніше, проведено 2 експерименти по класифікації тривимірних даних представлених у вигляді вокселів та сіток.

4.3.1 Класифікація воксельних даних

Після проведення навчання та тестування 15 моделей, отримані дані були консолідовані у вигляді графічних залежностей (додаток Г), що стали основою для систематичного аналізу результатів. Ці графіки дають змогу відобразити динаміку змін точності моделей на різних етапах навчання та валідації, а також оцінити ефективність використаних архітектур і алгоритмів оптимізації.

CNN

Аналіз архітектури CNN_model_1 (див. табл. В.1) демонструє, що ця модель включає шари з обмеженою кількістю параметрів (336 у першому шарі Conv3D і 8450 у другому), що призводить до значного спрощення обчислювальної складності. Вихідна форма даних поступово зменшується від (28, 28, 28, 12) до (2, 2, 2, 51) на останньому шарі MaxPooling3D, що вказує на втрату інформативності ознак на етапах згортки. Така архітектура забезпечує точність на рівні 70% (див. рис. Г.3) на етапі валідації, що є недостатнім для ефективної класифікації через обмежену здатність до узагальнення ознак.

У порівнянні, архітектури CNN_model_2 (див. табл. В.2), CNN_model_3 (див. табл. В.3) і CNN_model_4 (див. табл. В.4) показали значно вищу точність, досягаючи 96% (див. рис. Г.3) на етапі валідації. Наприклад, model_3 використовує більше фільтрів на кожному шарі (наприклад, 106,106 параметрів у третьому шарі Conv3D), що дозволяє зберігати більше просторових і текстурних ознак на кожному етапі обчислення. Зменшення вихідної форми до (2, 2, 2, 102) свідчить про кращий баланс між зменшенням розмірності та збереженням ключових ознак. Додаткові методи регуляризації, такі як Dropout, а також щільні шари (наприклад, Dense з 308,500 параметрами), забезпечують стабільність і точність під час навчання.

Однак архітектура CNN_model_5 (див. табл. В.5) з кількістю 2,269,760 параметрів у п'ятому шарі Conv3D показала ознаки перенавчання, досягаючи 95% точності (див. рис. Г.3), але втрачаючи здатність до узагальнення на

тестових наборах даних (див. рис. Г.1). Надмірна кількість параметрів і вихідна форма (1, 1, 1, 410) створюють умови для "запам'ятовування" навчальних даних замість їхнього узагальнення. Це підкреслює необхідність оптимізації шляхом видалення надлишкових параметрів і регуляризації.

Модель CNN_model_4, яка використовує більш збалансовану архітектуру, забезпечує найбільш оптимальні результати. Її вихідна форма (2, 2, 2, 205) і параметри, що становлять 564,775 у ключових шарах, дозволяють досягати високої точності без ознак перенавчання.

SNN

У результаті навчання та тестування моделей SNN (Sparse Neural Networks), отримані дані було систематизовано у вигляді діаграм та графіків, що ілюструють динаміку змін точності та втрат на етапах навчання і валідації. Проведений аналіз дозволяє зробити висновки щодо продуктивності різних архітектур, їхньої здатності до узагальнення ознак та ефективності використання обчислювальних ресурсів.

Архітектура SNN_model_1 (див. табл. В.6) відзначається компактністю, маючи 294,497 параметрів і розмір моделі 1,12 МБ. За 10 епох навчання модель досягає точності валідації 92,87% (див. рис. Г.9) при зниженні втрат до 0,273 (див. рис. Г.10). Хоча зростання точності відносно повільне, така структура демонструє стабільність і низьку схильність до перенавчання, що робить її придатною для задач з обмеженими обчислювальними ресурсами.

Модель SNN_model_2 (див. табл. В.7), яка містить 429,808 параметрів (1,64 МБ), демонструє значне покращення, досягаючи точності валідації 97,07% (див. рис. Г.9) і втрат 0,123 (див. рис. Г.10). Завдяки більшій кількості параметрів, ця архітектура має поліпшену здатність до узагальнення. Оптимальна структура цієї моделі забезпечує баланс між продуктивністю та стійкістю до перенавчання, навіть без використання спеціалізованих методів регуляризації.

Архітектури SNN_model_3 (див. табл. В.8) та SNN_model_4 (див. табл. В.9) демонструють ще вищі результати. Зокрема, SNN_model_4, яка включає 829,448 параметрів (3,16 МБ), досягає точності валідації 98,47% (див. рис. Г.9) зі зменшенням втрат до 0,072 (див. рис. Г.10). Її збалансована архітектура дозволяє ефективно обробляти складні задачі класифікації, забезпечуючи високу стабільність і точність. Зменшення ризику перенавчання досягається завдяки особливостям топології SNN, які мінімізують перенасичення моделі даними.

Найкращі результати демонструє SNN_model_5 (див. табл. В.10), яка містить 3,287,950 параметрів (12,54 МБ) і досягає точності валідації 99,17% (див. рис. Г.9) зі зменшенням втрат до 0,029 (див. рис. Г.10). Велика кількість параметрів цієї моделі дозволяє виділяти високорівневі ознаки, однак це супроводжується значними вимогами до обчислювальних ресурсів. Незважаючи на складність, архітектура SNN демонструє вбудовану стійкість до перенавчання, що зумовлено специфікою її структурної організації.

Загальний аналіз моделей SNN підкреслює ключову роль вибору архітектури для досягнення оптимального співвідношення між кількістю параметрів, обчислювальними ресурсами та продуктивністю. Компактні моделі є ефективними для задач із низькими вимогами до ресурсів, тоді як складніші архітектури забезпечують вищу точність і здатність до узагальнення, але вимагають більше ресурсів для реалізації.

SNNO

У результаті навчання та тестування моделей SNNO (Sparse Neural Networks Optimized), результати було систематизовано у вигляді діаграм та графіків, які демонструють динаміку змін точності, втрат та компресії параметрів під час навчання й валідації. Головною особливістю цих моделей є механізм оптимізації, що активується з 4-ї епохи, створюючи нові ваги та видаляючи надлишкові. Це дозволяє досягти високої продуктивності при суттєвому зменшенні кількості параметрів.

Архітектура SNNO_model_1 (див. табл. В.6) вирізняється компактністю, маючи 294,497 параметрів на початку навчання, які після оптимізації зменшуються до 130,144 (див. рис. 4.6). Розмір моделі зменшується з 1,12 МБ до 0,50 МБ. Протягом 10 епох навчання ця модель досягає точності валідації 93,37% (див. рис. Г.15) при зниженні втрат до 0,256. (див. рис. Г.16) Завдяки компресії кількість обчислень значно зменшується, водночас зберігається стабільність та ефективність моделі.

Модель SNNO_model_2 (див. табл. В.7) демонструє компресію із зменшенням параметрів з 429,808 до 194,234 (див. рис. 4.6). Розмір моделі скорочується з 1,64 МБ до 0,74 МБ. Ця модель забезпечує точність валідації 96,84% (див. рис. Г.15) та втрати на рівні 0,127 (див. рис. Г.16). Механізм оптимізації дозволяє зберегти здатність до узагальнення, що значно підвищує її ефективність.

Архітектура SNNO_model_3 (див. табл. В.8) зменшує параметри з 606,207 до 279,298 (див. рис. 4.6), а розмір моделі — з 2,31 МБ до 1,07 МБ. Модель демонструє точність валідації 98,17% (див. рис. Г.15) при втраті 0,079 (див. рис. Г.16), що свідчить про її здатність до ефективного узагальнення навіть у випадках значної компресії.

Модель SNNO_model_4 (див. табл. В.9) зменшує кількість параметрів із 829,448 до 388,419 (див. рис. 4.6). Розмір моделі скорочується з 3,16 МБ до 1,48 МБ. Після 10 епох навчання модель досягає точності валідації 98,41% та втрат на рівні 0,067. Адаптивне створення нових ваг і видалення зайвих дозволяє зберігати високу стабільність і точність.

Модель SNNO_model_5 (див. табл. В.10), із початковою кількістю параметрів 3,287,950, після оптимізації зменшує їх до 687,218 (див. рис. 4.6). Розмір моделі скорочується з 12,54 МБ до 2,62 МБ. За 10 епох навчання ця модель досягає точності валідації 99,02% (див. рис. Г.15) при втраті 0,035 (див.

рис. Г.16). Така продуктивність підтверджує здатність SNNO працювати з великою кількістю параметрів без втрати точності.

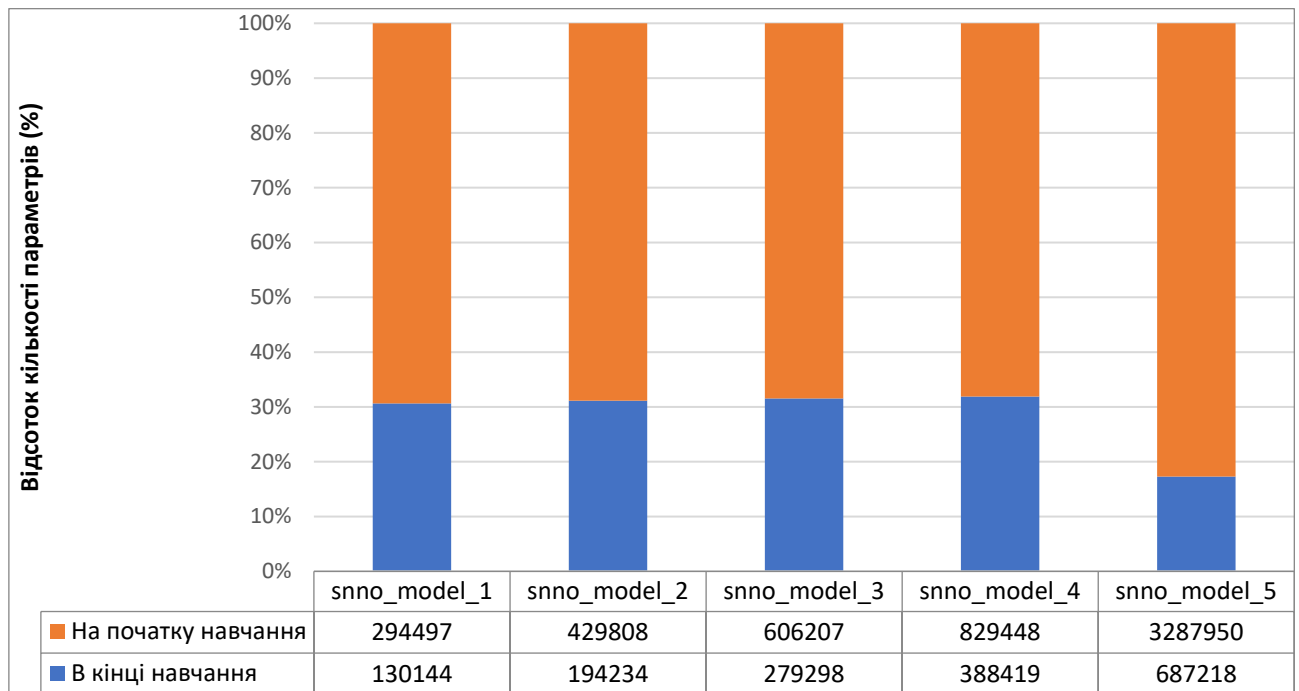


Рисунок 4.4 - Початкова та кінцева кількість параметрів при оптимізації на вокселях.

Загальний аналіз моделей SNNO підкреслює ефективність використаного методу оптимізації. Він дозволяє значно зменшувати кількість параметрів і обчислювальних ресурсів, не погіршуючи точності моделі. Це робить SNNO особливо привабливими для використання в задачах з обмеженими ресурсами, забезпечуючи високу продуктивність та адаптивність до узагальнення ознак.

4.3.2 Класифікація даних в вигляді сіток

CNN

У рамках нового експерименту було проведено навчання та тестування моделей CNN (Convolutional Neural Networks), використовуючи дані у форматі вершин сіток (mesh data). Результати роботи моделей систематизовано у вигляді діаграм та графіків, які демонструють зміну точності, втрат та інших показників на різних етапах навчання та валідації. Експеримент виявив, що даний тип моделей демонструє обмежену ефективність у роботі з вершинами сіток, і

збільшення їх розміру не забезпечує значного покращення продуктивності, а в деяких випадках навіть погіршує результати.

Архітектура CNN_model_1 (див. табл. В.1) характеризується найменшою кількістю параметрів серед розглянутих моделей — 269,179, що відповідає розміру 1,03 МБ. За 10 епох навчання модель досягає точності валідації 60,25% (див. рис. Г.21) при втраті 2,0565 (див. рис. Г.22). Протягом навчання спостерігається повільне зростання точності, що свідчить про обмежену здатність моделі до узагальнення ознак у даному типі даних. Водночас процес втрат стабілізується, але залишається відносно високим.

Модель CNN_model_2 (див. табл. В.2) має більшу кількість параметрів — 471,227 (1,80 МБ), що забезпечує покращення продуктивності. Точність валідації досягає 79,02% (див. рис. Г.21), а втрати знижуються до 0,1268 (див. рис. Г.22). Однак, незважаючи на покращення точності, результати все ще залишаються недостатніми для ефективного вирішення складних задач класифікації.

CNN_model_3 (див. табл. В.3) демонструє результати, подібні до cnn_model_2: за наявності 605,677 параметрів (2,31 МБ) модель досягає точності валідації 79,01% (див. рис. Г.21) і втрат 0,1402 (див. рис. Г.22). Незважаючи на більшу кількість параметрів, її продуктивність не перевищує cnn_model_2. Це свідчить про те, що збільшення кількості параметрів не завжди призводить до покращення результатів, а також може бути марним у задачах із цим типом даних.

Модель CNN_model_4 (див. табл. В.4) із 864,952 параметрами (3,30 МБ) досягає точності валідації 79,01% (див. рис. Г.21) при втраті 0,1302 (див. рис. Г.22). Архітектура моделі забезпечує стабільний процес навчання, але знову ж таки не демонструє суттєвого покращення точності у порівнянні з CNN_model_3. Це підтверджує, що збільшення розміру моделі понад певний поріг не дає додаткових переваг.

Найбільшу кількість параметрів має CNN_model_5 — 3,237,212 (12,35 МБ) (див. табл. В.5). Ця модель досягає точності валідації 78,03% (див. рис. Г.21) при втраті 0,2521 (див. рис. Г.22). Незважаючи на значну обчислювальну складність, її продуктивність є навіть нижчою, ніж у менших моделей. Це свідчить про те, що надлишкова кількість параметрів може погіршувати продуктивність і не адаптується належним чином до даних у форматі сіток.

Загальний аналіз показує, що моделі CNN мають обмежену здатність до роботи з даними у форматі сіток. Збільшення кількості параметрів не призводить до суттєвого покращення результатів і може навіть погіршувати продуктивність. Цей результат підкреслює необхідність розробки нових підходів або архітектур для ефективної роботи з даними такого типу.

SNN

Також було проведено навчання та тестування моделей SNN (Sparse Neural Networks) з використанням даних у форматі сіток (mesh data). Результати експерименту представлені у вигляді таблиць і графіків, які ілюструють динаміку змін точності, втрат та інших ключових показників на різних етапах навчання і валідації. Мета дослідження полягала у визначенні здатності моделей SNN адаптуватися до специфіки даного типу даних та оцінці їх ефективності порівняно з іншими архітектурами.

Архітектура SNN_model_1 (див. табл. В.6) вирізняється найменшою кількістю параметрів серед розглянутих моделей — 294,497, що відповідає розміру 1,12 МБ. Після 10 епох навчання модель досягає точності валідації 83,05% (див. рис. Г.27) при втраті 0,3502 (див. рис. Г.28). Проте, поступове зростання точності протягом навчання свідчить про певні обмеження цієї архітектури у здатності до ефективного узагальнення ознак у даному форматі даних.

Модель SNN_model_2 (див. табл. В.7) із 429,808 параметрами (1,64 МБ) демонструє помітне покращення, досягаючи точності валідації 88,50% (див. рис.

Г.27) при втраті 0,1503 (див. рис. Г.28). Завдяки збалансованій структурі, ця модель демонструє стабільну динаміку зростання точності та зменшення втрат, особливо після п'ятої епохи, що свідчить про її добру адаптацію до нового типу даних.

SNN_model_3 (див. табл. В.8), яка містить 606,207 параметрів (2,31 МБ), забезпечує суттєве покращення порівняно з попередньою моделлю, досягаючи точності валідації 92,46% (див. рис. Г.27) при втраті 0,1102 (див. рис. Г.28). Зростання точності та зменшення втрат демонструють високу ефективність цієї моделі для роботи з даними у форматі сіток.

Модель SNN_model_4 (див. табл. В.9) із 829,448 параметрами (3,16 МБ) демонструє найкращі результати серед представлених архітектур. Вона досягає точності валідації 94,93% (див. рис. Г.27) при втраті 0,0902 (див. рис. Г.28). Стабільність навчання спостерігається вже після шостої епохи, що підкреслює її високу здатність до узагальнення ознак та адаптації до складних даних.

Найбільша за кількістю параметрів модель, SNN_model_5 (3,287,950 параметрів, 12,54 МБ) (див. табл. В.10), досягає точності валідації 94,30% (див. рис. Г.27) при втраті 0,0354 (див. рис. Г.28). Незважаючи на значну кількість параметрів і обчислювальну складність, продуктивність цієї моделі вказує на її здатність ефективно працювати зі складними ознаками. Проте зростання параметрів після певного рівня демонструє зменшення ефективності щодо зростання точності.

Загальний аналіз свідчить, що моделі SNN добре адаптуються до роботи з даними у форматі сіток. Вони демонструють стабільне зростання точності та зменшення втрат із кожною епохою, проте продуктивність значно залежить від балансу між складністю архітектури та обчислювальними ресурсами. Моделі SNN_model_4 і SNN_model_5 показали найкращі результати, тоді як SNN_model_1 та SNN_model_2 залишаються ефективними для задач із обмеженими обчислювальними можливостями.

SNNO

У межах нового експерименту було проведено навчання та тестування оптимізованих моделей SNNO (Sparse Neural Networks Optimized), спрямоване на оцінку їхньої здатності до компресії параметрів і збереження високої продуктивності при роботі з даними у форматі сіток (mesh data). Результати експерименту представлені у вигляді таблиць і графіків, які відображають динаміку змін кількості параметрів, точності, втрат та інших ключових показників на різних етапах навчання і валідації.

Архітектура `snno_model_1` (див. табл. В.6) включає 294,497 параметрів, які після компресії зменшуються до 173,207 (див. рис. 4.7). Розмір моделі скорочується з 1,12 МБ до 0,66 МБ. Після 10 епох навчання модель демонструє точність валідації на рівні 84,03% (див. рис. Г.33) при втраті 0,2816 (див. рис. Г.34). Незважаючи на значне зменшення кількості параметрів, модель демонструє стабільне зростання точності, що свідчить про її адаптивність до специфіки даних.

Модель `snno_model_2` (див. табл. В.7), із початковою кількістю параметрів 429,808, після оптимізації зменшує їх до 229,779 (див. рис. 4.7). Розмір моделі скорочується з 1,64 МБ до 0,88 МБ. Після 10 епох навчання модель досягає точності валідації 87,16% (див. рис. Г.33) при втраті 0,1393 (див. рис. Г.34). Ефективність компресії підтверджується збереженням здатності до узагальнення ознак та стабільним зменшенням втрат.

`snno_model_3` (див. табл. В.8) демонструє ще вищий рівень компресії, зі зменшенням кількості параметрів з 606,207 до 357,566 (див. рис. 4.7). Розмір моделі знижується з 2,31 МБ до 1,36 МБ. Точність валідації після 10 епох становить 88,35% (див. рис. Г.33), при втраті 0,0869 (див. рис. Г.34). Ця архітектура ефективно поєднує оптимізацію параметрів із високою продуктивністю при аналізі даних у форматі сіток.

Модель `snno_model_4` (див. табл. В.9) має початкову кількість параметрів 829,448, яка після оптимізації зменшується до 442,169 (див. рис. 4.7). Розмір моделі скорочується з 3,16 МБ до 1,69 МБ. Точність валідації досягає 94,56% (див. рис. Г.33) при втраті 0,0744 (див. рис. Г.34) після 10 епох навчання. Завдяки збалансованій архітектурі модель забезпечує глибоке узагальнення ознак, зберігаючи стабільність і високу точність навіть при значній компресії параметрів.

`snno_model_5` (див. табл. В.10), найбільша за кількістю параметрів (3,287,950), після оптимізації зменшує їх до 1,098,343 (див. рис. 4.7). Розмір моделі скорочується з 12,54 МБ до 4,19 МБ. Точність валідації досягає 96,54% (див. рис. Г.33) при втраті 0,0372 (див. рис. Г.34). Незважаючи на суттєве зменшення параметрів, модель демонструє виняткову продуктивність і здатність працювати зі складними наборами даних.

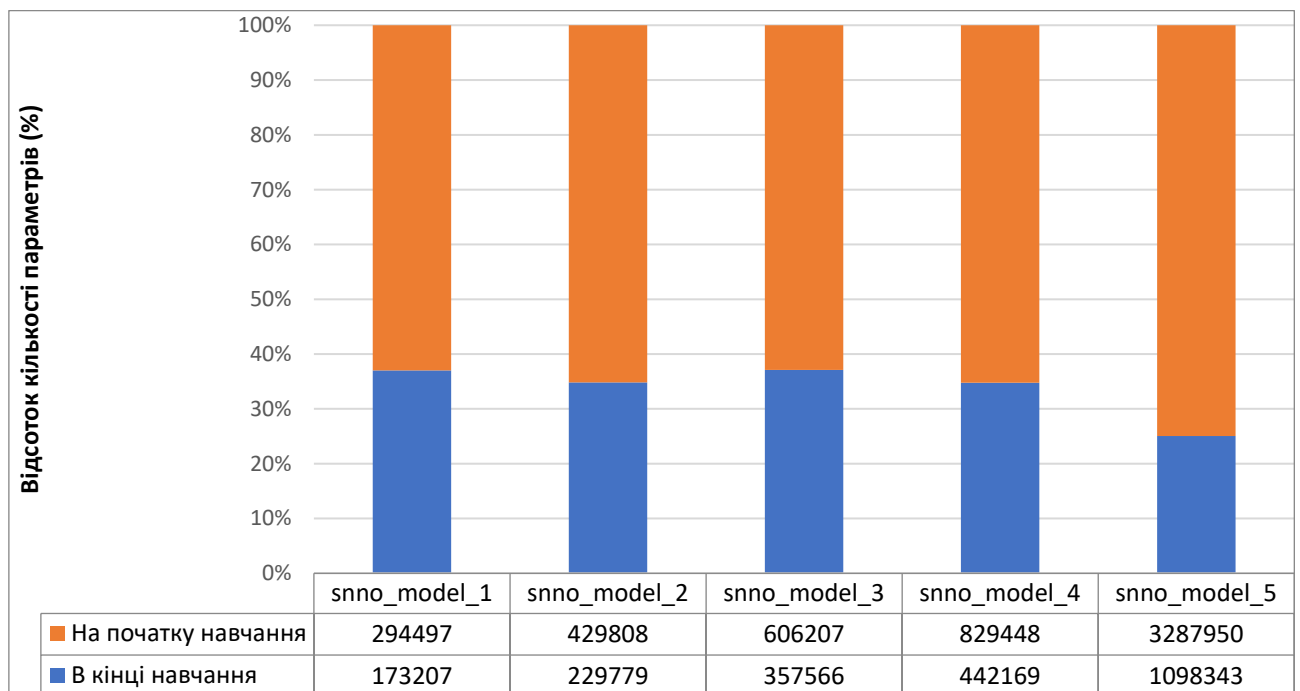


Рисунок 4.5 - Початкова та кінцева кількість параметрів при оптимізації на сітках.

Загальний аналіз підтверджує, що моделі SNNO демонструють високу ефективність у компресії параметрів, зберігаючи здатність до узагальнення та адаптації до даних у форматі сіток. Найкращі результати досягнуто за допомогою моделей `snno_model_4` і `snno_model_5`, які забезпечують

оптимальний баланс між компресією, точністю та обчислювальною ефективністю. Такі архітектури є перспективними для задач із високими вимогами до продуктивності та обмеженими обчислювальними ресурсами.

4.4 Порівняльний аналіз

Для формулювання обґрунтованих висновків щодо ефективності роботи нової топології та застосованих оптимізаторів, проведено порівняльний аналіз моделей класифікаторів [53]. Об'єктом порівняння стали моделі однакового розміру, де розмір визначається як загальна кількість параметрів, що піддаються навчанню шляхом зворотного розповсюдження помилки.

У рамках дослідження під час вирівнювання моделей за кількістю параметрів дотримувались принципу приблизної рівності, оскільки специфіка топологій накладає певні обмеження на кратність параметрів. Зокрема, обмеження можуть бути обумовлені архітектурними вимогами, такими як фіксована кількість нейронів у шарах або залежність між розмірами шарів у розширювально-звужувальних топологіях. Це забезпечує коректність порівняння, зберігаючи рівність вихідних умов для аналізу.

Відповідно до методики, були протестовані різні архітектури нейронних мереж з урахуванням таких параметрів:

- забезпечено відносно рівну кількість параметрів для різних архітектур, що дозволяє зіставляти результати без впливу диспропорції у складності моделей;
- досліджено адаптивні можливості моделей у процесі навчання з використанням різних алгоритмів оптимізації, включаючи адаптивні методи перебудови зв'язків;
- оцінювались результати моделей щодо здатності коректно класифікувати тривимірні дані при різних наборах вхідних умов.

Отримані результати дозволяють об'єктивно оцінити вплив запропонованих удосконалень на точність і продуктивність класифікації. Такий підхід є універсальним для порівняння моделей із різними топологіями та забезпечує об'єктивність аналізу навіть за умов неоднорідних параметричних обмежень.

Розглянемо розподіл кількості параметрів в моделях.

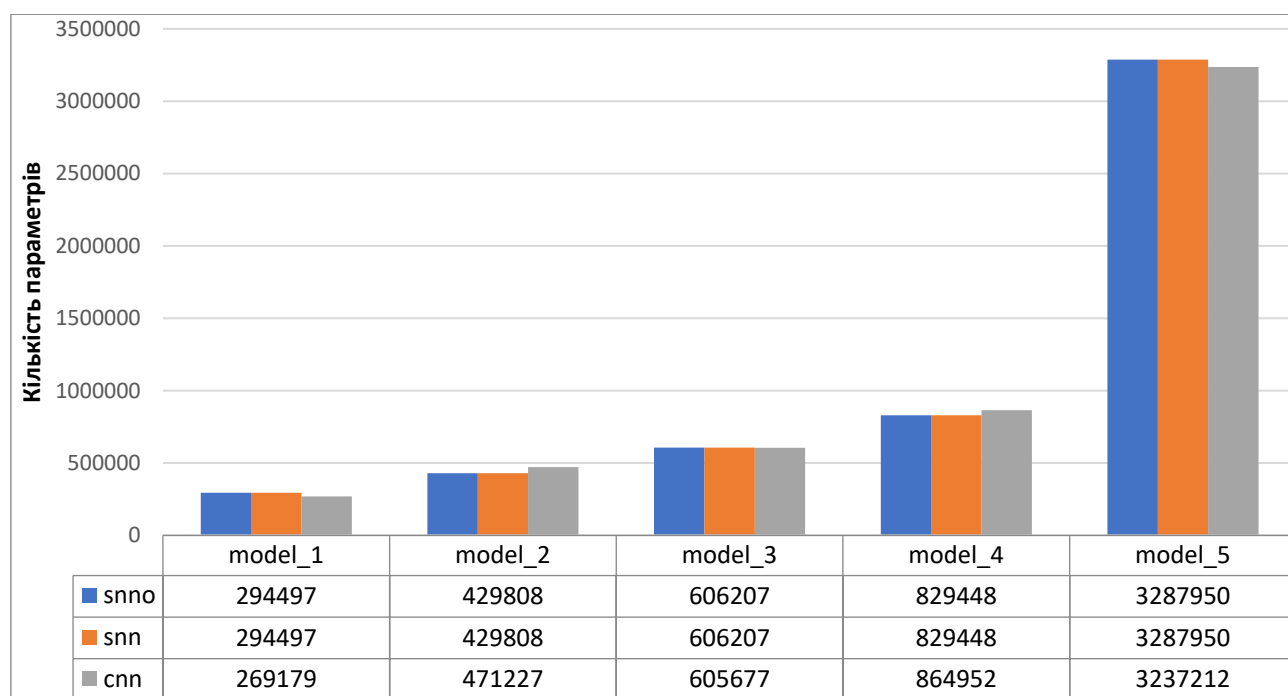


Рисунок 4.6 - Початковий розподіл моделей за кількістю параметрів.

Бачимо (див. рис. 4.8), що моделі SNN та SNNO мають однакову кількість параметрів на початку навчання. Це зумовлено тим, що архітектури цих моделей є ідентичними, за винятком використовуваних методів оптимізації. У моделі SNNO реалізовано механізми динамічного управління зв'язками, що передбачає видалення малозначущих зв'язків і створення нових під час навчання. Завдяки цьому кількість параметрів у моделях цього типу змінюється протягом навчання, що дозволяє оцінити ефективність роботи оптимізаторів на різних етапах.

Даний розподіл параметрів є універсальним для обох експериментів, оскільки структура та архітектура моделей залишаються незмінними. Важливо також зазначити, що перші чотири моделі демонструють поступове зростання кількості параметрів. Це зроблено для того, щоб дослідити, як залежить продуктивність

моделей від збільшення кількості параметрів. Зокрема, аналіз таких моделей дозволяє виявити оптимальний баланс між складністю моделі та її здатністю до узагальнення.

Окрему увагу заслуговує п'ята модель, яка має значно більшу кількість параметрів у порівнянні з попередніми. Ця модель була розроблена не лише для аналізу залежності між кількістю параметрів і точністю класифікації, але й для дослідження здатності моделей справлятися з перенавчанням. Використання таких великих моделей дозволяє оцінити, наскільки адаптивні оптимізатори здатні знижувати обчислювальну складність, зберігаючи при цьому високу точність і стійкість до перенавчання.

Тепер розглянемо інший розподіл, після навчання та зробимо висновки.

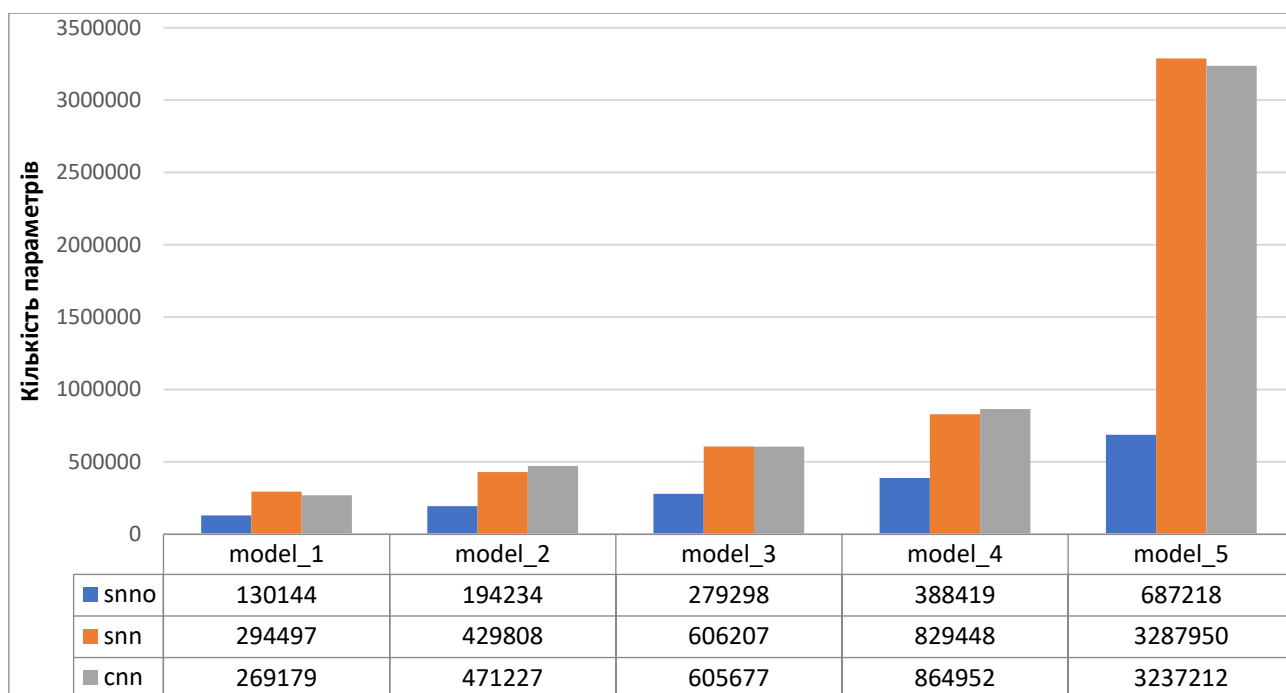


Рисунок 4.7 - Кінцевий розподіл моделей за кількістю параметрів для вокселів.

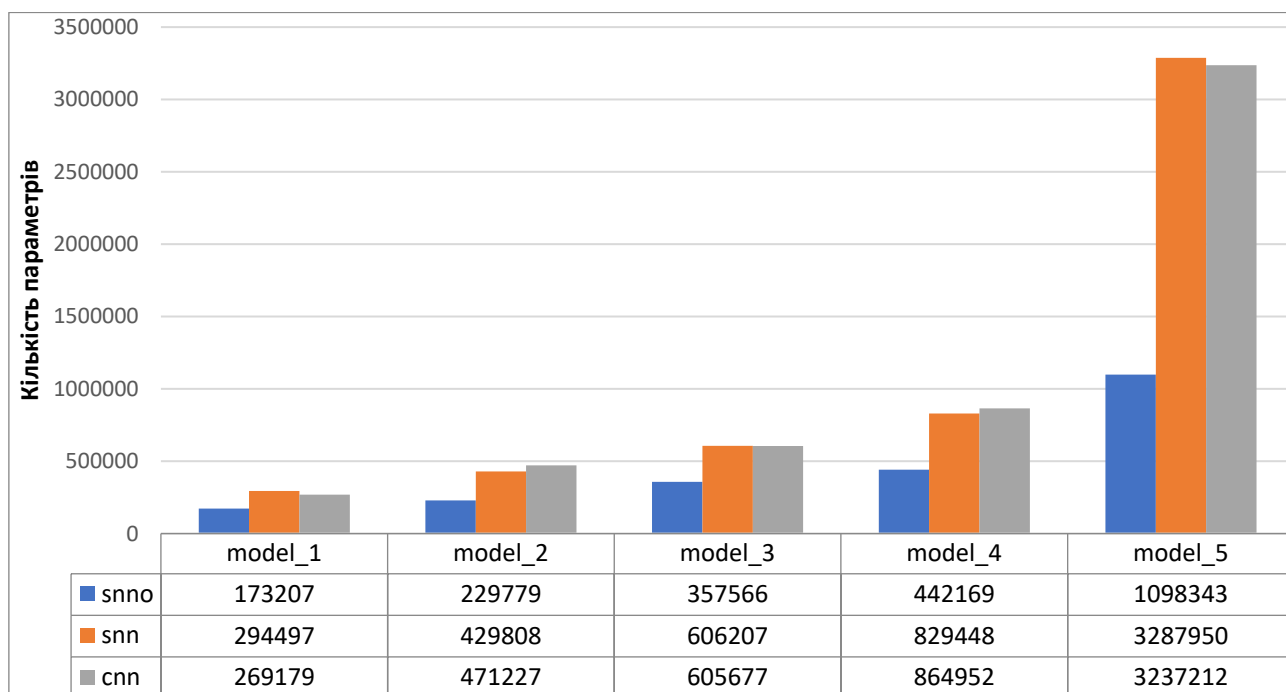


Рисунок 4.8 - Кінцевий розподіл моделей за кількістю параметрів для сіток.

Як бачимо (див. рис. 4.9, 4.10), оптимізатори демонструють позитивні результати. Зокрема, можна помітити, що при аналізі вокселів оптимізатор на перших чотирьох моделях залишає близько 30% параметрів від початкової кількості. Цей результат наочно ілюструє графік 4.3. У випадку роботи із сітками оптимізатор зберігає приблизно 35% параметрів, що свідчить про складність цього типу даних для класифікації моделлю. Сітки є структурно складнішими сутностями, оскільки вони спочатку створювались для моделювання 3D-об'єктів і дизайну, що потребує більшого обсягу параметрів для точного представлення та аналізу їхньої геометрії.

Ці спостереження дозволяють зробити припущення, що точність моделей у випадку роботи із сітками буде нижчою, адже оптимізатор зберіг більшу кількість параметрів для обробки цієї складної структури. У той час як у випадку з вокселями оптимізатор здатен зменшити кількість параметрів без значного впливу на продуктивність, що вказує на більш ефективну обробку цього типу даних. Таким чином, характер збереження параметрів моделі демонструє чутливість і адаптивність оптимізатора до типу вхідних даних, підкреслюючи

важливість використання специфічних налаштувань для забезпечення оптимальної продуктивності класифікації.

Тепер порівняємо моделі за точністю. Для цього винесемо на графік результати моделей однакового розміру та проаналізуємо їхню продуктивність. Порівняння проводитимемо виключно на основі даних із валідаційного набору, оскільки саме вони відображають здатність моделей до узагальнення, уникаючи впливу перенавчання. Всі інші результати, включаючи втрати та точність на навчальному наборі, представлені у Додатку Г.

Розпочнемо з моделей що працювали з вокселями.

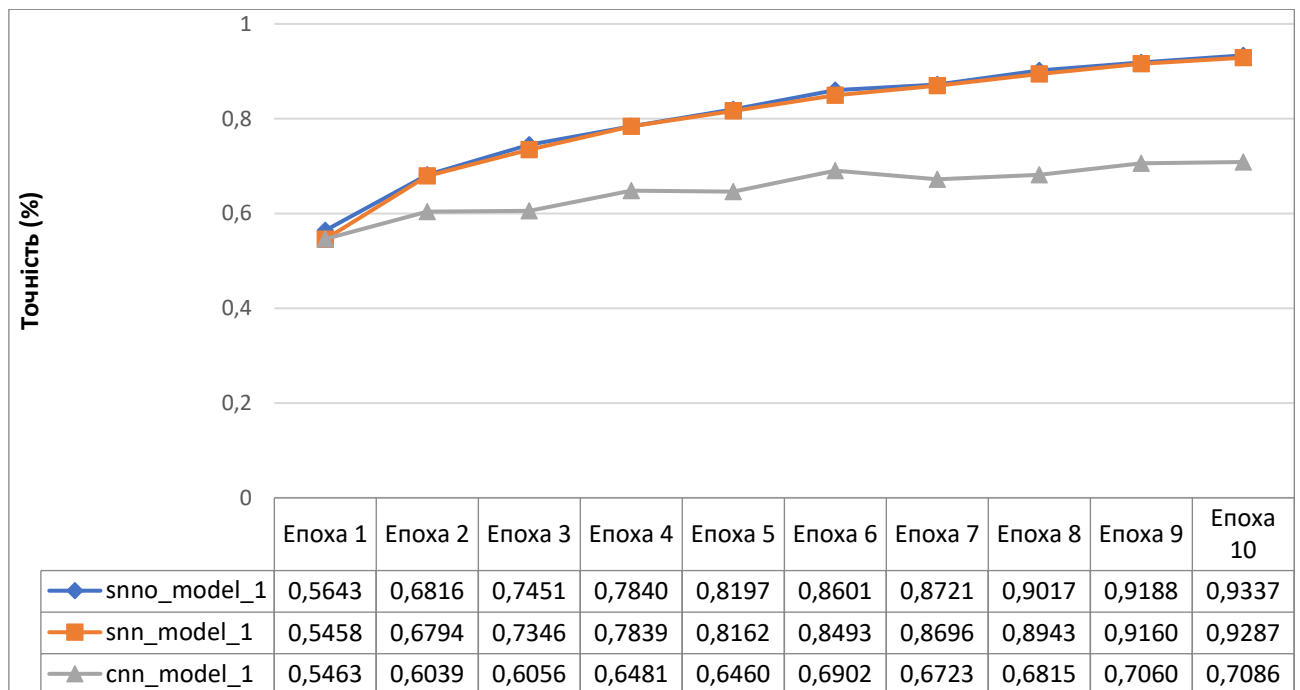


Рисунок 4.9 - Порівняння валідаційної точності моделей класу model_1 для вокселів (більше - краще).

Model_1 (див. рис. 4.11) показує, що для згорткової мережі такої кількості параметрів недостатньо для узагальнення, тоді як SNN моделі показують значно кращий результат.

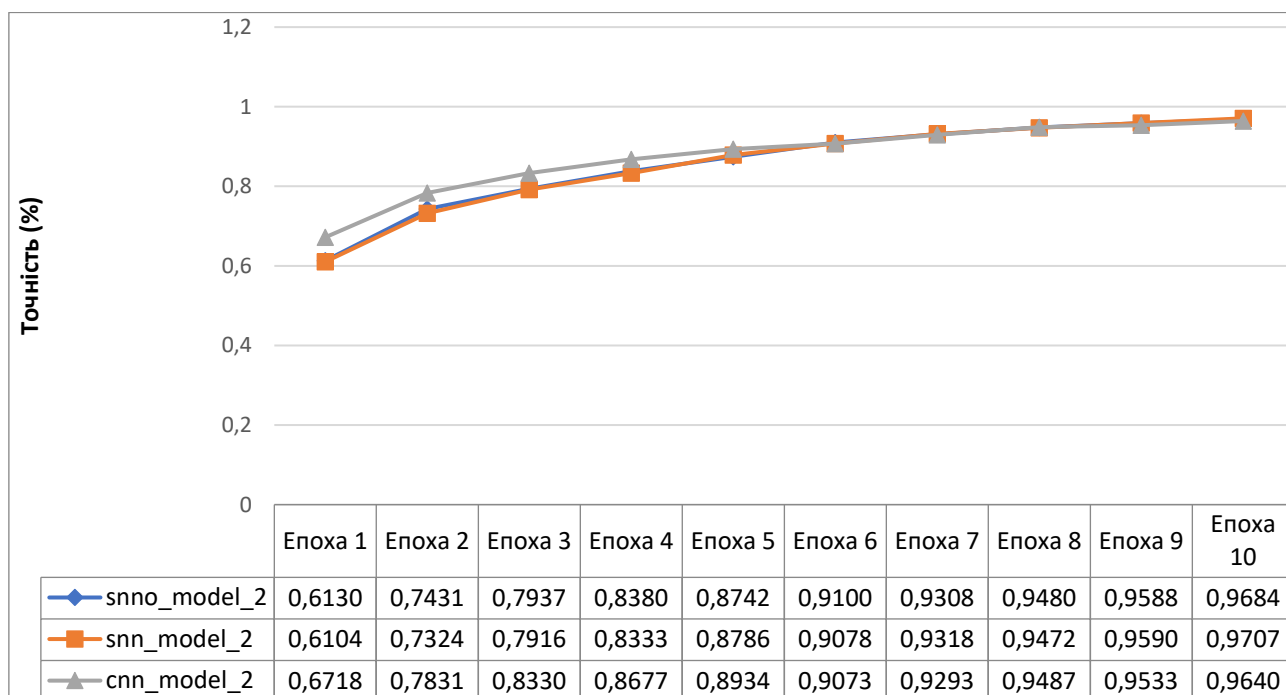


Рисунок 4.10 - Порівняння валідаційної точності моделей класу model_2 для вокселів (більше - краще).

Всього трохи більше параметрів і вже на Model_2 згорткова мережа майже повністю вирівнюється з розрідженими (див. рис. 4.12). І навіть трохи обганяє їх до 5 епохи.

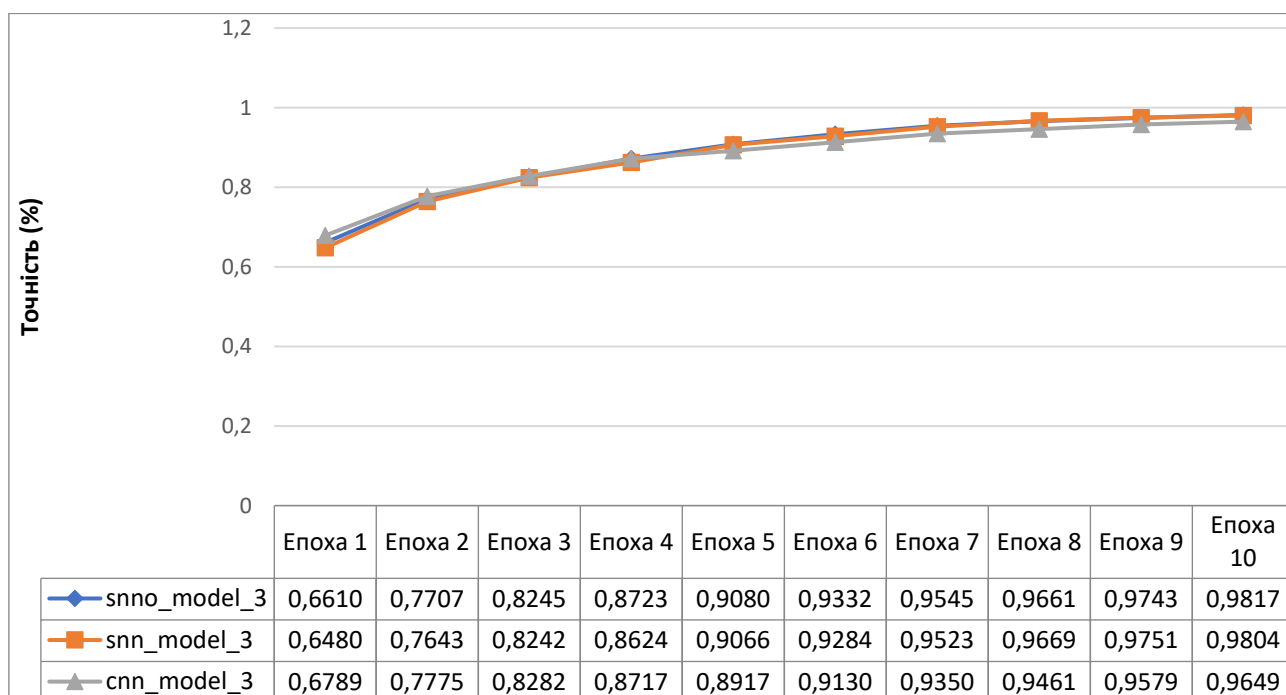


Рисунок 4.11 - Порівняння валідаційної точності моделей класу model_3 для вокселів (більше - краще).

Бачимо (див. рис. 4.13), що всі три моделі демонструють майже ідентичний результат на всіх епохах навчання. Така схожість у поведінці моделей частково пояснюється їхніми схожими механізмами роботи з даними всередині мережі. Зокрема, кожна з моделей реалізує обчислення, які передбачають множення значень у певній області на встановлені коефіцієнти, подальше сумування цих значень та передачу результату до функції активації. Цей підхід є універсальним для багатьох архітектур нейронних мереж, що і забезпечує подібність отриманих результатів.

Однак, незважаючи на подібність механізмів обчислень, природа моделей є різною. Згорткові нейронні мережі (CNN) використовують зафіксовані ядра згортки, які визначаються до навчання і залишаються незмінними. Ці ядра є основою для вилучення локальних ознак із даних, що робить CNN надзвичайно ефективними для роботи з просторовими структурами, такими як зображення або тривимірні дані.

На відміну від цього, у запропонованій топології моделі "ядра згортки" є частиною ваг шару. Це означає, що коефіцієнти, які в CNN залишаються незмінними, у новій топології можуть динамічно змінюватися під час навчання та адаптуватися до специфіки вхідних даних. Такий підхід забезпечує більшу гнучкість моделі, дозволяючи їй більш точно враховувати залежності між параметрами вхідних даних.

Таким чином, хоча графіки моделей демонструють схожу продуктивність, підходи до обробки даних у цих архітектурах суттєво відрізняються. Запропонована топологія завдяки своїй динамічності потенційно здатна перевершувати класичні згорткові моделі у випадках, коли адаптація до нових умов даних є критичною.

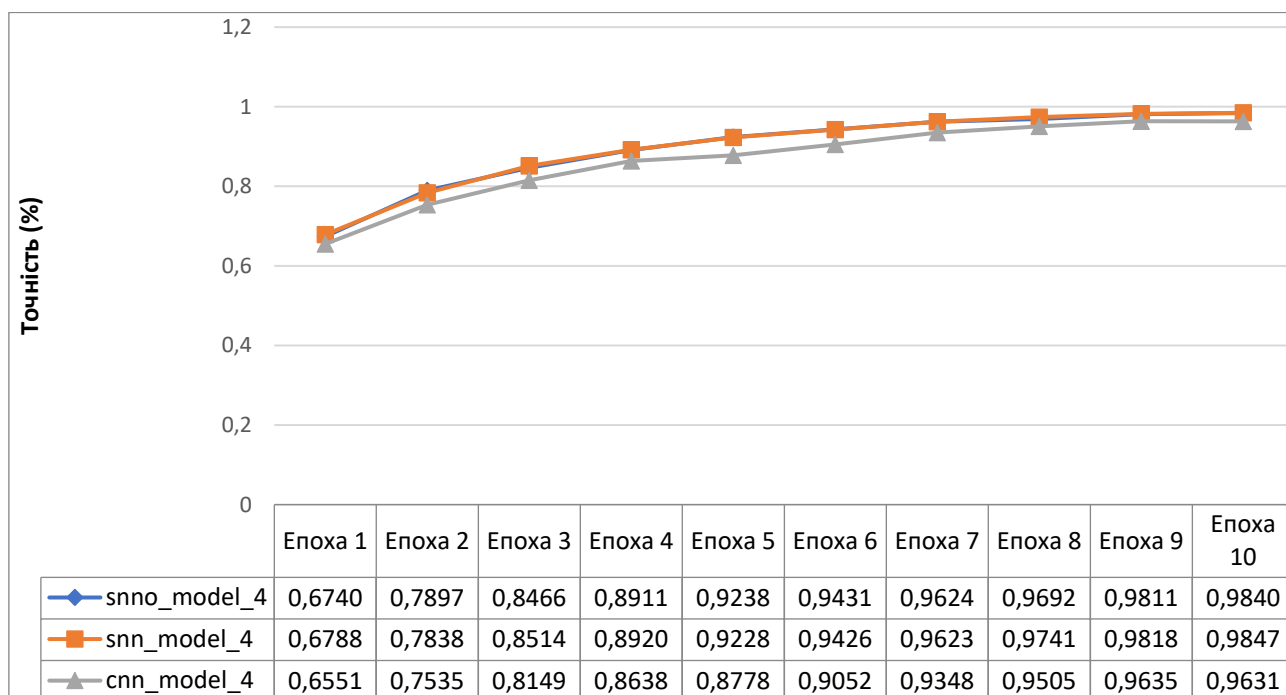


Рисунок 4.12 - Порівняння валідаційної точності моделей класу model_4 для вокселів (більше - краще).

І вже на Model_4 бачимо, що згорткова мережа почала перенавчатись (див. рис. 4.14). На даних моделях це не настільки помітно, тому є сенс розглянути Model_5.

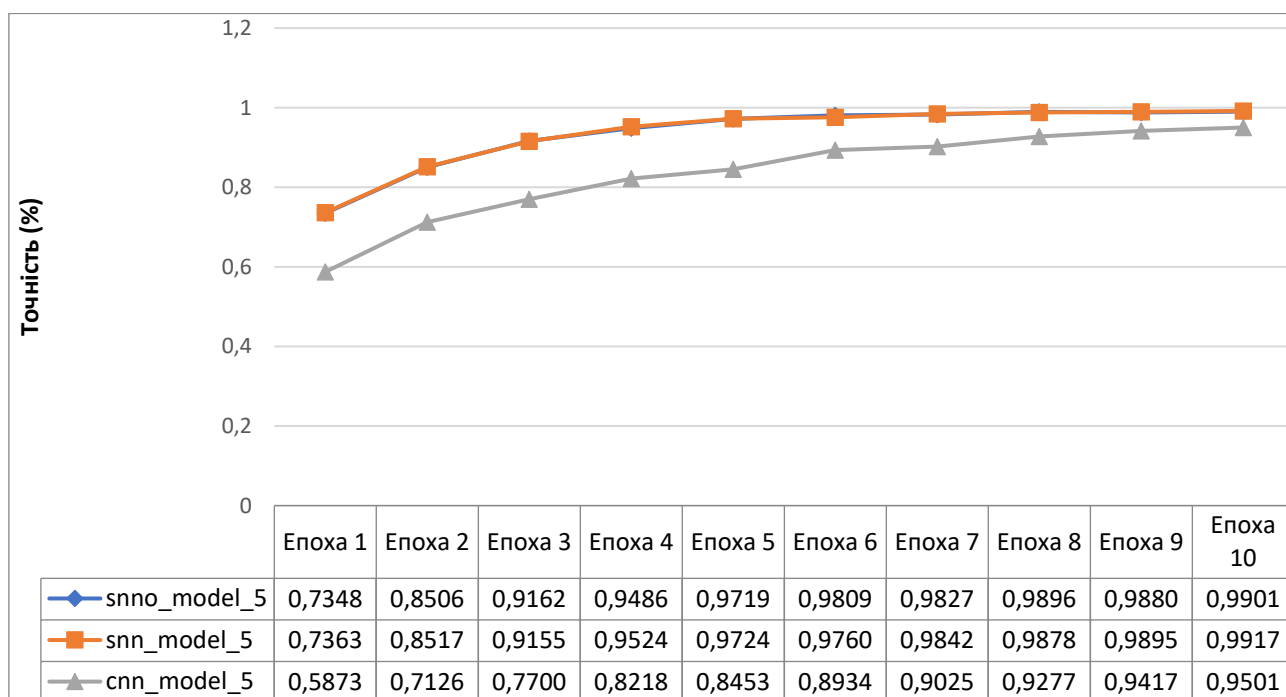


Рисунок 4.13 - Порівняння валідаційної точності моделей класу model_5 для вокселів (більше - краще).

Спостерігаємо помітну тенденцію (див. рис. 4.15), згорткова мережа демонструє значно сильніше перенавчання, тоді як розріджені моделі, навпаки, показали покращення точності приблизно на 1%. Ця різниця є важливим показником ефективності використання розрідженої топології з алгоритмом формування зв'язків.

Варто також підкреслити, що видалення ознак у розріджених моделях не мало негативного впливу на їхню точність. Попри суттєве зменшення кількості параметрів, модель зберегла свою здатність до класифікації з достатньою точністю. Цей результат демонструє ефективність видалення малозначущих зв'язків, що дозволяє оптимізувати структуру моделі без втрати її продуктивності.

Проте, аналіз також вказує на те, що вплив нових нейронних зв'язків, які створюються в рамках оптимізаторів, залишається незначним. На цьому етапі не спостерігається суттєвого покращення точності, яке можна було б безпосередньо пов'язати з новими зв'язками. Проте, їхня роль у компенсації можливих втрат точності після видалення старих зв'язків є потенційно важливою. Можна припустити, що ці нові зв'язки виконують функцію нівелювання просадок у точності, які виникають через редукцію параметрів.

Тепер розглянемо та порівняємо результати роботи мереж в класифікації сіток.

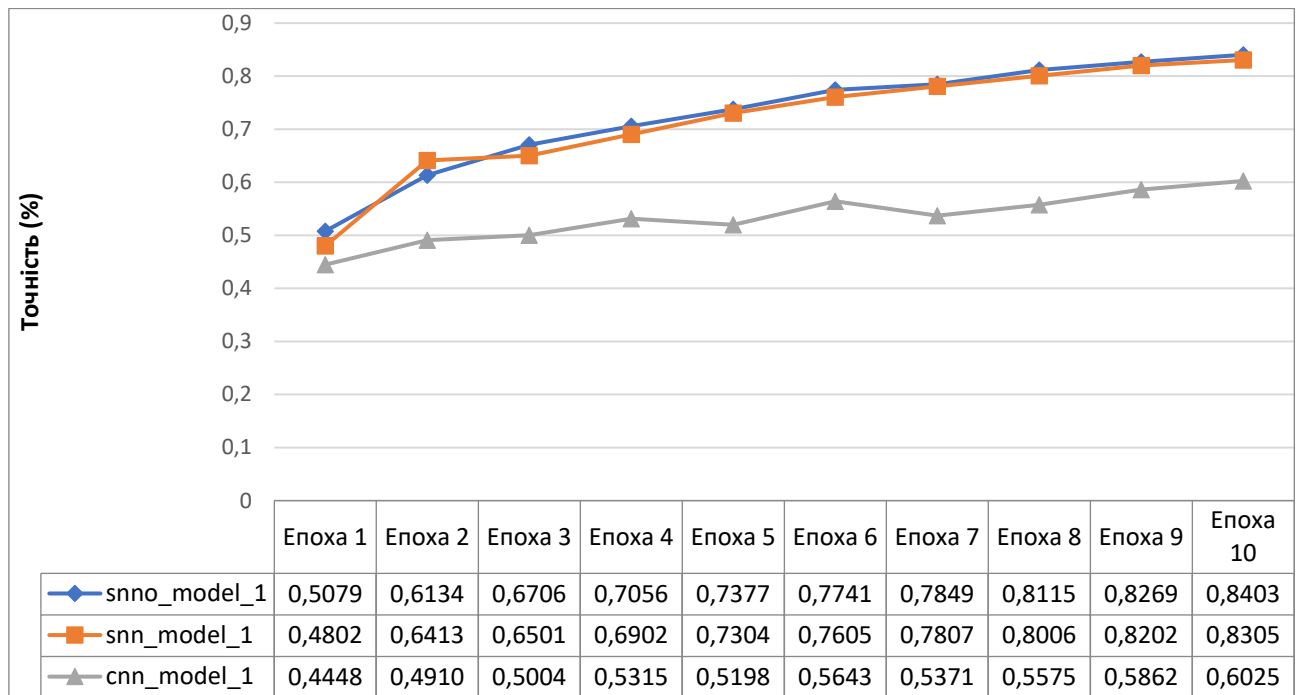


Рисунок 4.14 - Порівняння валідаційної точності моделей класу model_1 для сіток (більше - краще).

З графіку (див. рис. 4.16) видно, що згорткова мережа демонструє нижчий результат точності порівняно з розрідженими моделями. Однак, якщо звернути увагу на результати Model_1 для вокселів, можна помітити, що різниця в точності між моделями складає близько 22%, а для сіток — 23,5%. Ці схожі результати свідчать про певну закономірність, яка може бути пов'язана з характером оброблюваних даних.

Отримані показники підтверджують припущення про те, що дані, представлені у форматах сіток, є складнішими для обробки. Це пояснюється складністю геометричних залежностей, які потребують глибшого аналізу. Ускладненість даних також виявлялась раніше під час порівняння ступеня компресії параметрів, виконаної оптимізаторами.

Особливості цього порівняння дозволяють зробити висновок, що різниця в точності між моделями, особливо на початкових рівнях, може бути частково обумовлена різницею у здатності моделей враховувати ці складні просторові залежності.

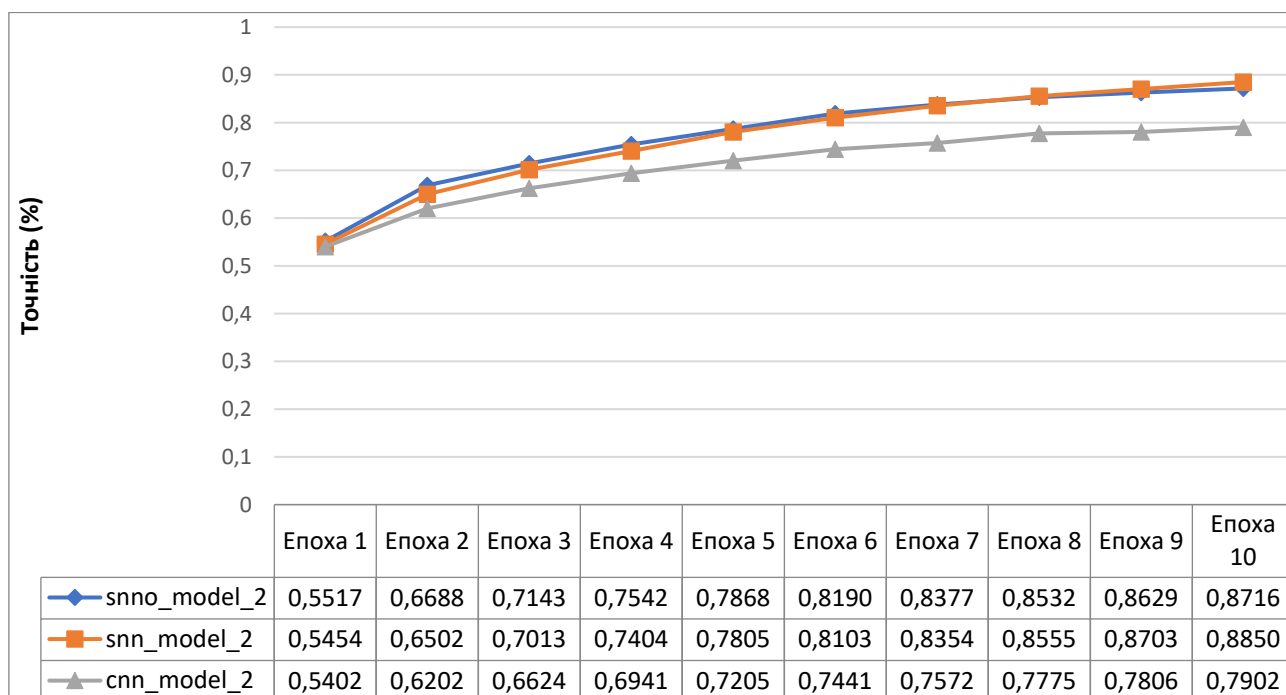


Рисунок 4.15 - Порівняння валідаційної точності моделей класу model_2 для сіток (більше - краще).

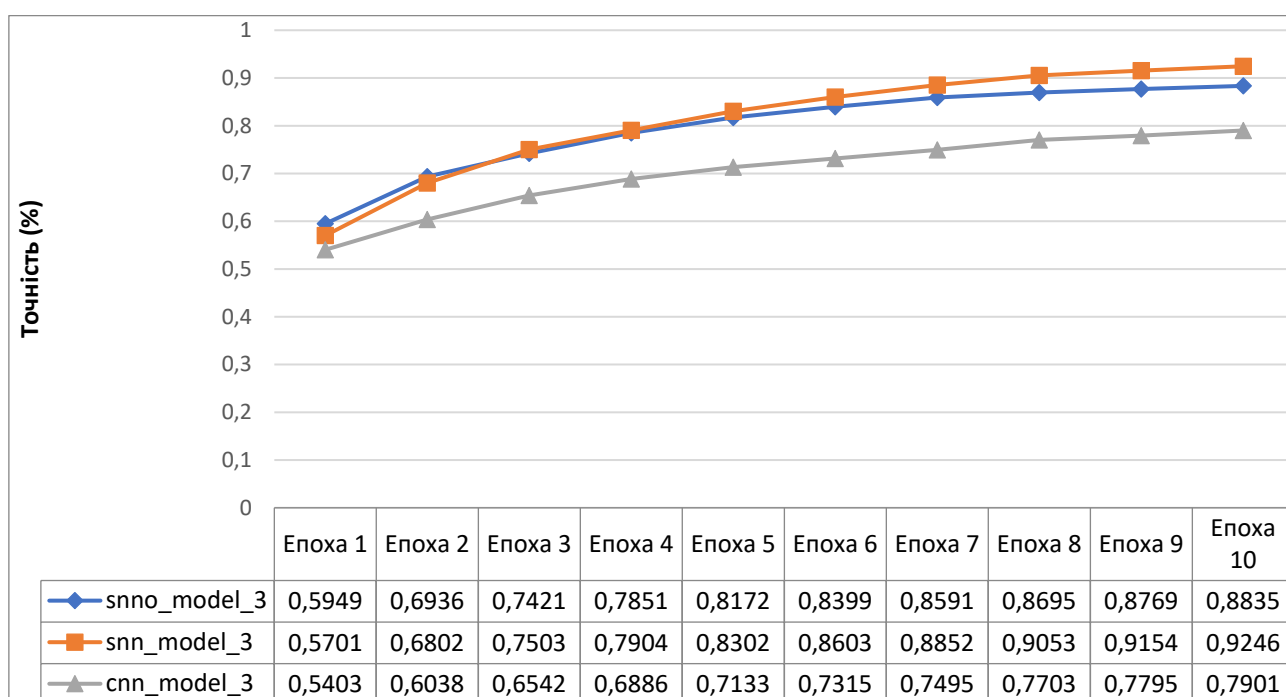


Рисунок 4.16 - Порівняння валідаційної точності моделей класу model_3 для сіток (більше - краще).

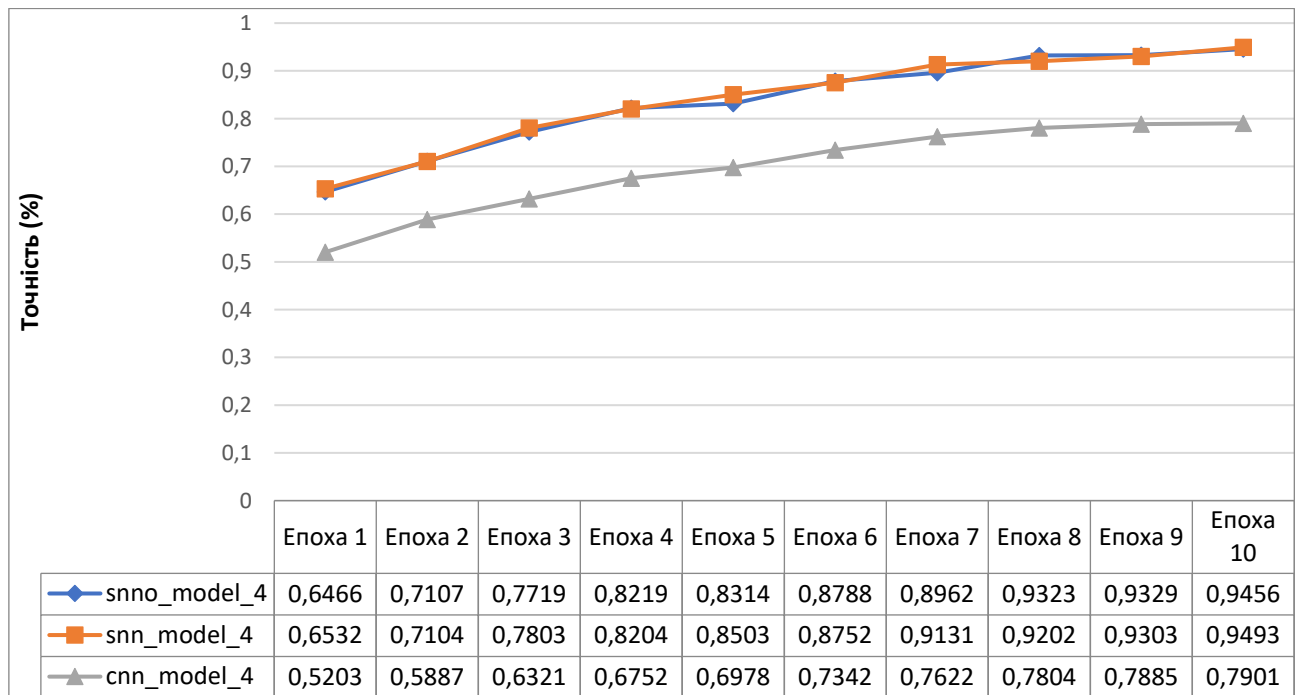


Рисунок 4.17 - Порівняння валідаційної точності моделей класу model_4 для сіток (більше - краще).

Для Model_2 (див. рис. 4.17), Model_3 (див. рис. 4.18) та Model_4 (див. рис. 4.19) спостерігається схожа поведінка моделей. Розріджені моделі поступово набирають точності на кожній ітерації, тоді як згорткова мережа досягає свого піку точності на рівні 79%. Важливо зазначити, що навіть зі збільшенням кількості параметрів точність згорткової мережі не зростає. Це може свідчити про обмеження в здатності згорткових мереж ефективно виокремлювати значущі дані для подальшого використання у тренуванні повнозв'язної частини моделі.

Такий результат підкреслює важливість саме повнозв'язної мережі для досягнення вищої точності. Збільшення її розміру або глибини може бути критичним для подальшого покращення продуктивності згорткової архітектури. Інакше кажучи, поточна архітектура згорткової мережі може стати вузьким місцем, яке обмежує її здатність до подальшого вдосконалення.

Щодо розріджених моделей, особливо цікавими є результати для Model_3. У цьому випадку модель із оптимізатором параметрів показує дещо нижчу точність у порівнянні з базовою розрідженою моделлю. Цей феномен потенційно може підтвердити припущення, що створення нових нейронних зв'язків

компенсує втрати точності, спричинені видаленням попередніх зв'язків. Іншими словами, нові зв'язки, хоч і не дають суттєвого приросту продуктивності, допомагають підтримувати стабільність моделі, компенсуючи втрати від редукції параметрів.

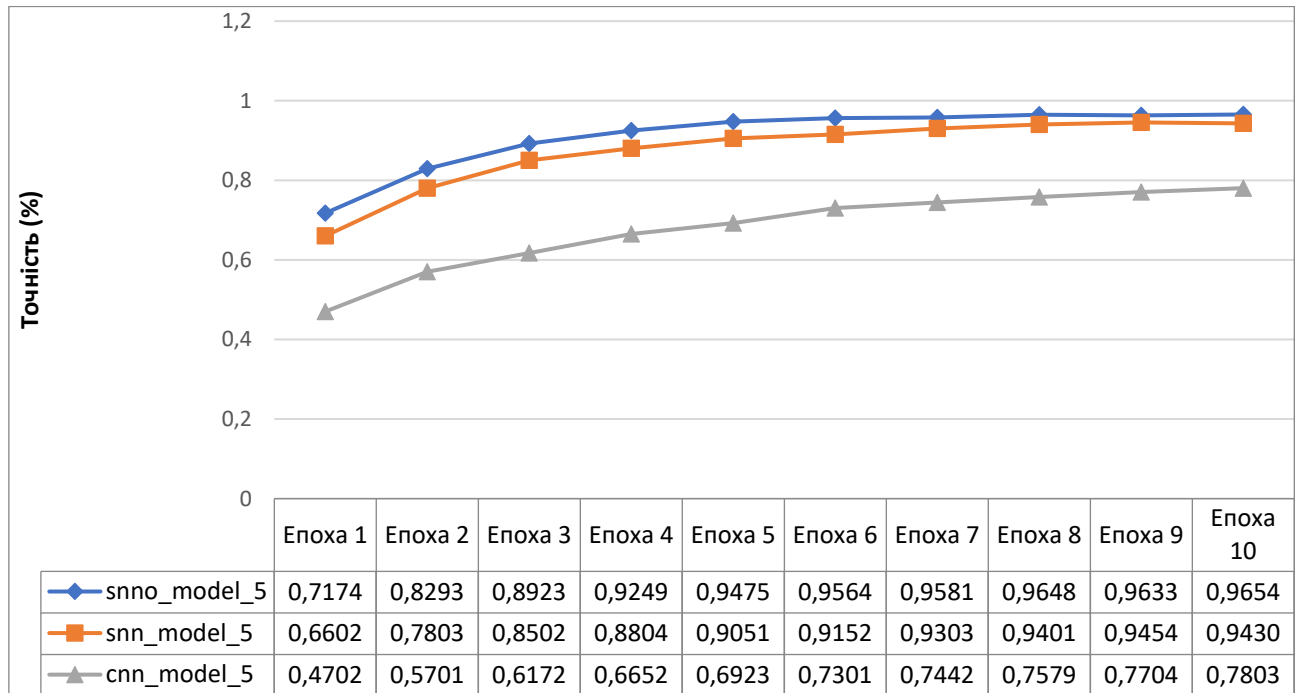


Рисунок 4.18 - Порівняння валідаційної точності моделей класу model_5 для сіток (більше - краще).

Model_5 демонструє чіткі ознаки перенавчання у згортковій моделі (див. рис. 4.20). Це особливо помітно за стабільністю точності, яка не тільки перестає зростати, але й може почати знижуватися на валідаційних даних. Така поведінка свідчить про те, що збільшення кількості параметрів у згортковій моделі не забезпечує відповідного приросту продуктивності, а натомість ускладнює узагальнення моделі на нові дані.

У свою чергу, модель із оптимізацією параметрів демонструє трохи кращі результати. Різниця у точності між розрідженою моделлю з оптимізацією та базовою розрідженою моделлю становить 2,5%, що є значущим показником у задачах класифікації. Такий приріст свідчить про ефективність механізмів оптимізації, які дозволяють динамічно управляти структурою моделі, видаляючи слабкі зв'язки та створюючи нові, що підвищує адаптивність і точність.

Цей результат також підкреслює потенціал моделей із динамічними оптимізаторами для роботи зі складними наборами даних. Водночас, поміrne покращення точності вказує на те, що оптимізатор може досягти певних меж у своїй ефективності, що вимагає подальшого вдосконалення механізмів адаптації параметрів.

Висновки до розділу 4

У розділі 4 дисертаційної роботи здійснено поглиблений аналіз результатів навчання та експериментальних досліджень, спрямованих на перевірку ефективності розробленого методологічного підходу до класифікації тривимірних даних із застосуванням вдосконалених архітектур нейронних мереж. Отримані результати демонструють наукову та практичну значущість запропонованих методів і технологій у сучасних умовах автоматизованої обробки даних.

Дослідження підтвердило, що запропонована розріджена архітектура нейронної мережі забезпечує високу ефективність у виділенні ключових ознак із тривимірних даних. Це дозволяє суттєво підвищити точність класифікації. Такий підхід гарантує стабільні результати, що має вирішальне значення для задач високого рівня складності.

Одним із важливих досягнень є впровадження оптимізатора для видалення слабких зв'язків у нейронній мережі. Результати експериментів продемонстрували, що використання цього оптимізатора дає змогу суттєво зменшити кількість параметрів моделі, що, у свою чергу, позитивно впливає на її продуктивність. Зменшення розміру моделі та скорочення часу обробки забезпечують високу ефективність навіть у середовищах із обмеженими обчислювальними ресурсами, що є важливим для інтеграції в системи реального часу.

Аналіз впливу оптимізатора, відповідального за створення нових зв'язків у мережі, показав, що його ефективність варіюється залежно від характеристик

задачі. Хоча експерименти не дозволили зробити однозначні висновки, спостереження свідчать про потенціал цього механізму у покращенні адаптивності та точності моделі. Подальші дослідження в цьому напрямі можуть сприяти кращому розумінню ролі динамічного утворення зв'язків у структурі нейронних мереж.

Особливо важливою є універсальність розробленої архітектури. Вона демонструє здатність ефективно працювати з різними типами даних без необхідності внесення змін до структури мережі. Це забезпечує широку сферу застосування методології — від медичної діагностики до аналізу геопросторових даних і автоматизації робототехнічних процесів. Гнучкість архітектури також сприяє її адаптації до змінних умов середовища, що є ключовим фактором для інтеграції в практичні системи.

Загалом результати проведених досліджень підтверджують ефективність запропонованого підходу та його здатність вирішувати складні задачі автоматизованої обробки тривимірних даних. Використання вдосконалених архітектур нейронних мереж і алгоритмів оптимізації відкриває нові горизонти для підвищення точності та продуктивності аналізу даних. Запропоноване рішення має значний потенціал для впровадження у широкому спектрі галузей, створюючи основу для розвитку новітніх прикладних систем і технологій.

ВИСНОВКИ

В дисертаційній роботі розв'язане актуальне наукове завдання створення методів та програмних засобів для ефективної класифікації тривимірних даних на основі тривимірних нейронних мереж.

Дисертаційна робота спрямована на підвищення продуктивності та ефективності програмних засобів для класифікації тривимірних зображень на основі тривимірних нейронних мереж, що базуються на сучасних підходах до їхньої архітектури, обробки даних та адаптивної оптимізації. В процесі дослідження успішно виконано поставлені завдання, що дозволило досягти поставленої мети. Основні результати дисертації полягають у наступному:

1. проаналізовано сучасні методи та підходи до класифікації тривимірних даних. Проведено ґрунтовне дослідження сучасних методів обробки тривимірних даних, включаючи глибокі нейронні мережі та методи мета-навчання. Виявлено основні обмеження існуючих підходів, які стосуються недостатньої точності, високої обчислювальної складності, недостатньої швидкодії та слабкої інтеграції з автоматизованими системами;
2. реалізовано методи попередньої обробки тривимірних даних, а саме: нормалізація координат, білатеральне згладжування, фільтрація шумів із використанням алгоритму k-ближчих сусідів, перетворення даних у воксельне представлення. Також запропоновано інноваційне використання опонентної кольорової системи, яка підвищує інформативність характеристик вхідних даних і їх інтерпретацію нейронними мережами. Експерименти довели ефективність запропонованого методу, а саме зменшення втрат в діапазоні від 8,64% до 53,33% в процесі навчання нейронних мереж;
3. спроектовано та реалізовано архітектуру класифікатора для тривимірних даних, що реалізує розріджену топологію з локалізованими зв'язками між нейронами, яка забезпечує покращене вилучення ключових ознак з даних,

- що призводить до покращення точності класифікації для вокселів, в середньому, на 9,82%, та для точок на 19,75%;
4. розроблені два методи зміни кількості нейронних зв'язків, які видаляють та створюють локальні зв'язки мережі під час навчання, що забезпечує її адаптацію до специфіки вхідних даних, що в свою чергу призводить до прискорення роботи класифікатора при роботі з вокселями, в середньому на 23,72%, та при роботі з точками на 10,43%;
 5. реалізовано програмні засоби з підтримкою WebAPI, а саме програмні засоби для класифікації тривимірних зображень, які підтримують інтеграцію з іншими інформаційними системами, що забезпечує автоматизацію обробки тривимірних даних з можливістю масштабування;
 6. розроблено метод тестування та верифікації роботи класифікатора, що ґрунтується на використанні різних топологій шарів для виокремлення ключових ознак та тренуванні однакових повнозв'язних шарів;
 7. проведено детальні експерименти, для визначення точності класифікації тривимірних даних та продуктивності роботи створених програмних засобів. Результати аналізу експериментів підтвердили конкурентоспроможність запропонованих методів у вирішенні складних задач класифікації.

Практична значущість роботи полягає у створенні програмних засобів, що поєднує сучасні архітектури нейронних мереж, алгоритми оптимізації та підтримку масштабованої обробки тривимірних даних. Рішення орієнтоване на автоматизацію аналізу даних у таких прикладних сферах, як медична діагностика, промисловий контроль, робототехніка та екологічний моніторинг.

Загалом, розроблений підхід до класифікації тривимірних зображень демонструє високу точність та ефективність, що відкриває перспективи його застосування у широкому спектрі наукових і прикладних задач. Подальший розвиток цього напрямку може бути пов'язаний із дослідженням гібридних

моделей, які поєднують можливості різних типів нейронних мереж та методів мета-навчання.

Таким чином, усі поставлені у даному науковому дослідженні завдання повністю виконані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Chollet F. Deep Learning with Python. Manning Publications, 2018.
2. Goodfellow I., Bengio Y., Courville A. Deep Learning. MIT Press, 2016.
3. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. 2015. Vol. 521. pp. 436–444.
4. Bengio Y. Learning Deep Architectures for AI // Foundations and Trends® in Machine Learning. 2009. Vol. 2, No. 1. pp. 1–127.
5. Ji S., Xu W., Yang M., Yu K. 3D Convolutional Neural Networks for Human Action Recognition // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2013. Vol. 35, No. 1. pp. 221–231.
6. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016.
7. Krizhevsky A., Sutskever I., Hinton G. ImageNet Classification with Deep Convolutional Neural Networks // Advances in Neural Information Processing Systems. 2012.
8. Паладієв О. О., Лісовиченко О. І. Вплив зменшення розмірів нейронної мережі на її здатність до узагальнення // Міжвідомчий науково-технічний журнал «Адаптивні системи автоматичного управління». 2023. Т. 2. № 43. С. 124–130.
9. Paladiiev O., Lisovychenko O. The influence of the opponent's color model on the general capabilities of neural networks // Interdepartmental scientific-technical journal «Adaptive systems of automatic control». 2022. Vol. 2, No. 41. pp. 22–27.
10. Паладієв О. О., Лісовиченко О. І. Тривимірні нейронні мережі у завданнях кластеризації // Міжвідомчий науково-технічний журнал «Адаптивні системи автоматичного управління». 2024. Т. 1. № 44. С. 166–171.

11. Maturana D., Scherer S. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition // IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2015.
12. Qi C. R., Su H., Mo K., Guibas L. J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2017.
13. Chen Z., Zhang H., Xu Y., Tian Y. Deep Learning for 3D Point Clouds: A Survey // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2022. Vol. 44, No. 4. pp. 1627–1647.
14. Girardeau-Montaut D. CloudCompare Documentation, version 2.1 [Электронный ресурс]. URL: https://www.danielgm.net/cc/doc/qCC/Documentation_CloudCompare_version_2_1_eng.pdf (дата звернення: 25.01.2025).
15. Rusu R. B., Cousins S. 3D is here: Point Cloud Library (PCL) // Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 2011.
16. Khoshelham K., Elberink S. O. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications // Sensors. 2012. Vol. 12, No. 2. pp. 1437–1454.
17. Wujanz D., Krueger D., Neitzel F. Identification of Stable Areas in Unreferenced Laser Scans for Deformation Measurement // Sensors. 2017. Vol. 17, No. 6. pp. 1330–1345.
18. Axelsson P. Processing of laser scanner data—algorithms and applications // ISPRS Journal of Photogrammetry and Remote Sensing. 1999. Vol. 54, No. 2–3. pp. 138–147.
19. Vosselman G., Maas H.-G. Airborne and Terrestrial Laser Scanning. Whittles Publishing, 2010.
20. Geiger A., Lenz P., Stiller C., Urtasun R. Vision meets robotics: The KITTI dataset // International Journal of Robotics Research. 2013. Vol. 32, No. 11. pp. 1231–1237.
21. Pu S., Vosselman G. Knowledge-based reconstruction of building models from terrestrial laser scanning data // ISPRS Journal of Photogrammetry and Remote Sensing. 2009. Vol. 64, No. 6. pp. 575–584.

22. Shan J., Toth C. Topographic Laser Ranging and Scanning: Principles and Processing. CRC Press, 2018.
23. Hyypä J., Kukko A., Litkey P., Kaartinen H., Yu X. 3D Data Collection, Processing and Applications with Airborne and Terrestrial Laser Scanning // Remote Sensing. 2013. Vol. 5, No. 10. pp. 4300–4323.
24. Zhou Q.-Y., Park J., Koltun V. Open3D: A Modern Library for 3D Data Processing // arXiv preprint. 2018.
25. Hackel T., Wegner J. D., Schindler K. Fast Semantic Segmentation of 3D Point Clouds with Strongly Varying Density // ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. 2016.
26. Holz D., Ichim A. E., Tombari F., Rusu R. B., Behnke S. Registration with the Point Cloud Library: A Modular Framework for Aligning in 3-D // IEEE Robotics & Automation Magazine. 2015. Vol. 22, No. 4. pp. 110–124.
27. Abadi M., Barham P., Chen J., Chen Z., Davis A., Dean J. та ін. TensorFlow: A System for Large-Scale Machine Learning // Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI). 2016.
28. Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G. та ін. PyTorch: An Imperative Style, High-Performance Deep Learning Library // Advances in Neural Information Processing Systems. 2019.
29. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks // Proceedings of the 36th International Conference on Machine Learning (ICML). 2019. pp. 6105–6114.
30. Schroeder W. J., Martin K. M., Lorensen W. E. The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics. Kitware, 2006.
31. Ibanez L., Schroeder W., Ng L., Cates J. The ITK Software Guide. Kitware, 2003.
32. Jacobson A., Panozzo D. libigl: A Simple C++ Geometry Processing Library // Computer Graphics Forum. 2017. Vol. 36, No. 5. pp. 79–87.
33. Botsch M., Kobbelt L., Pauly M., Alliez P., Levy B. Polygon Mesh Processing. CRC Press, 2010.

34. Isenburg M. LASzip: Lossless Compression of LiDAR Data // Photogrammetric Engineering & Remote Sensing. 2013. Vol. 79, No. 2. pp. 209–217.
35. Cook B. D., Corp L. A., Nelson R. F., Middleton E. M., Morton D. C., Masek J. G. та ін. NASA Goddard's LiDAR, Hyperspectral and Thermal (G-LiHT) Airborne Imager // Remote Sensing. 2013. Vol. 5, No. 8. pp. 4045–4066.
36. Minsky M., Papert S. Limitations of Perceptrons // Perceptrons. Cambridge, MA: MIT Press, 1988. pp. 73–94.
37. Cover T. M., Thomas J. A. Information Theory in Machine Learning // Elements of Information Theory. Hoboken, NJ: Wiley-Interscience, 2006. pp. 287–325.
38. Bishop C. M. Pattern Recognition and Neural Networks // Pattern Recognition and Machine Learning. New York: Springer, 2006. pp. 123–160.
39. Hastie T., Tibshirani R., Friedman J. Neural Networks and Deep Learning // The Elements of Statistical Learning. New York: Springer, 2009. pp. 389–417.
40. Graves A., Mohamed A.-r., Hinton G. Speech Recognition with Deep Recurrent Neural Networks // IEEE Transactions on Audio, Speech, and Language Processing. 2013. Vol. 21, No. 3. pp. 524–536.
41. Hinton G., Srivastava N., Krizhevsky A. Dropout: A Simple Way to Prevent Neural Networks from Overfitting // Proceedings of the International Conference on Machine Learning (ICML). 2014.
42. Kingma D. P., Ba J. Adam: A Method for Stochastic Optimization // International Conference on Learning Representations (ICLR). 2015. arXiv: 1412.6980.
43. Nielsen M. Neural Networks and Deep Learning. Determination Press, 2015.
44. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition // International Conference on Learning Representations (ICLR). 2015. arXiv: 1409.1556.
45. Ng A. Sparse Autoencoder [Електронний ресурс] // Лекційні матеріали курсу «CS294A», Стенфордський університет, 2011. URL: http://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf (дата звернення: 25.01.2025).

46. Sutton R. S., Barto A. G. Temporal-Difference Learning // Reinforcement Learning: An Introduction. Cambridge, MA: MIT Press, 2018. pp. 125–145.
47. Raschka S., Mirjalili V. Python Machine Learning. Birmingham: Packt Publishing, 2019.
48. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. Upper Saddle River, NJ: Prentice Hall, 2020.
49. Bergstra J., Yamins D., Cox D. D. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures // Proceedings of the 30th International Conference on Machine Learning (ICML). 2013. pp. 115–123.
50. Foster D. Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play. Sebastopol, CA: O'Reilly Media, 2019.
51. Aggarwal C. C. Neural Networks and Deep Learning: A Textbook. Cham: Springer, 2018.
52. Mohri M., Rostamizadeh A., Talwalkar A. Foundations of Machine Learning. Cambridge, MA: MIT Press, 2018.
53. Olah C., Satyanarayan A., Johnson I., Carter S., Schubert L., Ye K., Mordvintsev A. The Building Blocks of Interpretability // Distill. 2018. DOI: 10.23915/distill.00010.
54. Murphy K. P. Probabilistic Machine Learning: Advanced Topics. MIT Press, 2023.
55. LeCun Y., Bottou L., Orr G. B., Müller K.-R. Efficient BackProp // Neural Networks: Tricks of the Trade, 2nd ed. Berlin, Heidelberg: Springer, 2012. pp. 9–48.
56. Arulkumaran K., Deisenroth M. P., Brundage M., Bharath A. A. Deep Reinforcement Learning: A Brief Survey // IEEE Signal Processing Magazine. 2017. Vol. 34, No. 6. pp. 26–38.
57. Li Y. Deep Reinforcement Learning: An Overview // arXiv preprint. 2017. arXiv: 1701.07274.
58. Papadimitriou C., Roughgarden T. Computing Equilibria in Games // Algorithmic Game Theory. Cambridge: Cambridge University Press, 2007. pp. 111–138.

59. Choy C., Gwak J., Savarese S. 4D Spatio-Temporal Convolutional Networks: Minkowski Convolutional Neural Networks // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2019. pp. 3075–3084.
60. Zhou Y., Tuzel O. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. pp. 4490–4499.
61. Yan Y., Mao Y., Li B. SECOND: Sparsely Embedded Convolutional Detection // Sensors. 2018. Vol. 18, No. 10. p. 3337.
62. Lang A. H., Vora S., Caesar H., Zhou L., Yang J., Beijbom O. PointPillars: Fast Encoders for Object Detection from Point Clouds // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2019. pp. 12697–12705.
63. Shi S., Wang X., Li H. PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2020. pp. 10529–10538.
64. Qi C. R., Liu W., Wu C., Su H., Guibas L. J. Frustum PointNets for 3D Object Detection from RGB-D Data // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. pp. 918–927.
65. Qi C. R., Litany O., He K., Guibas L. J. ImVoteNet: Boosting 3D Object Detection in Indoor Scenes with Imagery // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2020. pp. 4404–4413.
66. Yang Z., Sun Y., Liu S., Shen X., Jia J. 3DSSD: Point-based 3D Single Stage Object Detector // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2020. pp. 11040–11048.
67. Hu Q., Yang B., Xie L., Rosa S., Li S., Wang Z., Markham A., Trigoni N. RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2020. pp. 11105–11114.

68. Wu W., Qi Z., Fuxin L. PointConv: Deep Convolutional Networks on 3D Point Clouds // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2019. pp. 9621–9630.
69. Zhang Y., Feng C., Chen Y., Vosselman G. ShellNet: Efficient Point Cloud Convolutional Neural Network Using Concentric Shells Statistics // Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2019. pp. 1607–1615.
70. Zhu X., Zhou H., Wang T., Song S. Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2021. pp. 9939–9948.
71. Milioto A., Vizzo I., Behley J., Stachniss C. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation // IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2019. pp. 4213–4220.
72. Graham B., Engelcke M., van der Maaten L. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. pp. 9224–9232.
73. Choy C., Dong W., Koltun V. Deep Global Registration // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2020. pp. 2514–2523.
74. Du S., Vemulapalli R., Mansfield P., Liu Y. 3D CNN Based Classification of Point Cloud Data // European Conference on Computer Vision (ECCV) Workshops. 2018. pp. 170–179.
75. Liu Y., Engelmann F., Kontogianni T., Leibe B. Deep Learning-Based Classification of 3D Urban Point Clouds: A Review // ISPRS Journal of Photogrammetry and Remote Sensing. 2022. Vol. 184. pp. 225–245.
76. Fan L., Wang L., Huang Y., Feng R., Wang X. 3D-GCN: Geometry-Aware 3D Convolutional Network // Neurocomputing. 2021. Vol. 444. pp. 322–331.
77. Yang L., Xie S., Chen Y., Hu L. A Survey of Deep Learning Techniques for LiDAR-based 3D Object Detection // Sensors. 2022. Vol. 22, No. 21. p. 8322.

78. Choy C., Park J., Koltun V. Fully Convolutional Geometric Features // Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2019. pp. 8958–8966.
79. Choy C., Gwak J., Savarese S. MinkowskiNet: Deep Sparse Tensor Convnets for 3D Segmentation // arXiv preprint. 2019. arXiv:1904.08755.
80. Park J., Zhou Q.-Y., Koltun V. Colored Point Cloud Registration Revisited // Proceedings of the IEEE International Conference on Computer Vision (ICCV). 2017. pp. 143–152.

ДОДАТОК А СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА

The influence of the opponent's color model on the general capabilities of neural networks / PaladiievO., LisovychenkoO.// Interdepartmental scientific-technical journal «Adaptive systems of automatic control».-2022.-№ 2 (41).-P. 22-27

DOI: 10.20535/1560-8956.41.2022.271335

Вплив зменшення розмірів нейронної мережі на її здатність до узагальнення / Паладієв О., Лісовиченко О.// Міжвідомчий науково-технічний журнал «Адаптивні системи автоматичного управління».-2023.-№ 2 (43).-С. 124-130

DOI: 10.20535/1560-8956.43.2023.292262

Тривимірні нейронні мережі у завданнях кластеризації / Паладієв О., Лісовиченко О.// Міжвідомчий науково-технічний журнал «Адаптивні системи автоматичного управління».-2024.-№ 1 (44).-С. 166-171

DOI: 10.20535/1560-8956.44.2024.302431

ДОДАТОК Б ЧАСТИНА ПРОГРАМНОГО КОДУ

```

import sys

import time

import tqdm

import time

import numpy as np

import tensorflow as tf


class SparseLayer(tf.keras.layers.Layer):

    def __init__(self, output_shape, radius, prune_threshold=0.1, create_threshold=0.2,
activation=None, **kwargs):

        super(SparseLayer, self).__init__(**kwargs)

        self.output_shape = output_shape

        self.m = output_shape[0]

        self.r = radius

        self.activation = activation

        self.prune_threshold = prune_threshold

        self.create_threshold = create_threshold

        self.indices = None

        self.values = None

        self.train_step = tf.Variable(0, trainable=False, dtype=tf.int64, name="train_step")

    def build(self, input_shape):

        self.input_shape = input_shape

        self.n = input_shape[1]

```

```
if self.indices is None:
```

```
    connections = calculate_connections_asymmetric_3d(self.m, self.n, self.r)
```

```
    self.indices = tf.Variable(
```

```
        connections,
```

```
        dtype=tf.int64,
```

```
        trainable=False, name="sparse_indices"
```

```
    )
```

```
if self.values is None:
```

```
    self.values = self.add_weight(
```

```
        shape=(int(self.indices.shape[0]),),
```

```
        initializer=tf.keras.initializers.RandomNormal(mean=0.0, stddev=0.05),
```

```
        #initializer = tf.keras.initializers.TruncatedNormal(mean=0.0, stddev=0.05),
```

```
        #initializer = tf.keras.initializers.HeNormal(),
```

```
        #initializer = tf.keras.initializers.GlorotUniform(),
```

```
        trainable=True,
```

```
        name="sparse_values"
```

```
    )
```

```
self.dense_shape = [self.m * self.m * self.m, self.n * self.n * self.n]
```

```
super(SparseLayer, self).build(input_shape)
```

```
def call(self, inputs, training=False):
```

```
    start_time = time.time()
```

```
    batch_size = tf.shape(inputs)[0]
```

```

inputs_resaped = tf.reshape(inputs, (batch_size, -1))

inputs_transposed = tf.transpose(inputs_resaped)

kernel_sparse = tf.sparse.SparseTensor(
    indices=self.indices,
    values=self.values,
    dense_shape=self.dense_shape
)

kernel_sparse = tf.sparse.reorder(kernel_sparse)

outputs = tf.sparse.sparse_dense_matmul(kernel_sparse, inputs_transposed)

if self.activation is not None:
    activation_func = tf.keras.activations.get(self.activation)
    outputs = activation_func(outputs)

outputs_transposed = tf.transpose(outputs)

outputs_resaped = tf.reshape(outputs_transposed, (batch_size, self.m, self.m, self.m, 1))

end_time = time.time()

execution_time = end_time - start_time # Время выполнения `call` в секундах

model_name = self.get_model_name()

layer_name = self.name

phase = "train" if training else "validation" # Этап: обучение или валидация

tf.summary.histogram(f"{model_name}/{phase}/{layer_name}/values", self.values,
step=self.train_step)

tf.summary.scalar(f"{model_name}/{phase}/{layer_name}/values_count", tf.size(self.values),
step=self.train_step)

```

```
tf.summary.scalar(f'{model_name}/{phase}/{layer_name}/execution_time', execution_time,
step=self.train_step)
```

```
self.train_step.assign_add(1)
```

```
return outputs_reshaped
```

```
def meta_learning(self):
```

```
    start_time = time.time()
```

```
    unique_neurons = tf.unique(self.indices[:, 0])[0]
```

```
    new_indices = []
```

```
    new_values = []
```

```
    create_count = 0
```

```
    prune_rest_count = 0
```

```
    for neuron in tqdm(unique_neurons, desc="Meta learning"):
```

```
        mask = tf.equal(self.indices[:, 0], neuron)
```

```
        neuron_indices = tf.boolean_mask(self.indices, mask)
```

```
        neuron_values = tf.boolean_mask(self.values, mask)
```

```
        mean_weight = tf.reduce_mean(tf.abs(neuron_values))
```

```
        pruning_mask = tf.greater_equal(tf.abs(neuron_values), self.prune_threshold)
```

```
        new_indices.append(tf.boolean_mask(neuron_indices, pruning_mask))
```

```
        new_values.append(tf.boolean_mask(neuron_values, pruning_mask))
```

```
        prune_rest_count += len(tf.boolean_mask(neuron_values, pruning_mask))
```

```
        if mean_weight > self.create_threshold:
```

```
            existing_connections = tf.cast(neuron_indices[:, 1], dtype=tf.int64)
```

```

        new_connections = calculate_connections_asymmetric_index_3d(int(neuron), self.n,
self.m, self.r+1)

        new_connections = tf.cast(new_connections, dtype=tf.int64)

        mask_duplicates = tf.reduce_all(tf.expand_dims(new_connections, axis=1) !=
tf.expand_dims(existing_connections, axis=0), axis=1)

        filtered_connections = tf.boolean_mask(new_connections, mask_duplicates)

        new_connection = tf.random.shuffle(filtered_connections)[0]

        new_indices.append(tf.constant([[int(neuron.numpy()), int(new_connection.numpy())],
dtype=tf.int64))

        new_values.append(tf.constant([int(mean_weight.numpy())], dtype=tf.float32))

        create_count += 1

new_indices = tf.concat(new_indices, axis=0)

new_values = tf.concat(new_values, axis=0)

self.indices = tf.Variable(new_indices, dtype=tf.int64, trainable=False,
name="sparse_indices")

self.values = tf.Variable(new_values, dtype=tf.float32, trainable=True, name="sparse_values")

end_time = time.time()

execution_time = end_time - start_time # Время выполнения `meta_learning` в секундах

model_name = self.get_model_name()

layer_name = self.name

phase = "train"

tf.summary.scalar(f'{model_name}/{phase}/{layer_name}/meta_learning/create_count',
create_count, step=self.train_step)

```

```

tf.summary.scalar(f'{model_name}/{phase}/{layer_name}/meta_learning/prune_count',
tf.shape(self.indices)[0] - prune_rest_count, step=self.train_step)

tf.summary.scalar(f'{model_name}/{phase}/{layer_name}/meta_learning/execution_time',
execution_time, step=self.train_step)

```

```
def get_model_name(self):
```

```
    # Получение имени модели, в которой находится слой
```

```
    model = self._keras_history.model if hasattr(self, '_keras_history') else None
```

```
    return model.name if model else "UnnamedModel"
```

```
def compute_output_shape(self, input_shape):
```

```
    return (input_shape[0], self.m, self.m, self.m, 1)
```

```
def get_config(self):
```

```
    tf.print("Get config")
```

```
    config = super(SparseLayer, self).get_config()
```

```
    config.update({
```

```
        'output_shape': self.output_shape,
```

```
        'radius': self.r,
```

```
        'prune_threshold': self.prune_threshold,
```

```
        'create_threshold': self.create_threshold,
```

```
        'activation': self.activation,
```

```
        'indices': self.indices.numpy().tolist(),
```

```
        'values': self.values.numpy().tolist()
```

```
    })
```

```
    return config
```

```

@classmethod
def from_config(cls, config):

    indices = tf.constant(config.pop('indices'), dtype=tf.int64)

    values = tf.constant(config.pop('values'), dtype=tf.float32)

    layer = cls(**config)

    layer.indices = tf.Variable(indices, trainable=False, name="sparse_indices")

    layer.values = layer.add_weight(

        shape=values.shape,

        initializer=tf.constant_initializer(values.numpy()),

        trainable=True,

        name="sparse_values"

    )

    return layer


class MetaLearningCallback(tf.keras.callbacks.Callback):

    def __init__(self, layer, log_dir="logs"):

        super(MetaLearningCallback, self).__init__()

        self.layer = layer

        self.log_dir = log_dir

        self.writer = tf.summary.create_file_writer(log_dir)

    def get_layer_memory(self, layer):

```



```

size_values = sys.getsizeof(layer.values.numpy())

total_size = size_values

return total_size


def on_epoch_end(self, epoch, logs=None):

    if epoch > 2:

        model_name = self.layer.get_model_name()

        layer_name = self.layer.name # Имя слоя


        with self.writer.as_default():

            memory_usage_mb = self.get_layer_memory(self.layer) / 1048576

            tf.summary.scalar(f'{model_name}/{layer_name}/memory_usage_MB',
memory_usage_mb, step=epoch)

            self.writer.flush()


def calculate_connections_asymmetric_3d(n, m, r):

    start_time = time.time()

    input_size = n * n * n

    output_size = m * m * m

    scale = m / n

    z1, y1, x1 = np.indices((n, n, n))

    offsets = np.array([(dz, dy, dx)

        for dz in range(-r, r + 1)

        for dy in range(-r, r + 1)

        for dx in range(-r, r + 1)])

```

```

connections = []

for z, y, x in zip(z1.flatten(), y1.flatten(), x1.flatten()):

    cz, cy, cx = int(z * scale), int(y * scale), int(x * scale)

    for dz, dy, dx in offsets:

        nz, ny, nx = cz + dz, cy + dy, cx + dx

        if 0 <= nz < m and 0 <= ny < m and 0 <= nx < m:

            in_idx = z * n * n + y * n + x

            out_idx = nz * m * m + ny * m + nx

            if 0 <= out_idx < output_size:

                connections.append((in_idx, out_idx))

end_time = time.time()

return connections

def calculate_connections_asymmetric_index_3d(index, n, m, r):

    start_time = time.time()

    scale = m / n

    z = index // (n * n)

    y = (index % (n * n)) // n

    x = index % n

    cz, cy, cx = int(z * scale), int(y * scale), int(x * scale)

```

```

offsets = np.array([(dz, dy, dx)
                    for dz in range(-r, r + 1)
                    for dy in range(-r, r + 1)
                    for dx in range(-r, r + 1)])

connections = []

output_size = m * m * m

for dz, dy, dx in offsets:
    nz, ny, nx = cz + dz, cy + dy, cx + dx

    if 0 <= nz < m and 0 <= ny < m and 0 <= nx < m:
        out_idx = nz * m * m + ny * m + nx

        if 0 <= out_idx < output_size:
            connections.append(out_idx)

end_time = time.time()

return connections

```

ДОДАТОК В МОДЕЛІ КЛАСИФІКАТОРІВ

Моделі CNN класифікатора

Таблиця В.1 - Архітектура моделі CNN (model_1).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|------------------------|----------------------|
| Згортковий тривимірний шар (Conv3D) | (None, 28, 28, 28, 12) | 336 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 14, 14, 14, 12) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 12, 12, 12, 26) | 8,450 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 6, 6, 6, 26) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 4, 4, 4, 51) | 35,853 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 2, 2, 2, 51) | 0 |
| Шар згортання в один вектор (Flatten) | (None, 408) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 204,500 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Таблиця В.2 - Архітектура моделі CNN (model_2).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|------------------------|----------------------|
| Згортковий тривимірний шар (Conv3D) | (None, 28, 28, 28, 26) | 728 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 14, 14, 14, 26) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 12, 12, 12, 51) | 35,853 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 6, 6, 6, 51) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 4, 4, 4, 77) | 106,106 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 2, 2, 2, 77) | 0 |
| Шар згортання в один вектор (Flatten) | (None, 616) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 308,500 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Таблиця В.3 - Архітектура моделі CNN (model_3).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|------------------------|----------------------|
| Згортковий тривимірний шар (Conv3D) | (None, 28, 28, 28, 26) | 728 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 14, 14, 14, 26) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 12, 12, 12, 51) | 35,853 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 6, 6, 6, 51) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 4, 4, 4, 102) | 140,556 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 2, 2, 2, 102) | 0 |
| Шар згортання в один вектор (Flatten) | (None, 816) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 408,500 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Таблиця В.4 - Архітектура моделі CNN (model_4).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|------------------------|----------------------|
| Згортковий тривимірний шар (Conv3D) | (None, 28, 28, 28, 26) | 728 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 14, 14, 14, 26) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 12, 12, 12, 51) | 35,853 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 6, 6, 6, 51) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 4, 4, 4, 102) | 140,556 |
| Згортковий тривимірний шар (Conv3D) | (None, 2, 2, 2, 205) | 564,775 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 1, 1, 1, 205) | 0 |
| Шар згортання в один вектор (Flatten) | (None, 205) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 103,000 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Таблиця В.5 - Архітектура моделі CNN (model_5).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|------------------------|----------------------|
| Згортковий тривимірний шар (Conv3D) | (None, 28, 28, 28, 26) | 728 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 14, 14, 14, 26) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 12, 12, 12, 51) | 35,853 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 6, 6, 6, 51) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 4, 4, 4, 102) | 140,556 |
| Згортковий тривимірний шар (Conv3D) | (None, 2, 2, 2, 205) | 564,775 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 1, 1, 1, 205) | 0 |
| Згортковий тривимірний шар (Conv3D) | (None, 2, 2, 2, 410) | 2,269,760 |
| Тривимірний шар максимальної під вибірки (MaxPooling3D) | (None, 1, 1, 1, 410) | 0 |
| Шар згортання в один вектор (Flatten) | (None, 410) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 205,500 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Моделі SNN та SNNO класифікаторів

Таблиця В.6 - Архітектура моделей SNN та SNNO (model_1).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|-----------------------|----------------------|
| Розріджений тривимірний шар (SparseLayer3D) | (None, 18, 18, 18, 1) | 148,877 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 8, 8, 8, 1) | 12,167 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 6, 6, 6, 1) | 4,913 |
| Шар згортання в один вектор (Flatten) | (None, 216) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 108,500 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Таблиця В.7 - Архітектура моделей SNN та SNNO (model_2).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|-----------------------|----------------------|
| Розріджений тривимірний шар (SparseLayer3D) | (None, 20, 20, 20, 1) | 205,379 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 10, 10, 10, 1) | 24,389 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 7, 7, 7, 1) | 8,000 |
| Шар згортання в один вектор (Flatten) | (None, 343) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 172,000 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Таблиця В.8 - Архітектура моделей SNN та SNNO (model_3).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|-----------------------|----------------------|
| Розріджений тривимірний шар (SparseLayer3D) | (None, 22, 22, 22, 1) | 274,625 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 12, 12, 12, 1) | 42,875 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 8, 8, 8, 1) | 12,167 |
| Шар згортання в один вектор (Flatten) | (None, 512) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 256,500 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Таблиця В.9 - Архітектура моделей SNN та SNNO (model_4).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|-----------------------|----------------------|
| Розріджений тривимірний шар (SparseLayer3D) | (None, 24, 24, 24, 1) | 357,911 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 14, 14, 14, 1) | 68,921 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 9, 9, 9, 1) | 17,576 |
| Шар згортання в один вектор (Flatten) | (None, 729) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 365,000 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

Таблиця В.10 - Архітектура моделей SNN та SNNO (model_5).

| Тип шару | Вихідна форма | Кількість параметрів |
|---|-----------------------|----------------------|
| Розріджений тривимірний шар (SparseLayer3D) | (None, 30, 30, 30, 1) | 681,472 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 25, 25, 25, 1) | 1,771,561 |
| Розріджений тривимірний шар (SparseLayer3D) | (None, 11, 11, 11, 1) | 148,877 |
| Шар згортання в один вектор (Flatten) | (None, 1331) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 500) | 666,000 |
| Шар виключення нейронів (Dropout) | (None, 500) | 0 |
| Щільно зв'язаний шар (Dense) | (None, 40) | 20,040 |

ДОДАТОК Г ГРАФІКИ

Результати класифікаторів для роботи з вокселями

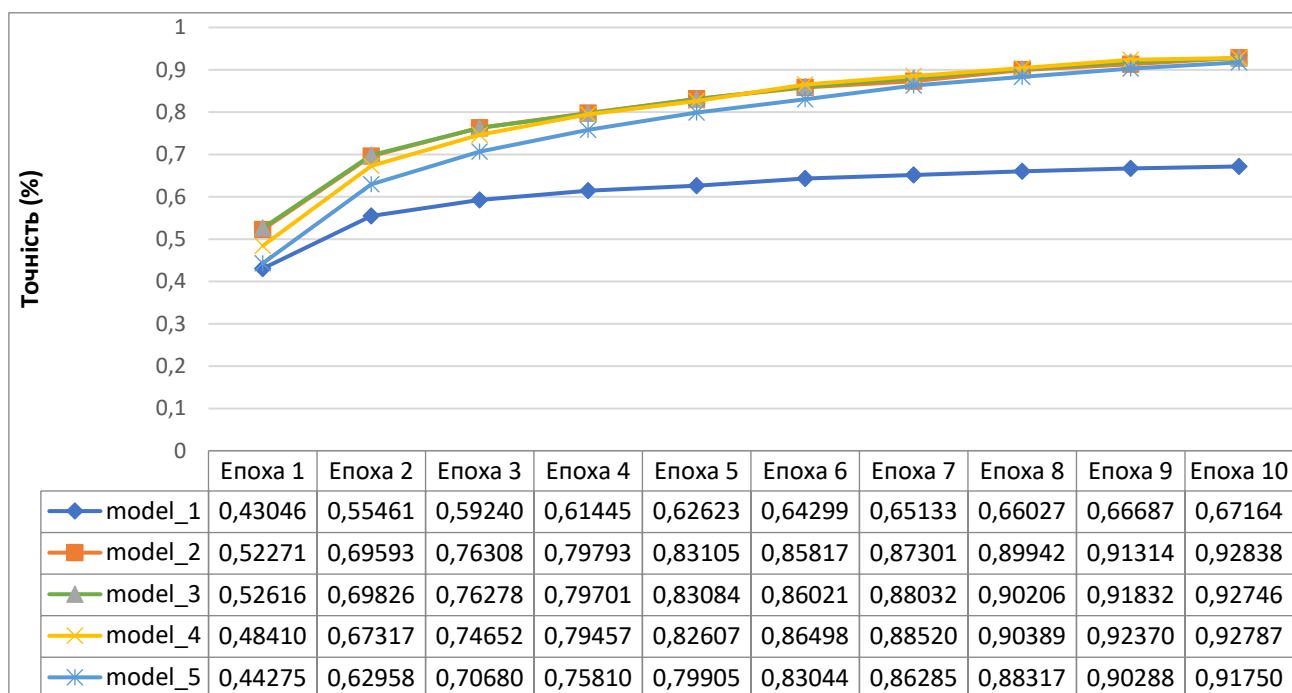


Рисунок Г.1 - Точність CNN моделей на навчанні на вокселях (більше - краще).

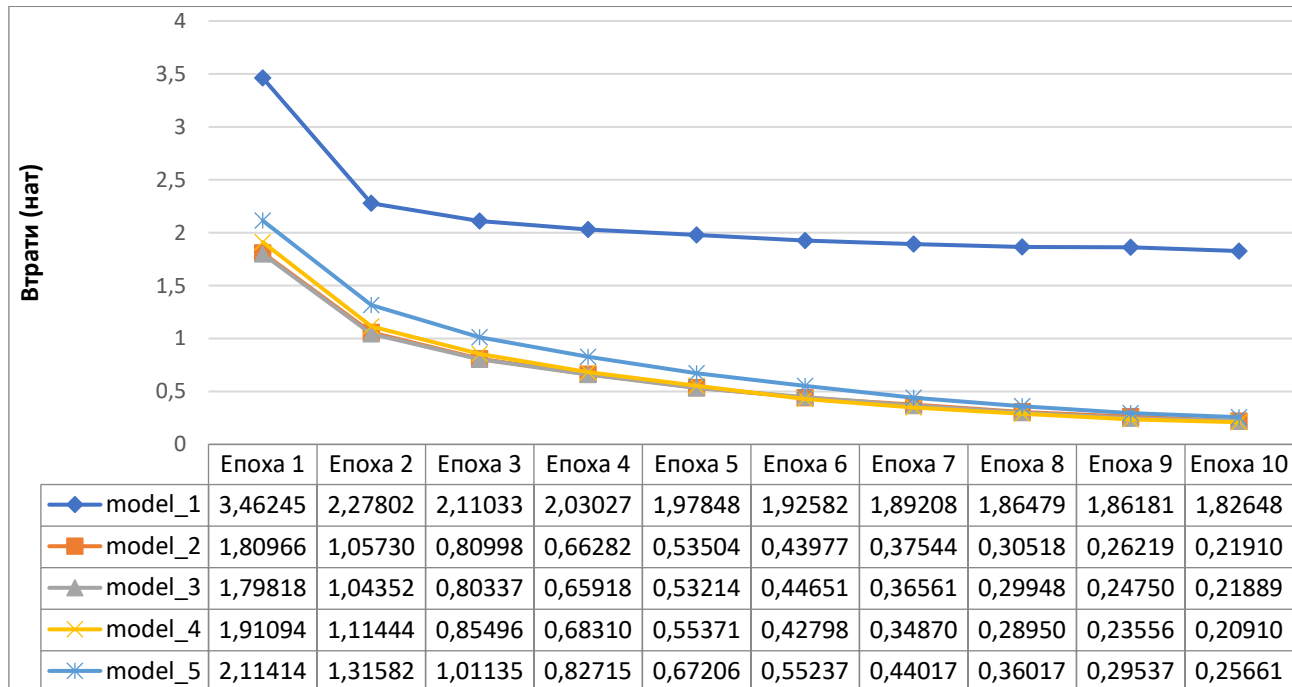


Рисунок Г.2 - Втрати CNN моделей на навчанні на вокселях (менше - краще).

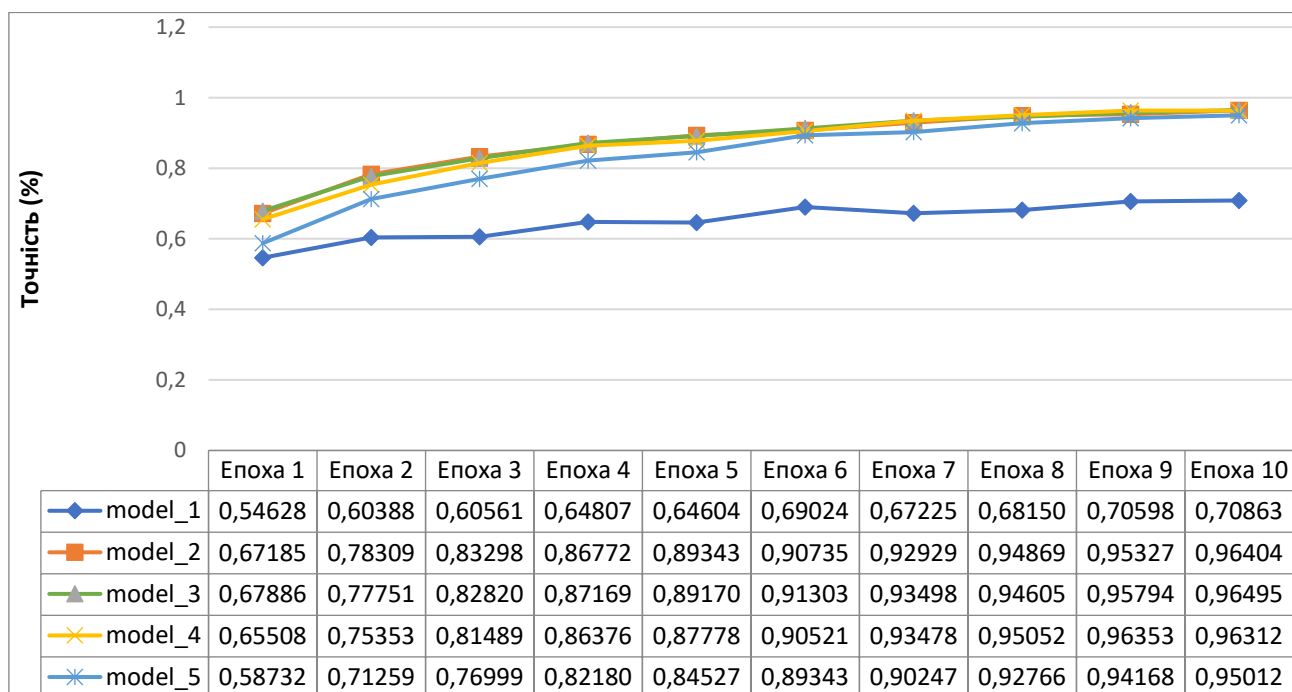


Рисунок Г.3 - Точність CNN моделей на валідації на вокселях (більше - краще).

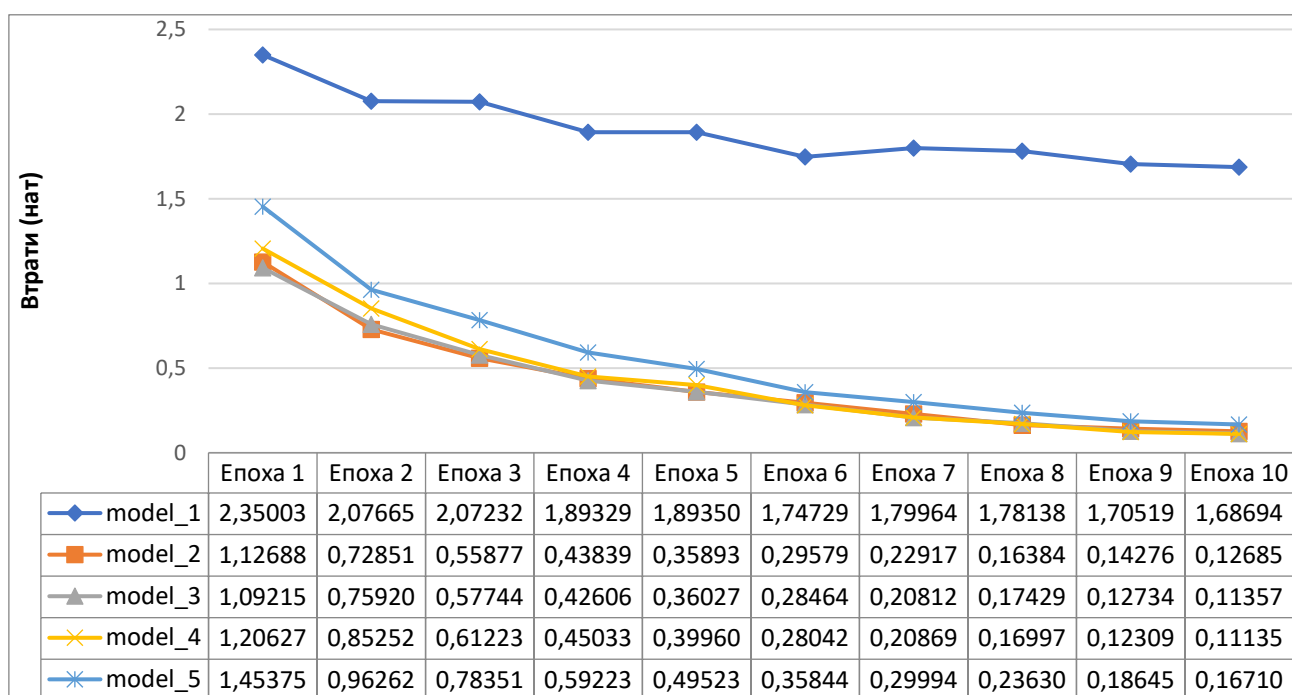


Рисунок Г.4 - Втрати CNN моделей на валідації на вокселях (менше - краще).

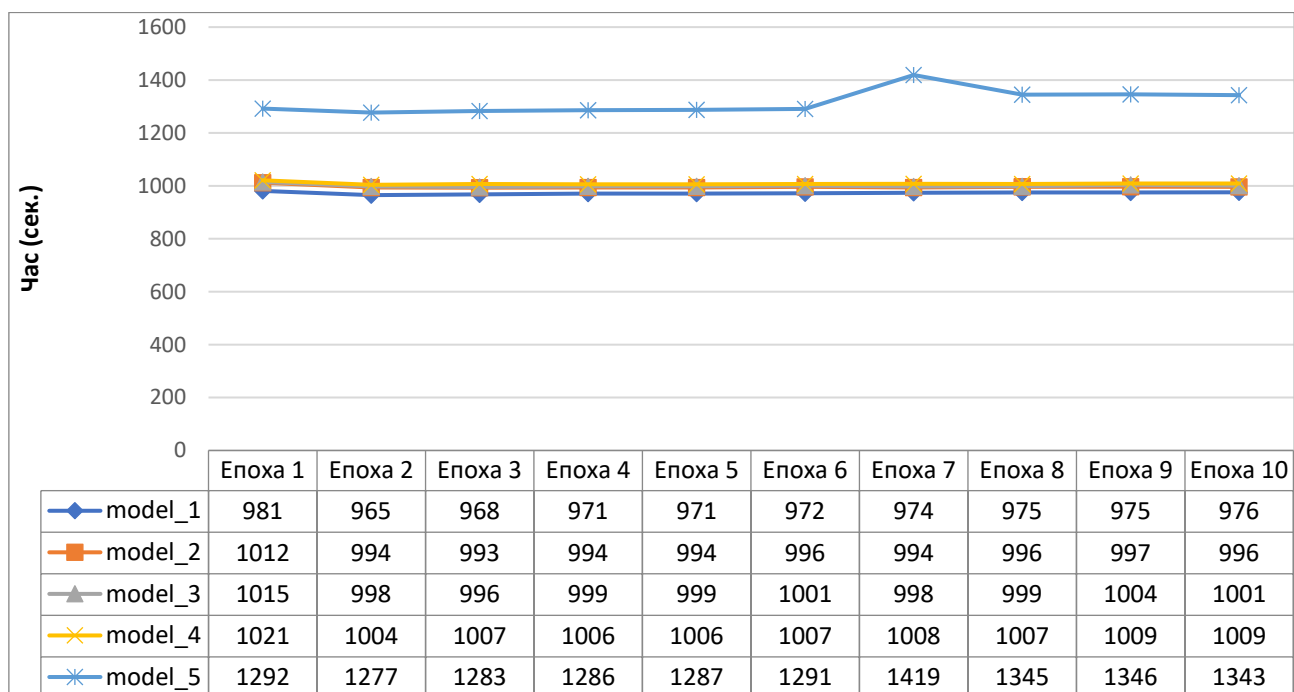


Рисунок Г.5 - Час навчання CNN моделей на вокселях (менше - краще).

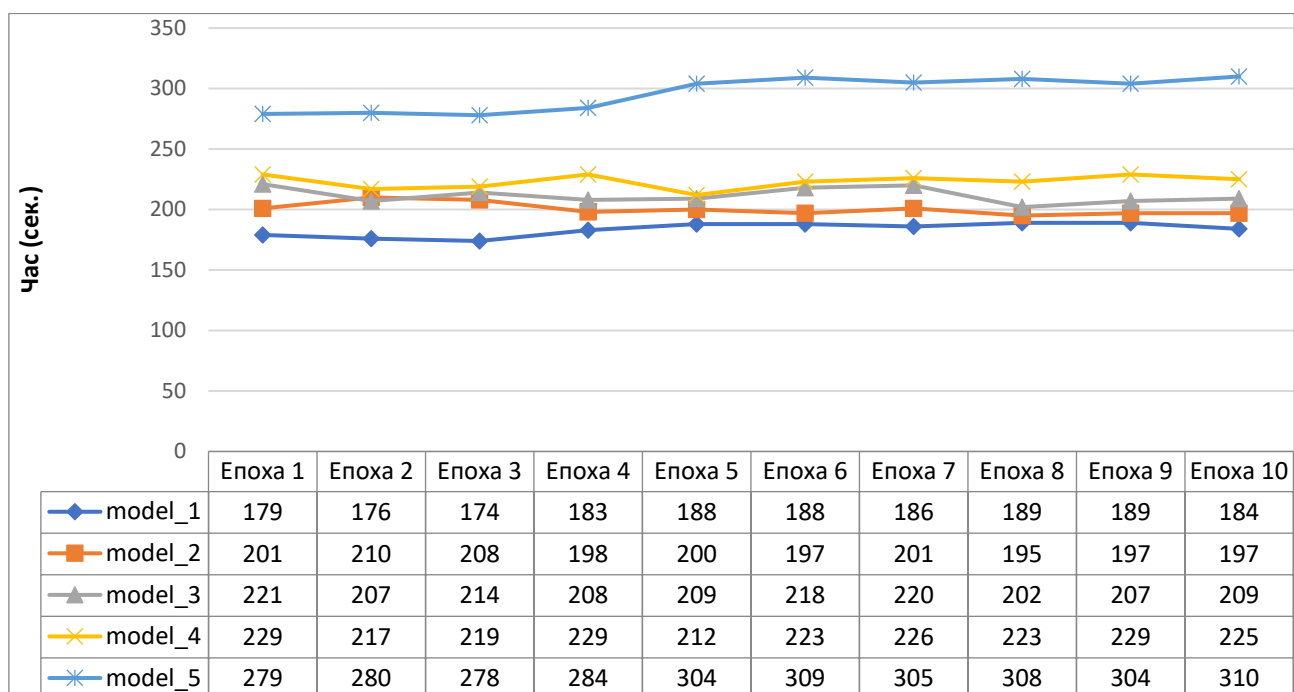


Рисунок Г.6 - Час роботи CNN моделей на вокселях (менше - краще).

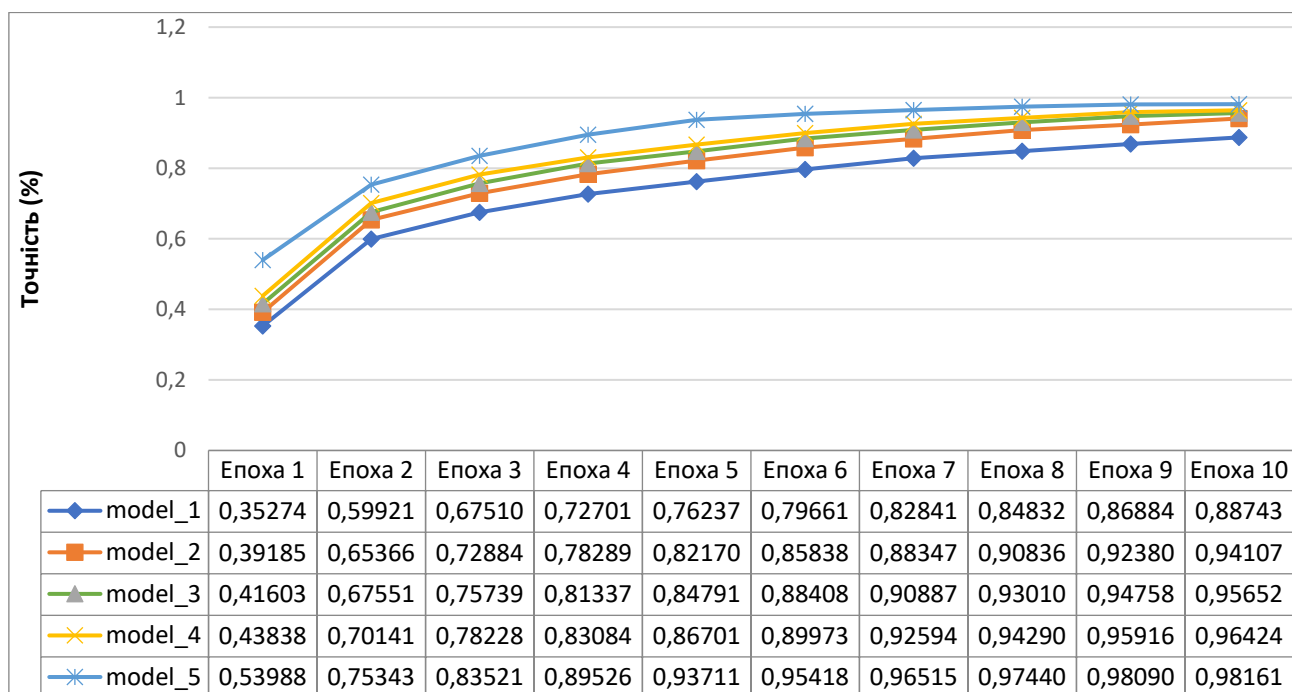


Рисунок Г.7 - Точність SNN моделей на навчанні на вокселях (більше - краще).

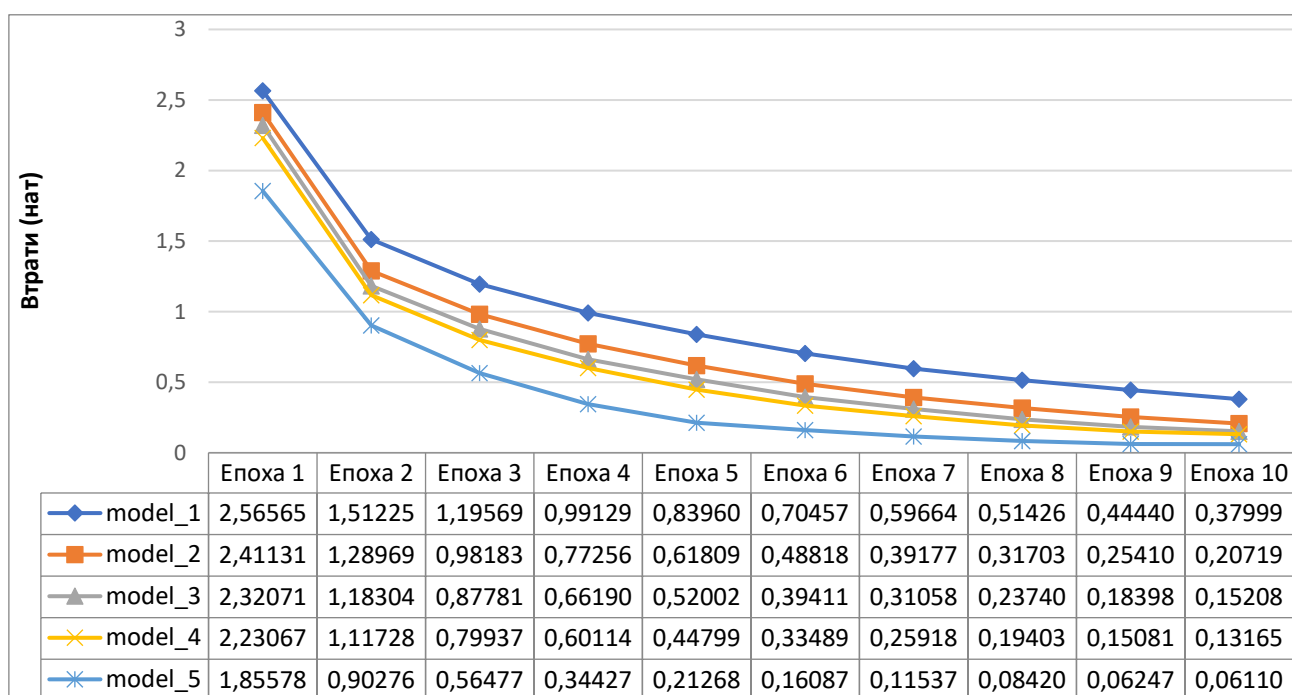


Рисунок Г.8 - Втрати SNN моделей на навчанні на вокселях (менше - краще).

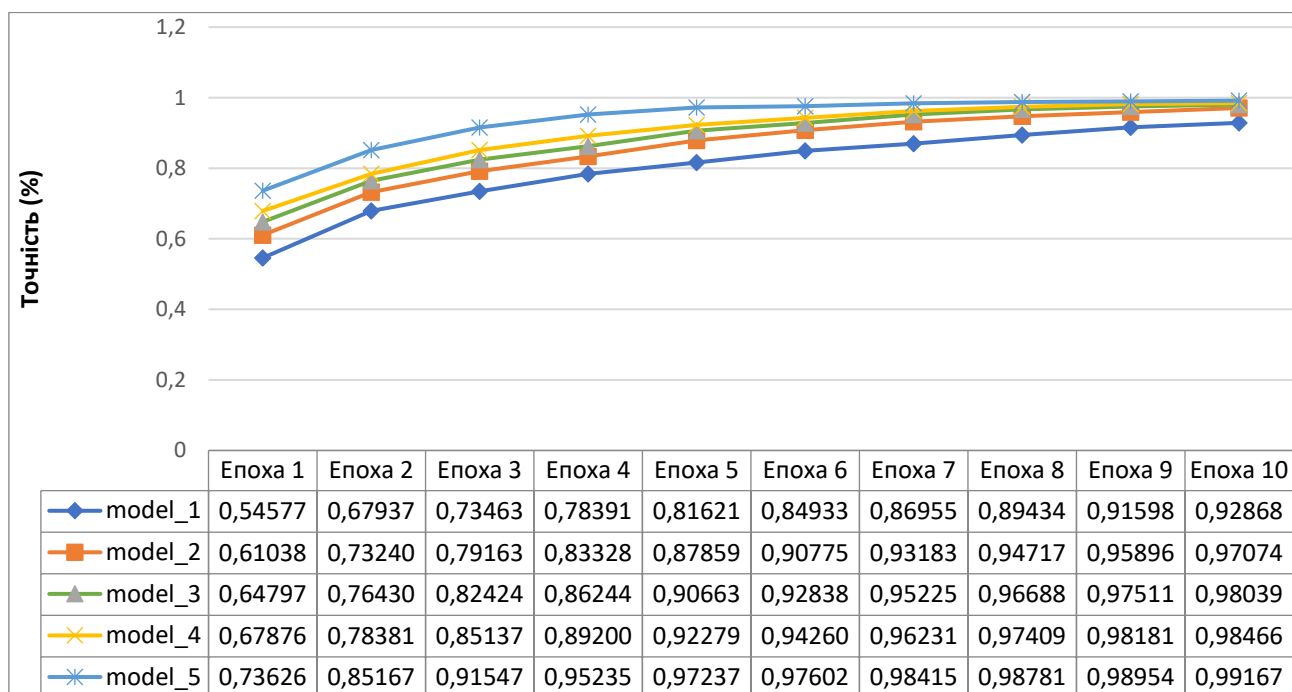


Рисунок Г.9 - Точність SNN моделей на валідації на вокселях (більше - краще).

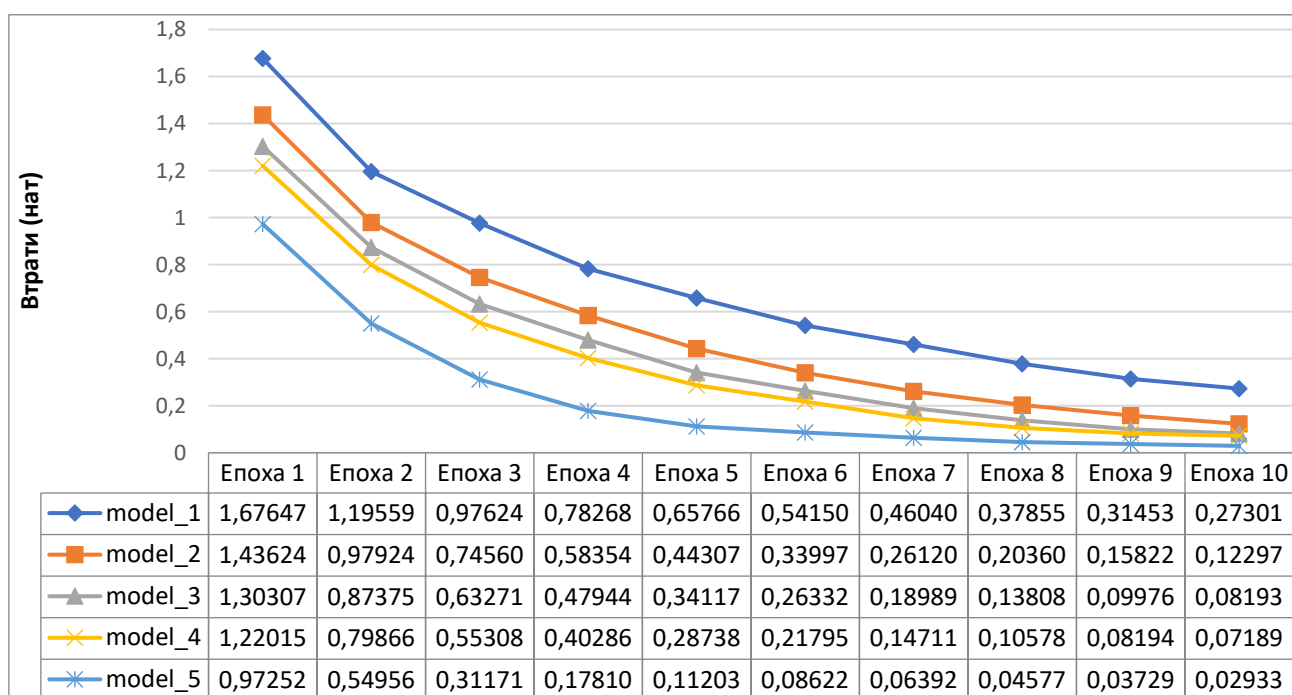


Рисунок Г.10 - Втрати SNN моделей на валідації на вокселях (менше - краще).

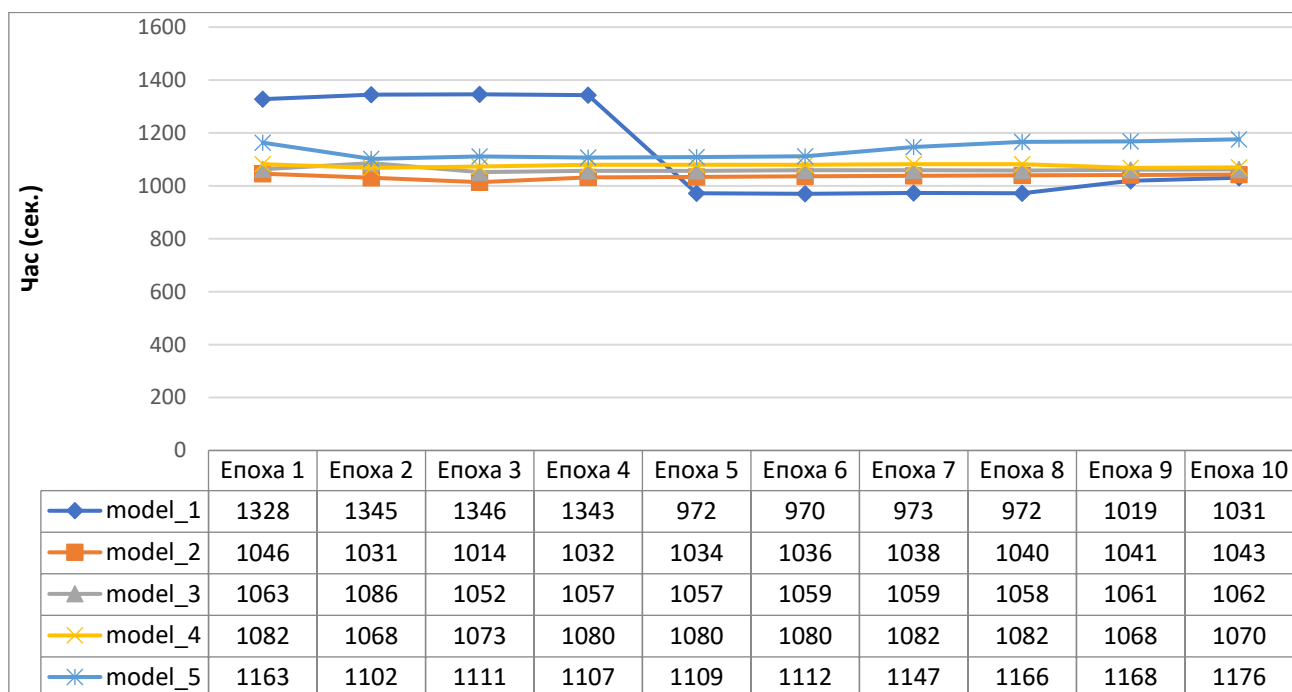


Рисунок Г.11 - Час навчання SNN моделей на вокселях (менше - краще).

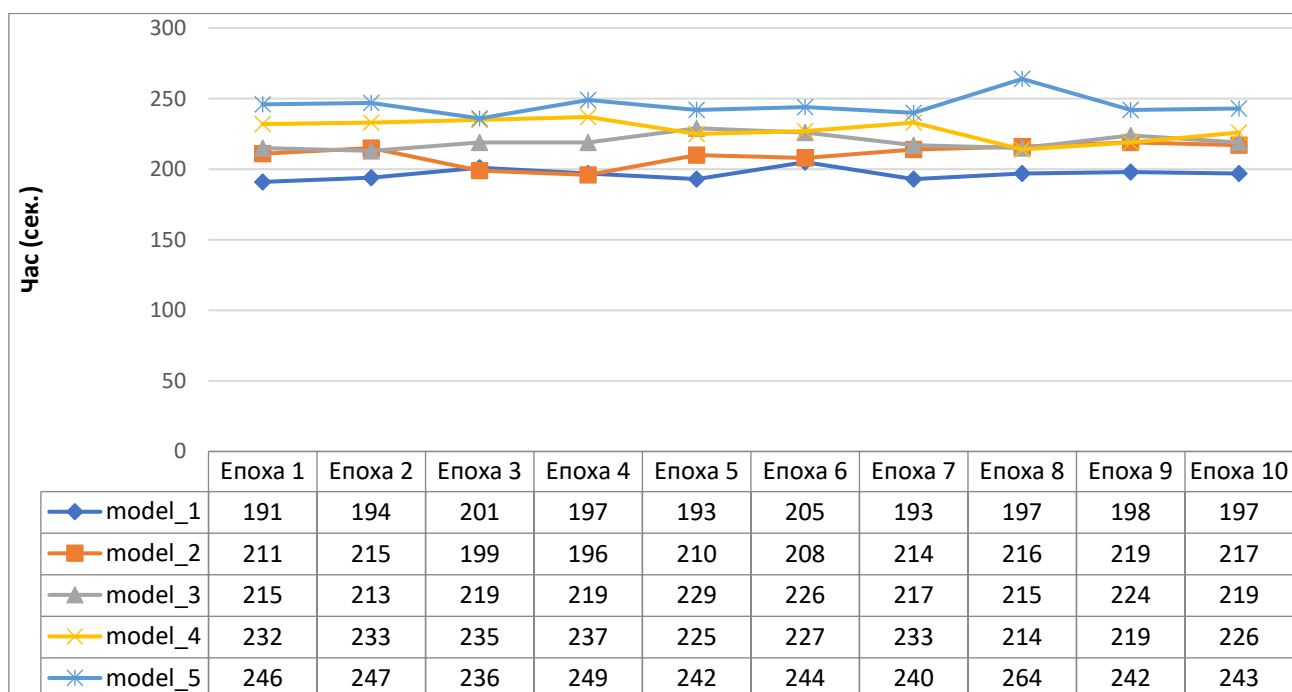


Рисунок Г.12 - Час роботи SNN моделей на вокселях (менше - краще).

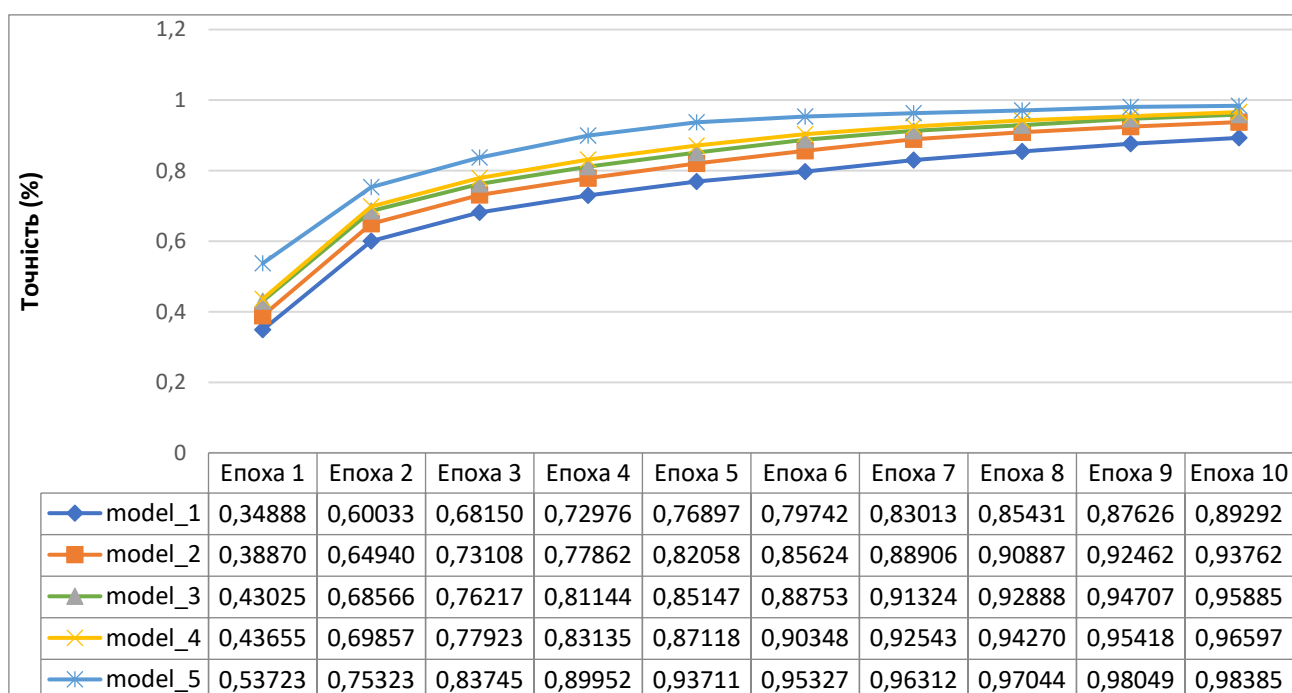


Рисунок Г.13 - Точність SNNO моделей на навчанні на вокселях (більше - краще).

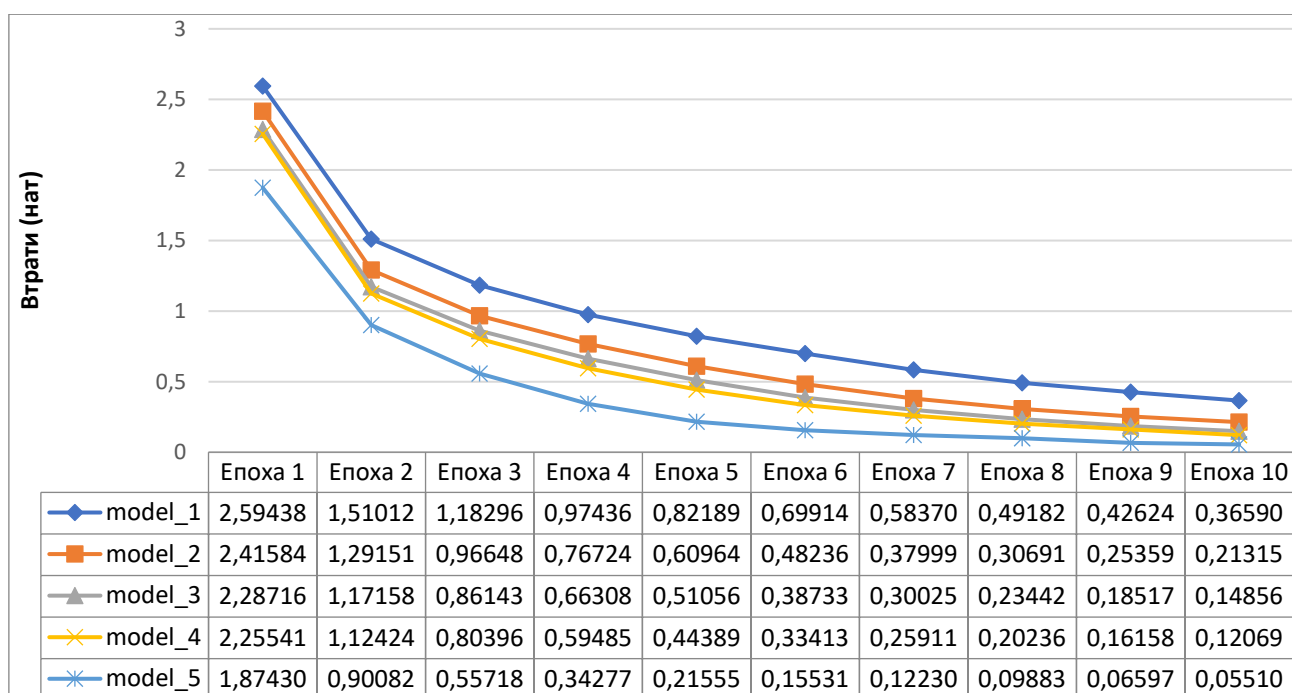


Рисунок Г.14 - Втрати SNNO моделей на навчанні на вокселях (менше - краще).

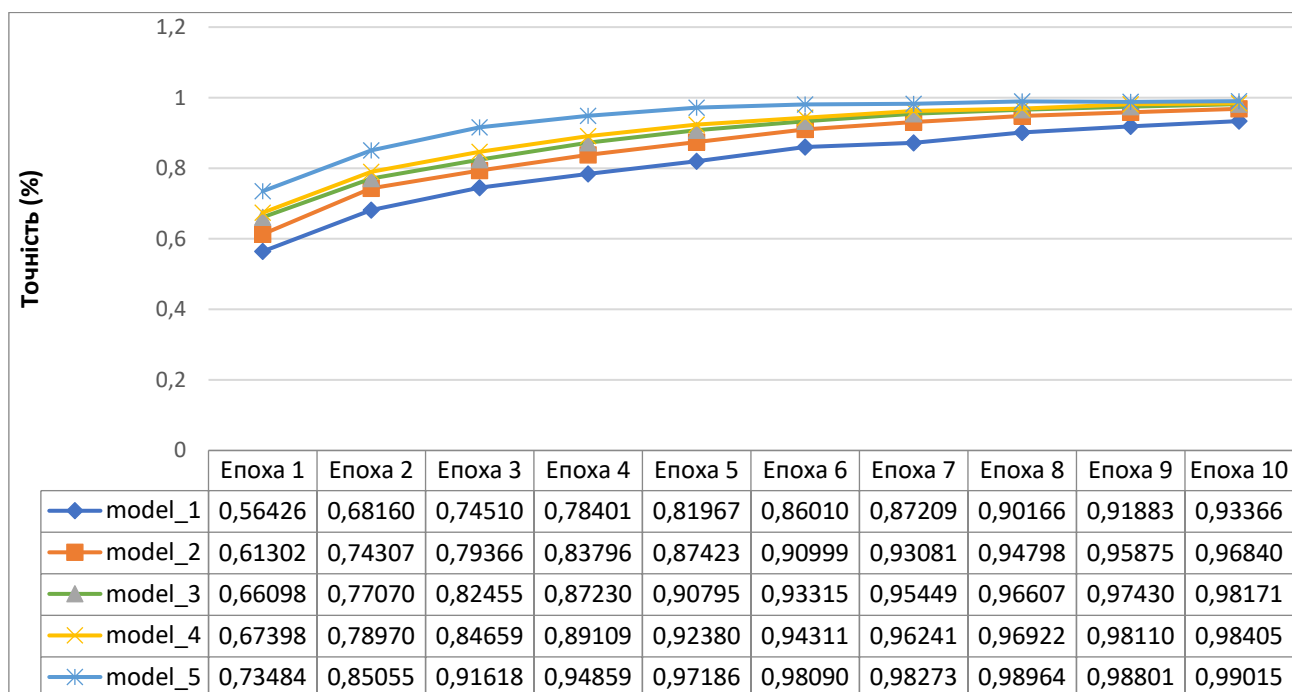


Рисунок Г.15 - Точність SNNO моделей на валідації на вокселях (більше - краще).

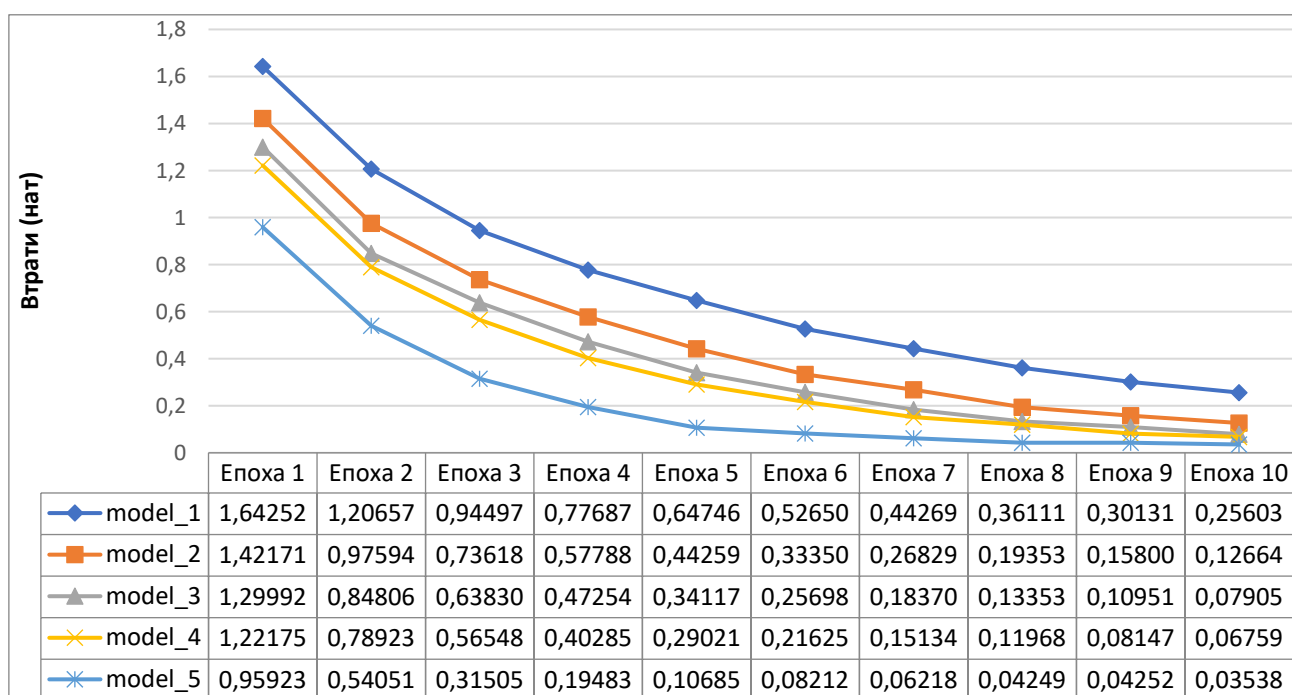


Рисунок Г.16 - Втрати SNNO моделей на валідації на вокселях (менше - краще).

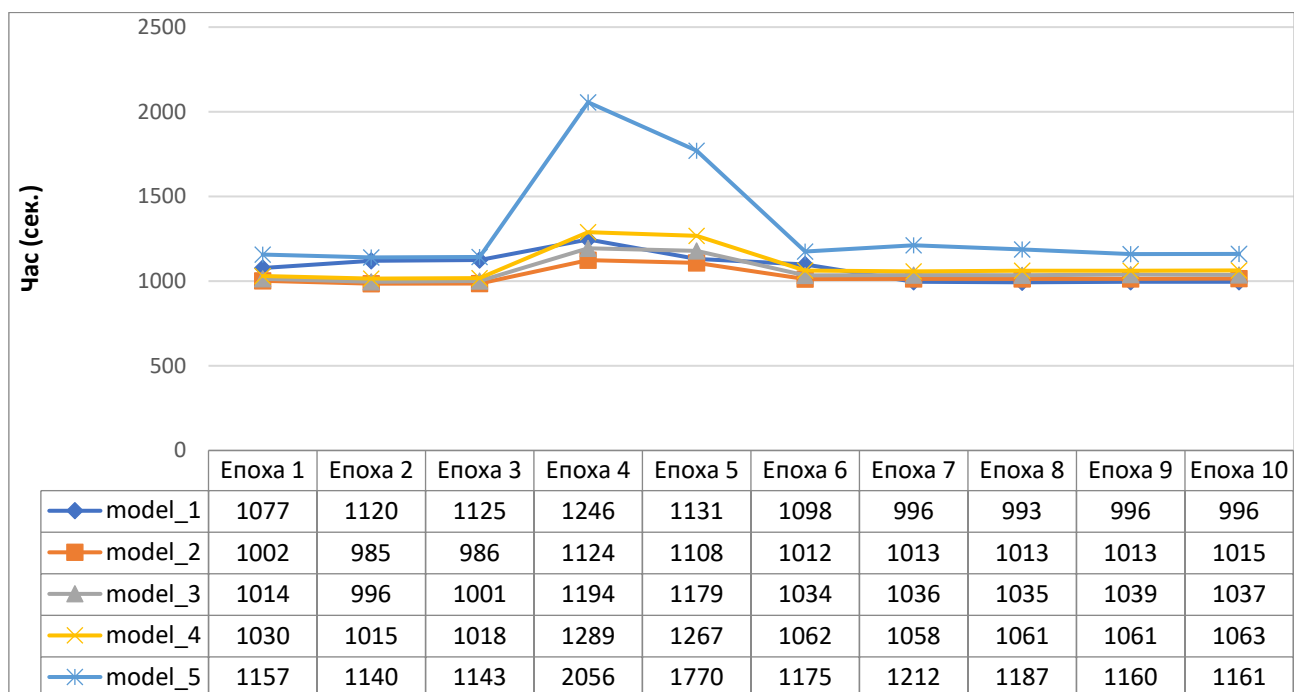


Рисунок Г.17 - Час навчання SNNO моделей на вокселях (менше - краще).

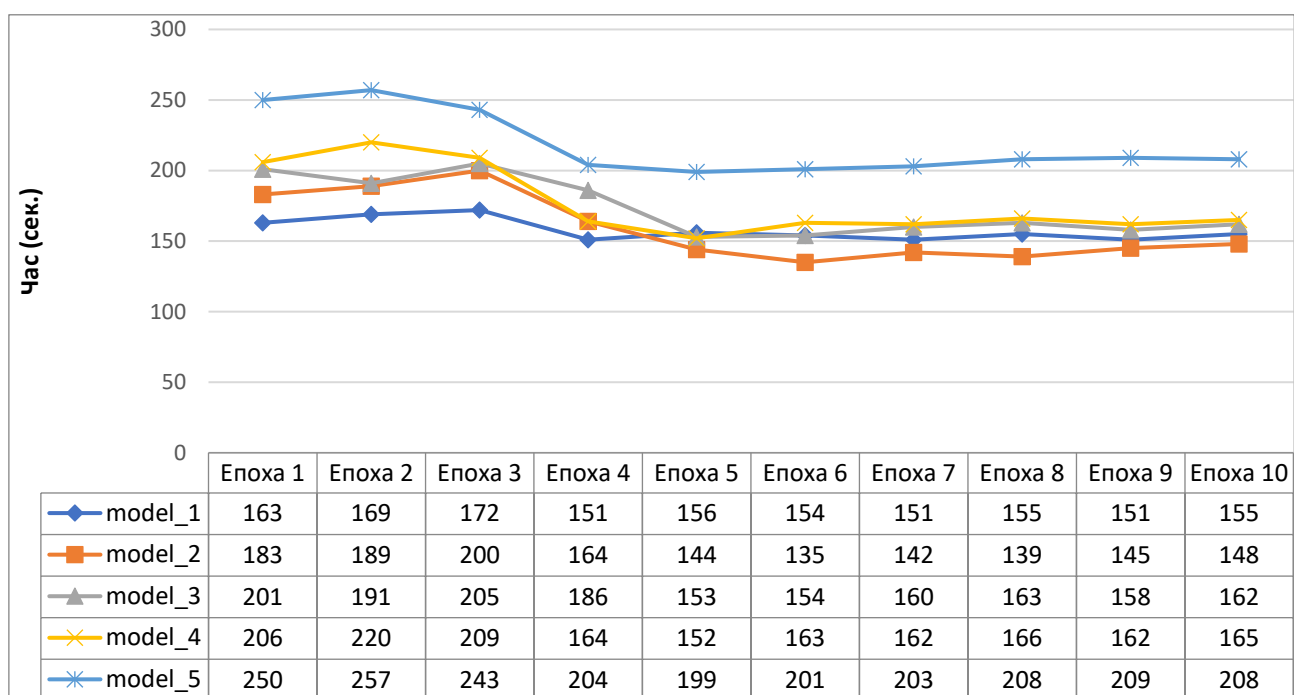


Рисунок Г.18 - Час роботи SNNO моделей на вокселях (менше - краще).

Результати класифікаторів для роботи з сітками

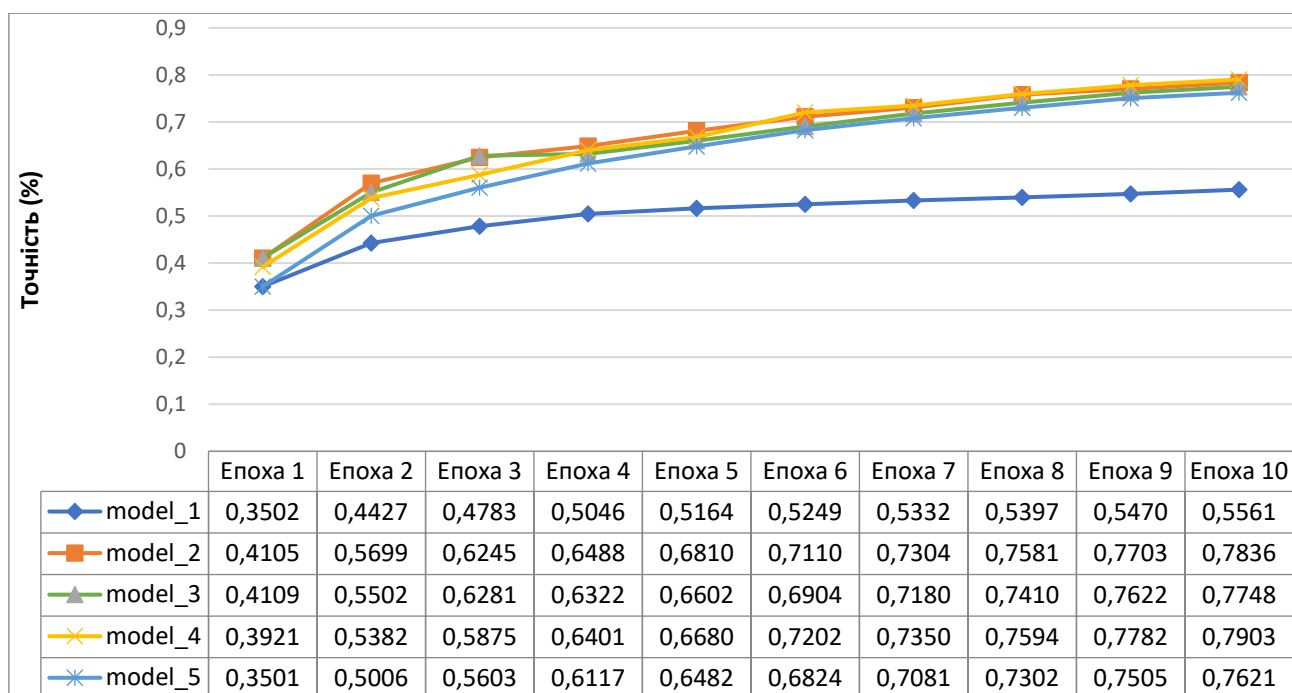


Рисунок Г.19 - Точність CNN моделей на навчанні на сітках (більше - краще).

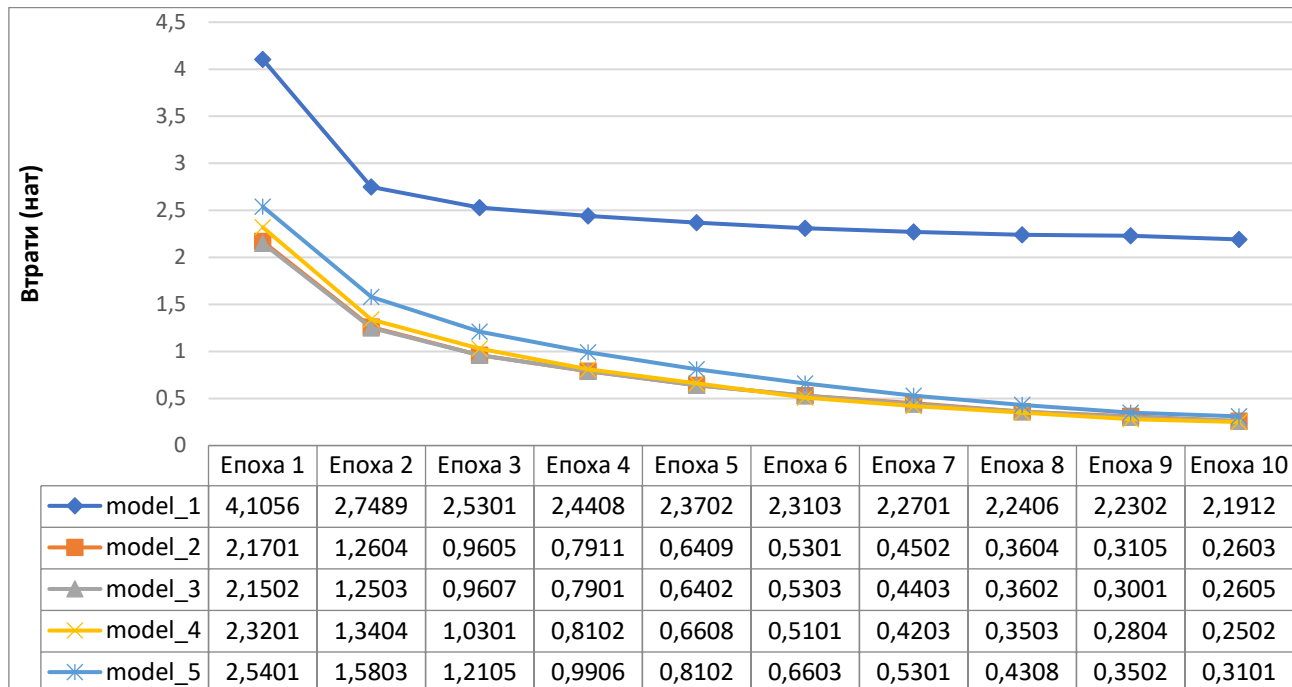


Рисунок Г.20 - Втрати CNN моделей на навчанні на сітках (менше - краще).

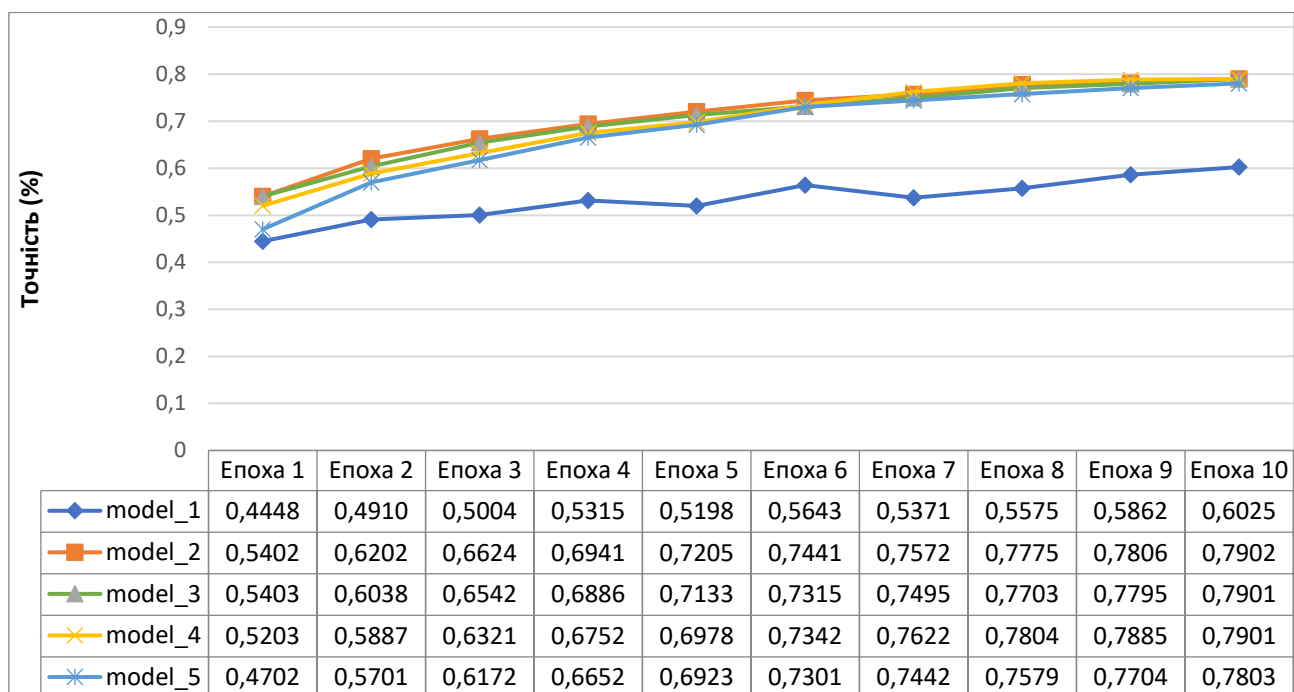


Рисунок Г.21 - Точність CNN моделей на валідації на сітках (більше - краще).

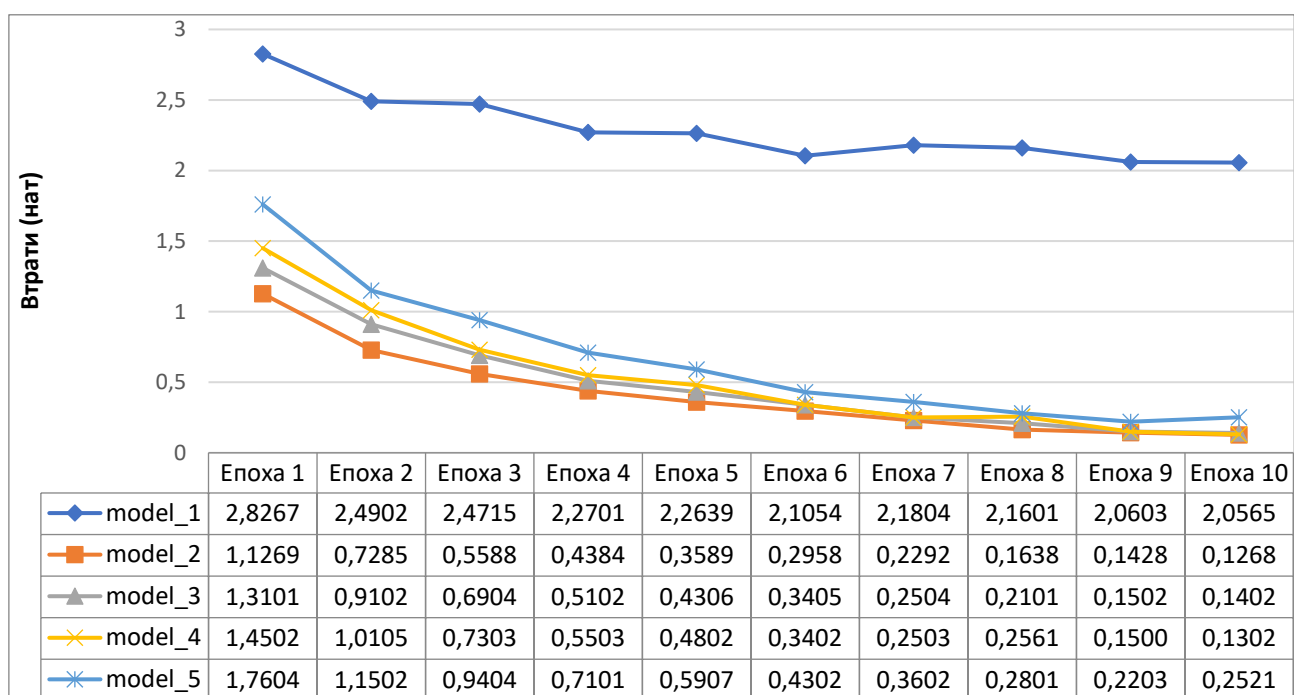


Рисунок Г.22 - Втрати CNN моделей на валідації на сітках (менше - краще).

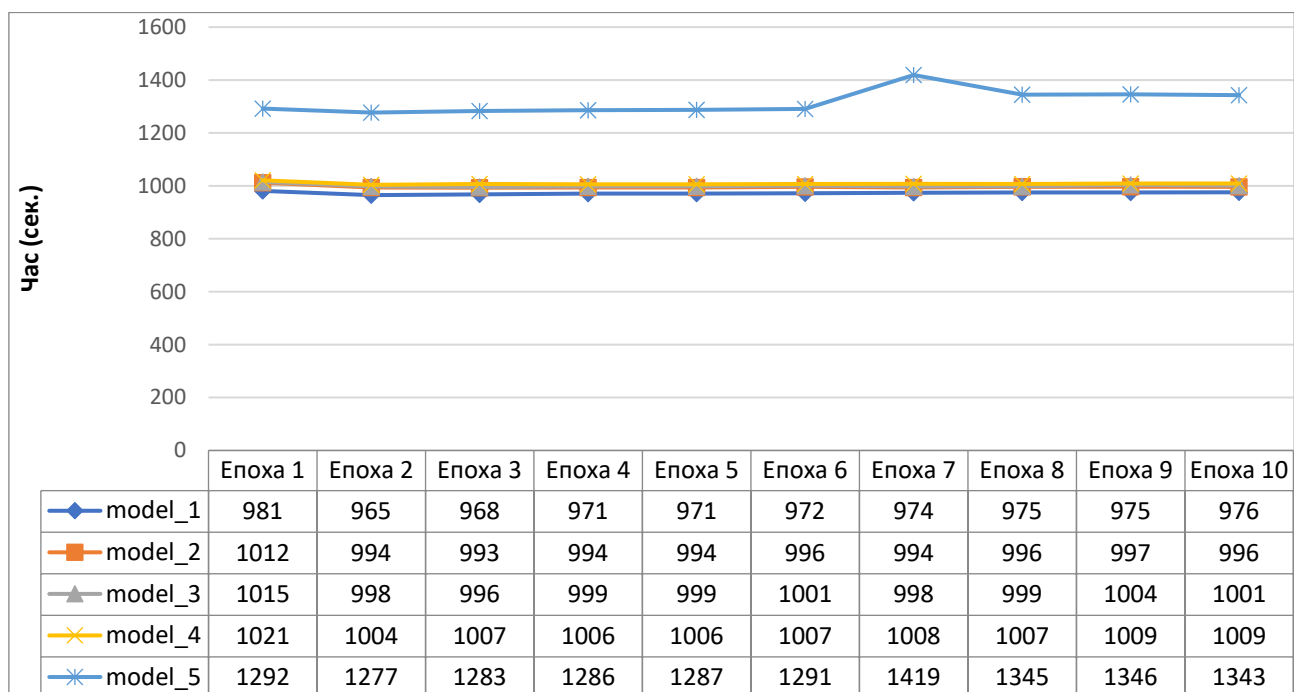


Рисунок Г.23 - Час навчання CNN моделей на сітках (менше - краще).

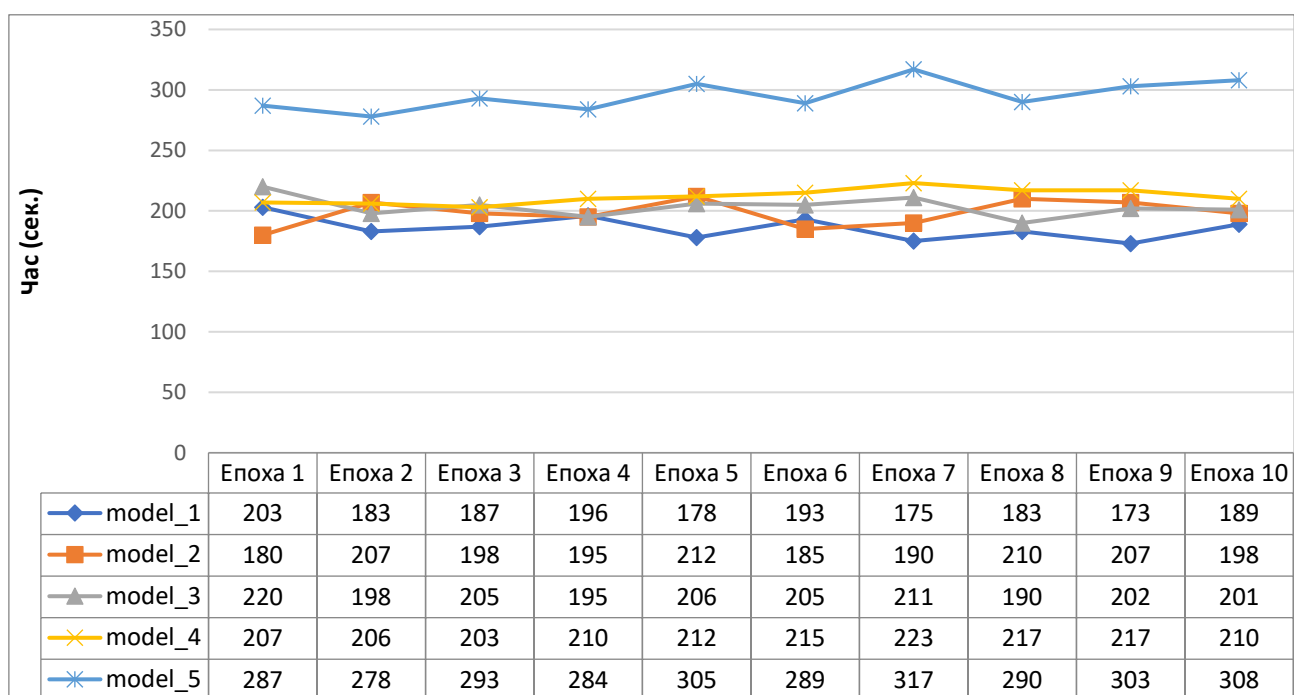


Рисунок Г.24 - Час роботи CNN моделей на сітках (менше - краще).

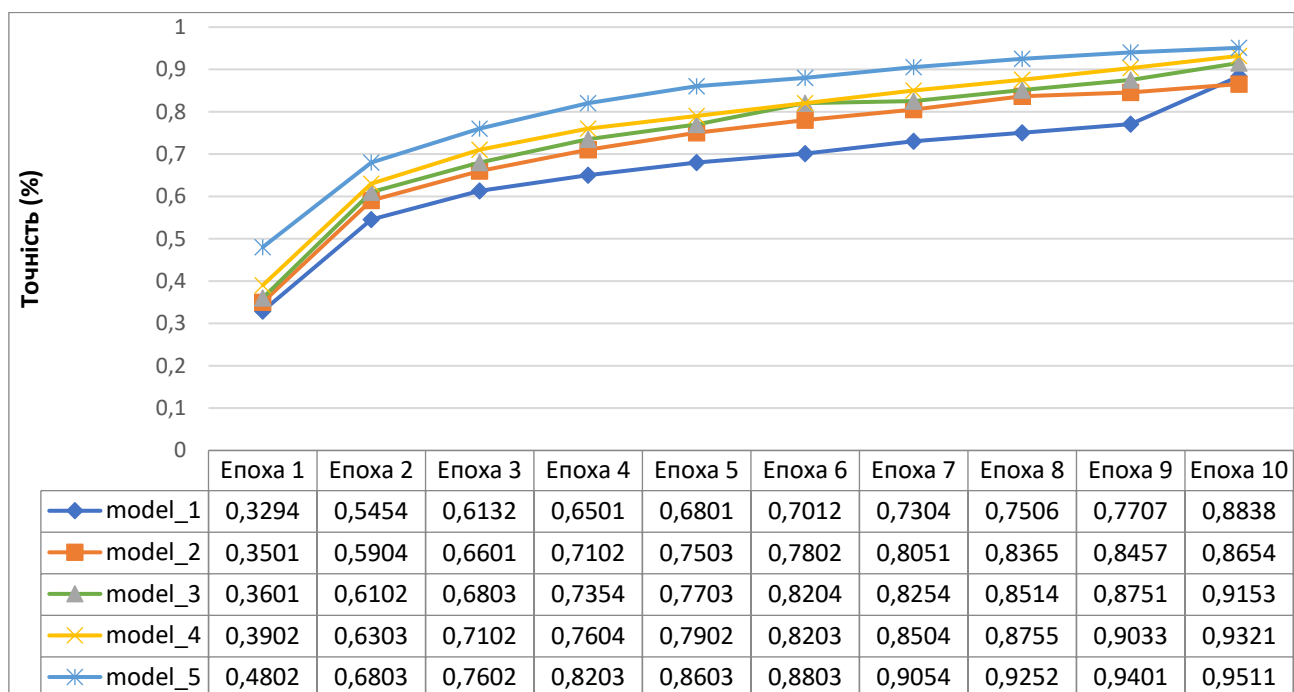


Рисунок Г.25 - Точність SNN моделей на навчанні на сітках (більше - краще).

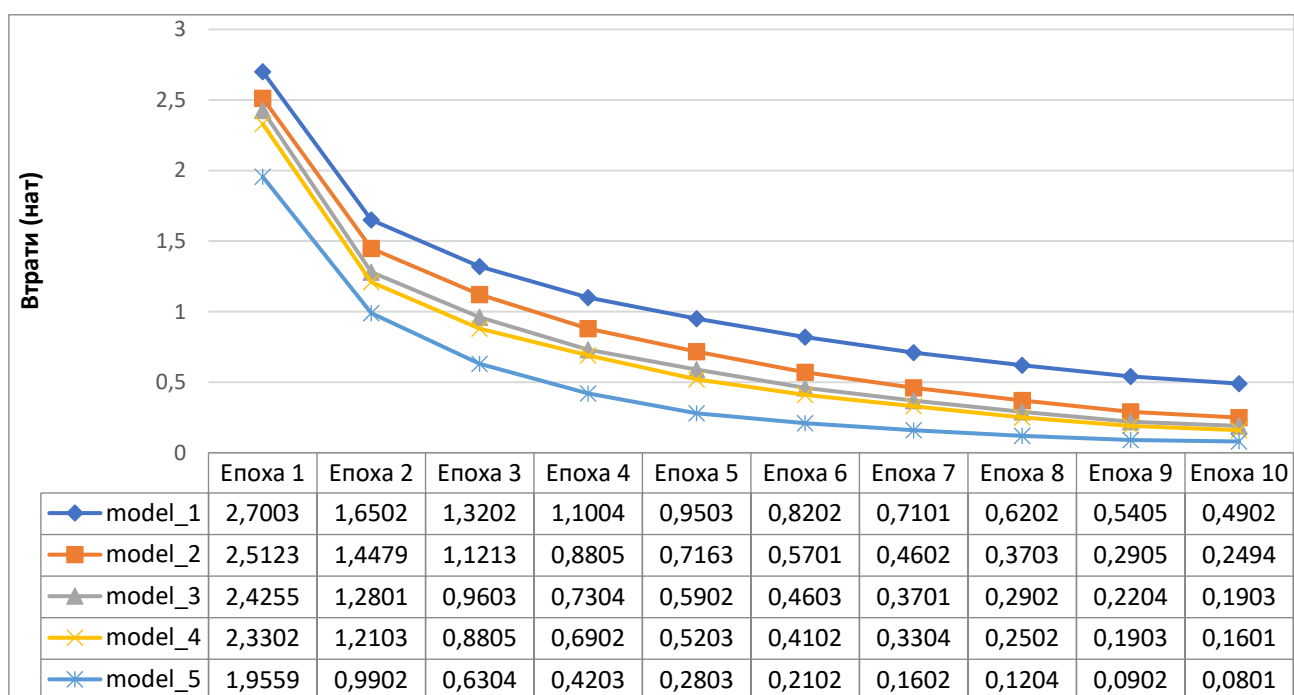


Рисунок Г.26 - Втрати SNN моделей на навчанні на сітках (менше - краще).

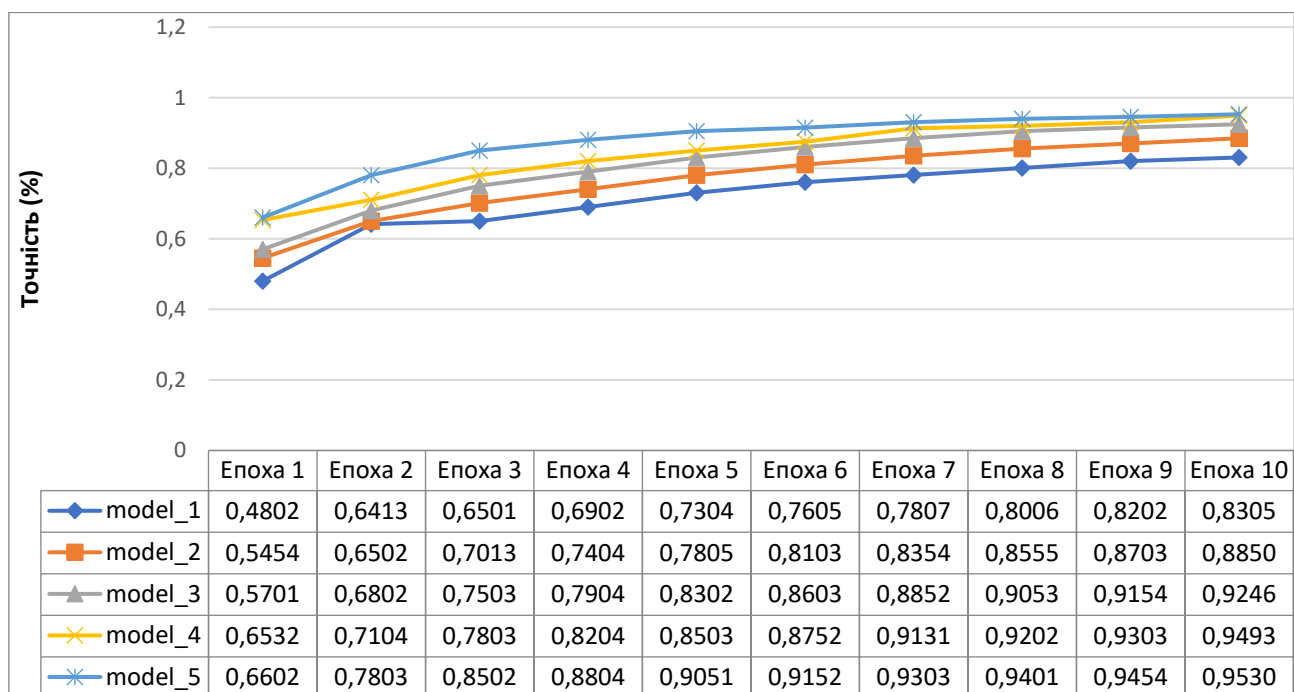


Рисунок Г.27 - Точність SNN моделей на валідації на сітках (більше - краще).

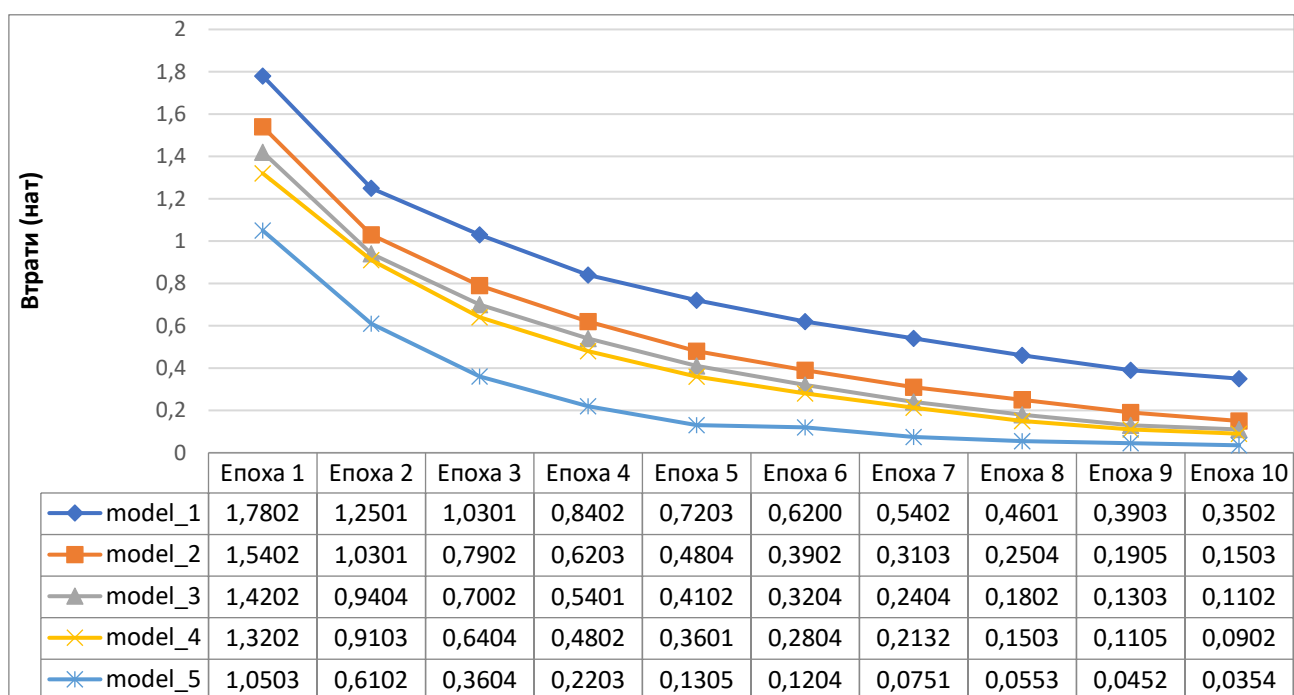


Рисунок Г.28 - Втрати SNN моделей на валідації на сітках (менше - краще).

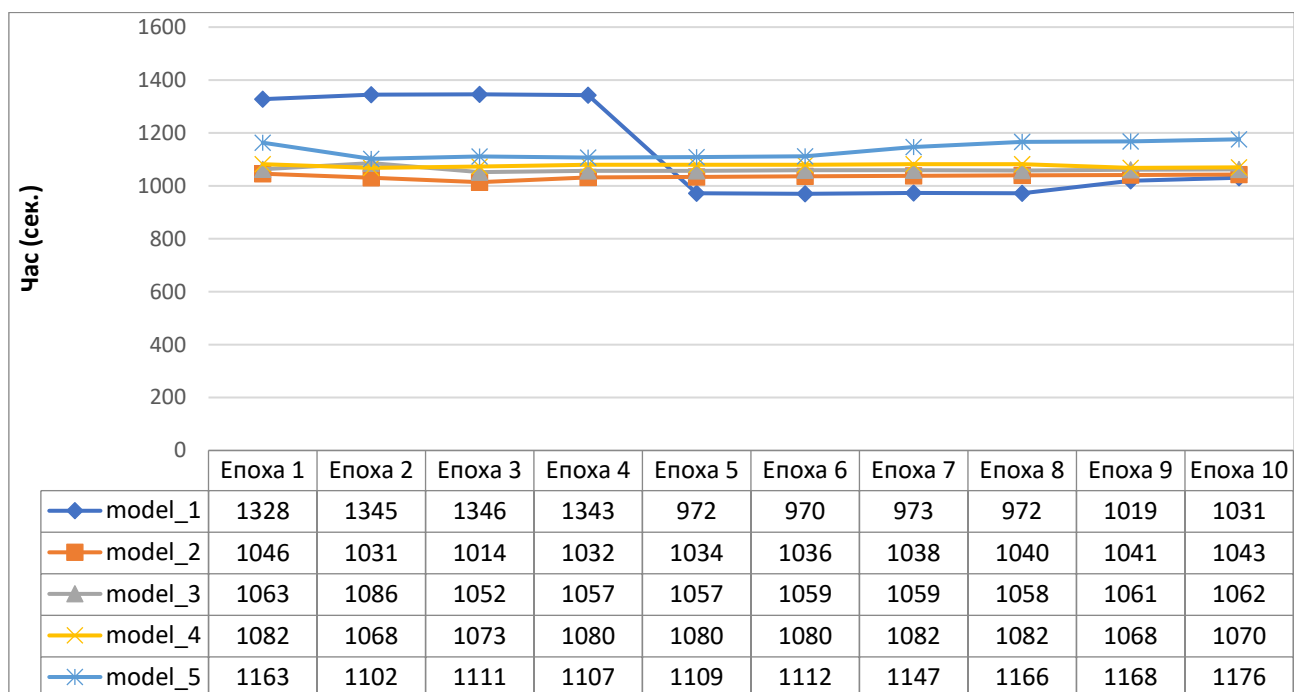


Рисунок Г.29 - Час навчання SNN моделей на сітках (менше - краще).

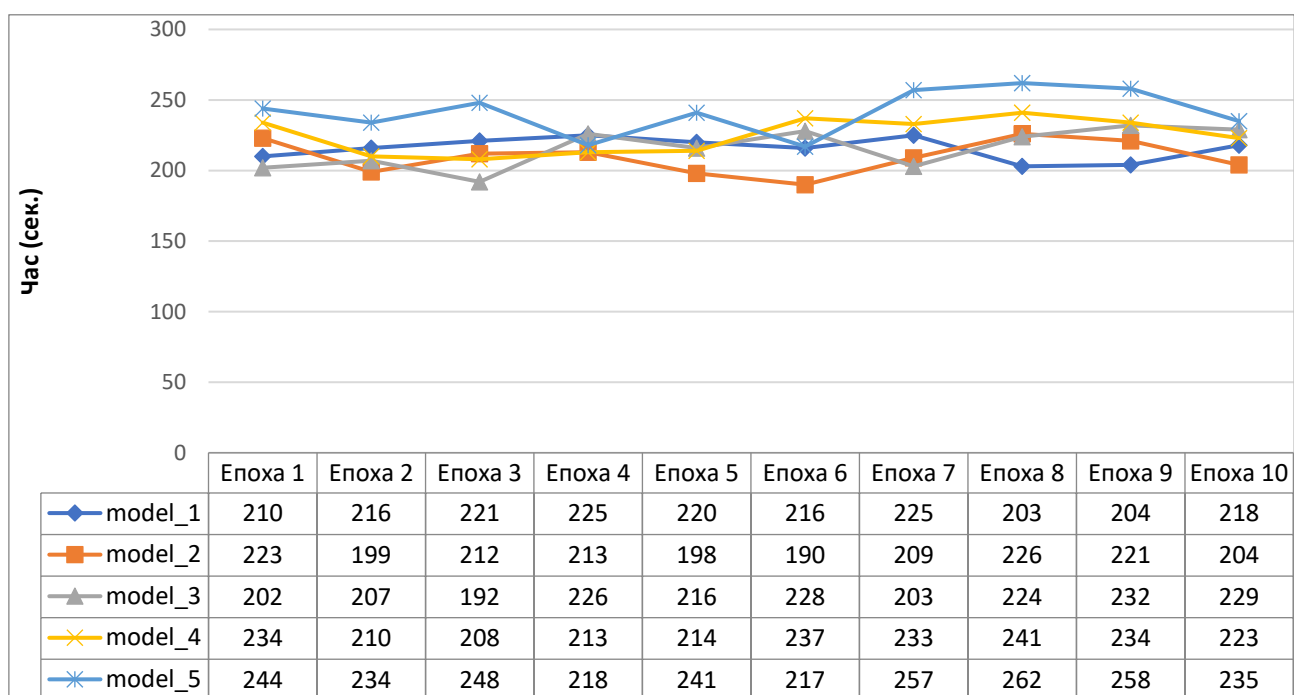


Рисунок Г.30 - Час роботи SNN моделей на сітках (менше - краще).

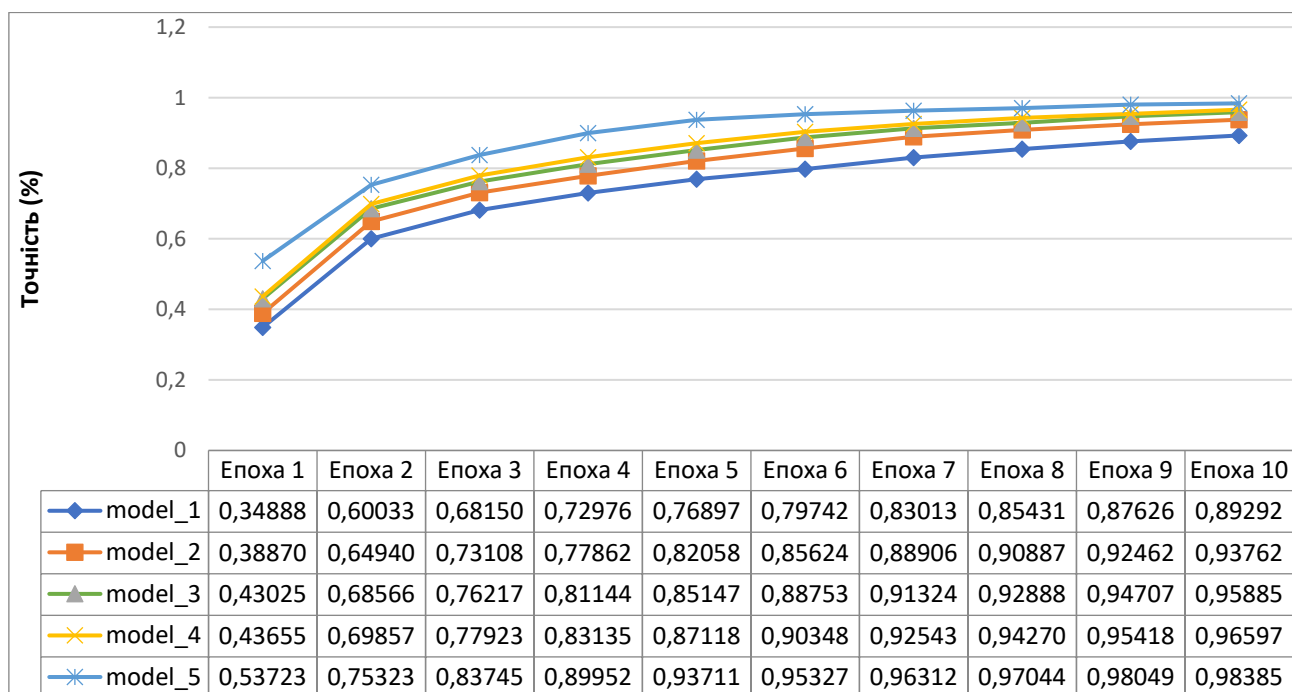


Рисунок Г.31 - Точність SNNO моделей на навчанні на сітках (більше - краще).

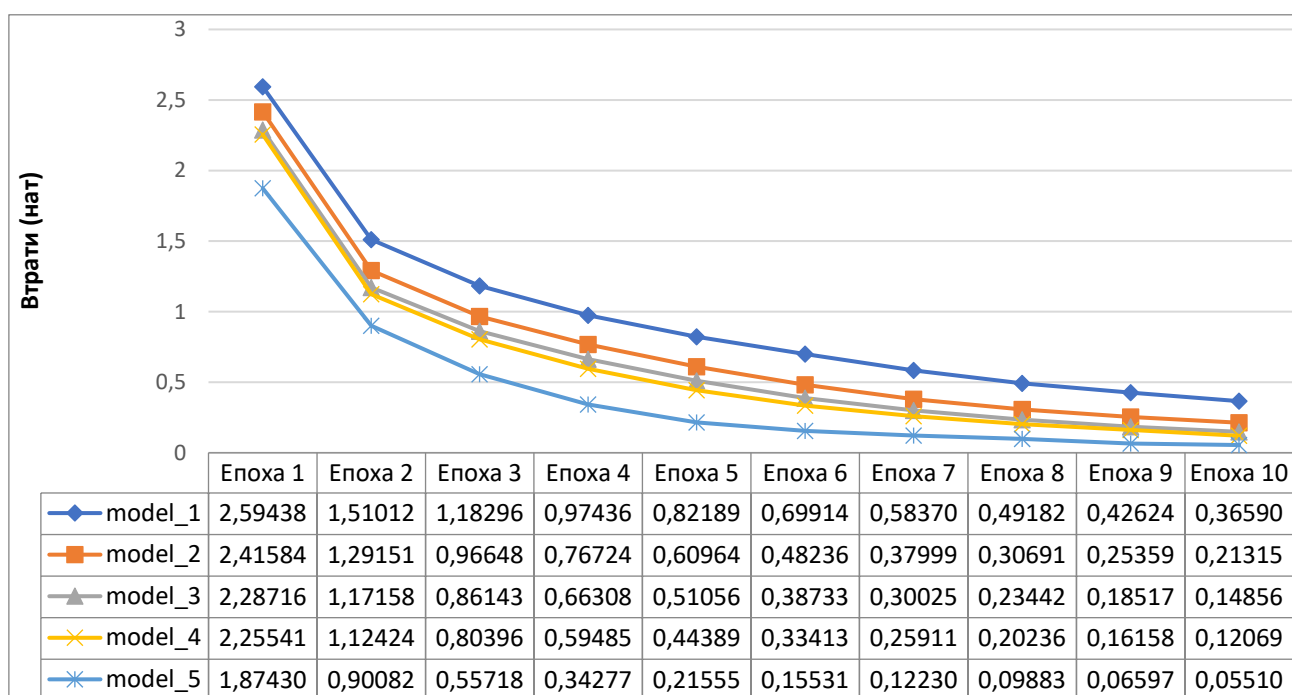


Рисунок Г.32 - Втрати SNNO моделей на навчанні на сітках (менше - краще).

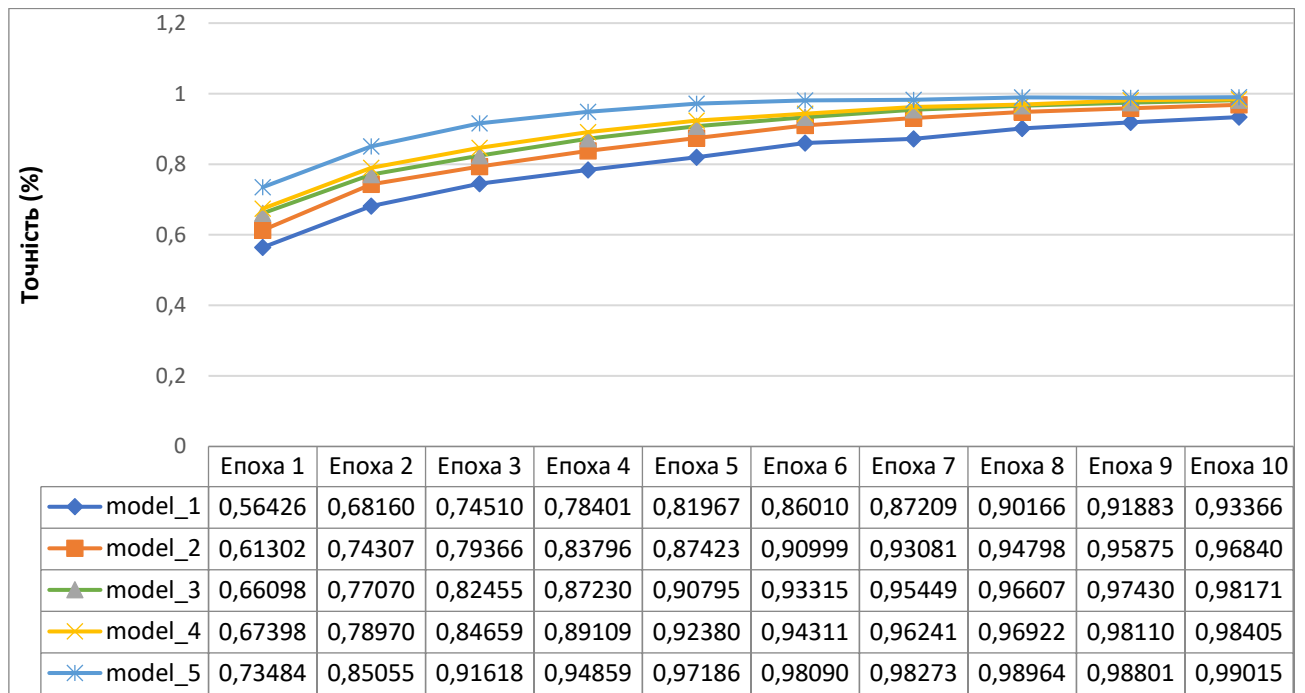


Рисунок Г.33 - Точність SNNO моделей на валідації на сітках (більше - краще).

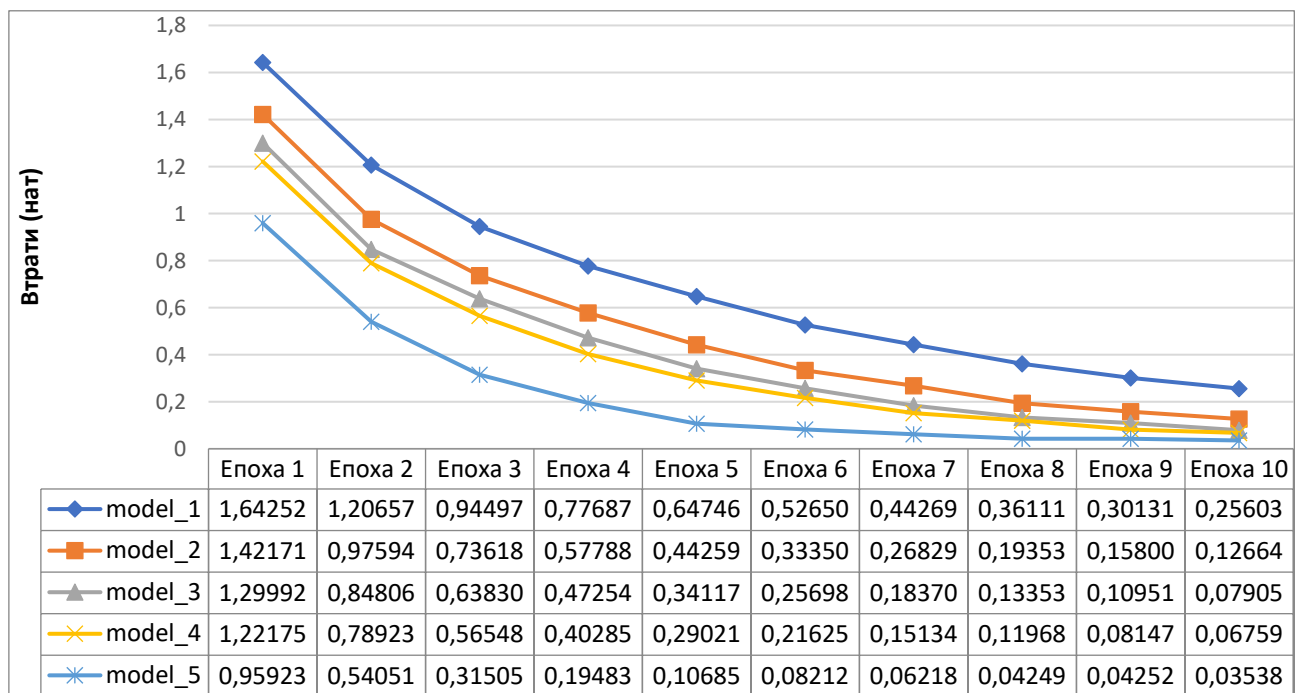


Рисунок Г.34 - Втрати SNNO моделей на валідації на сітках (менше - краще).

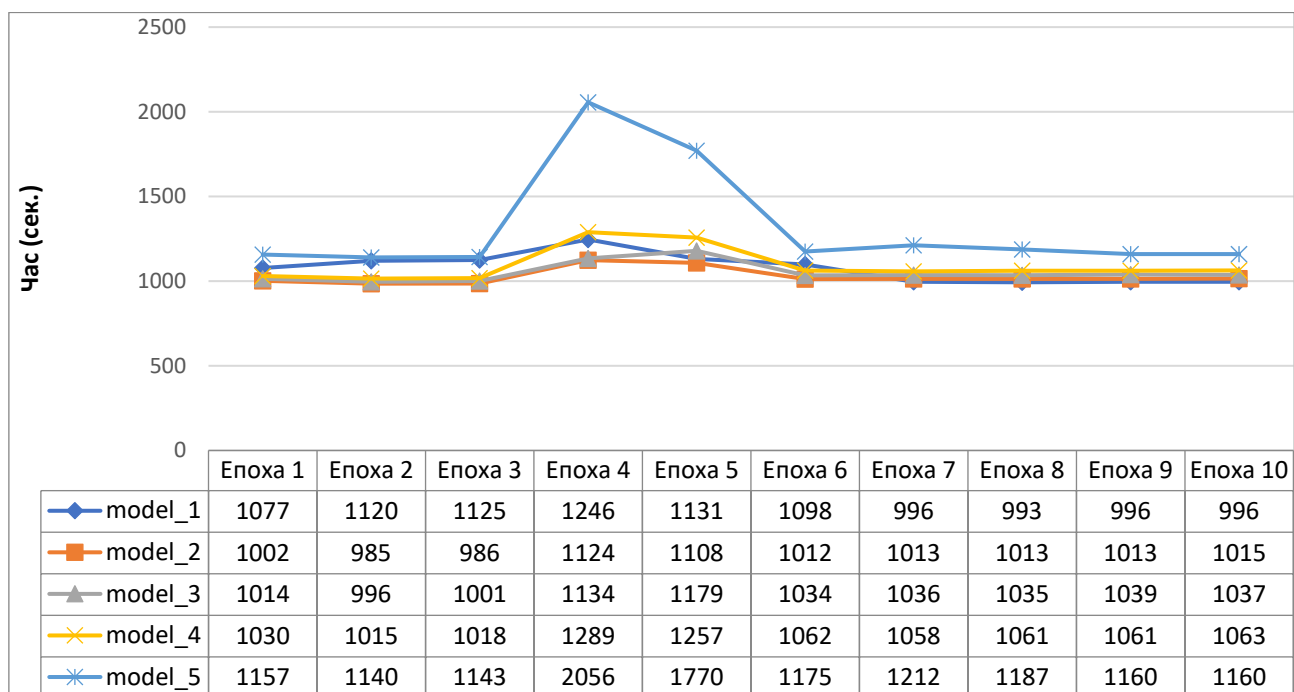


Рисунок Г.35 - Час навчання SNNO моделей на сітках (менше - краще).

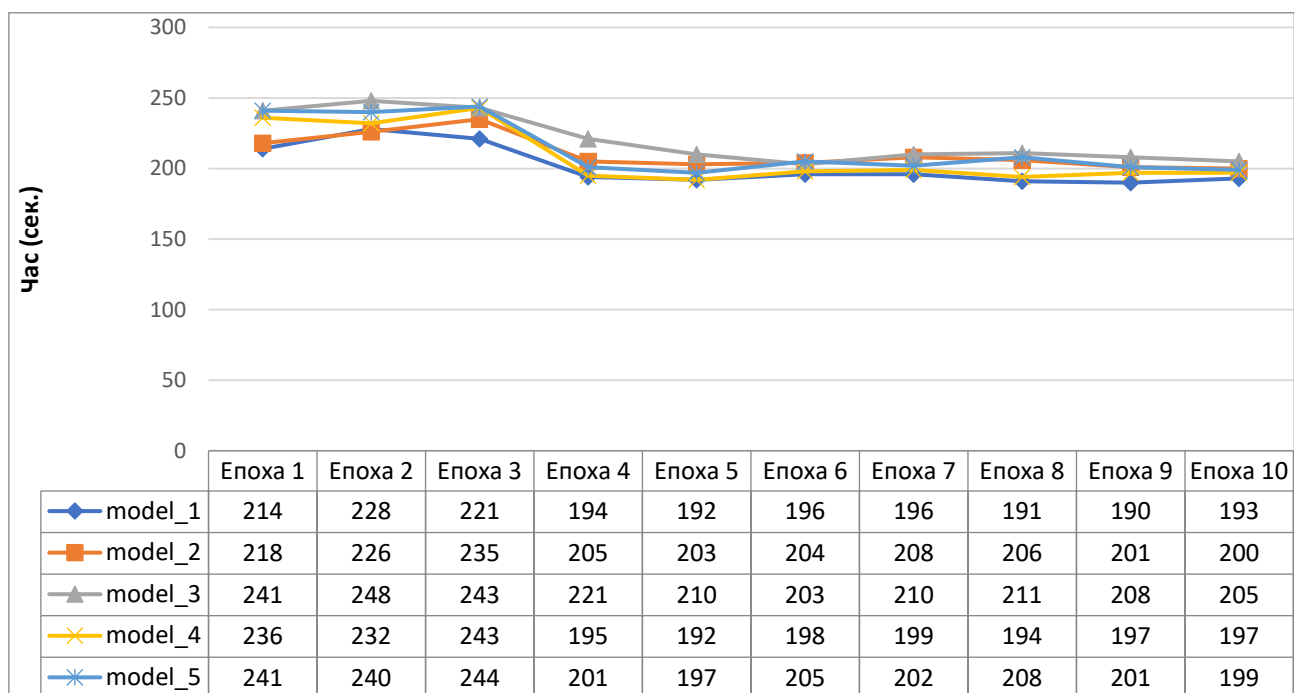


Рисунок Г.36 - Час роботи SNNO моделей на сітках (менше - краще).

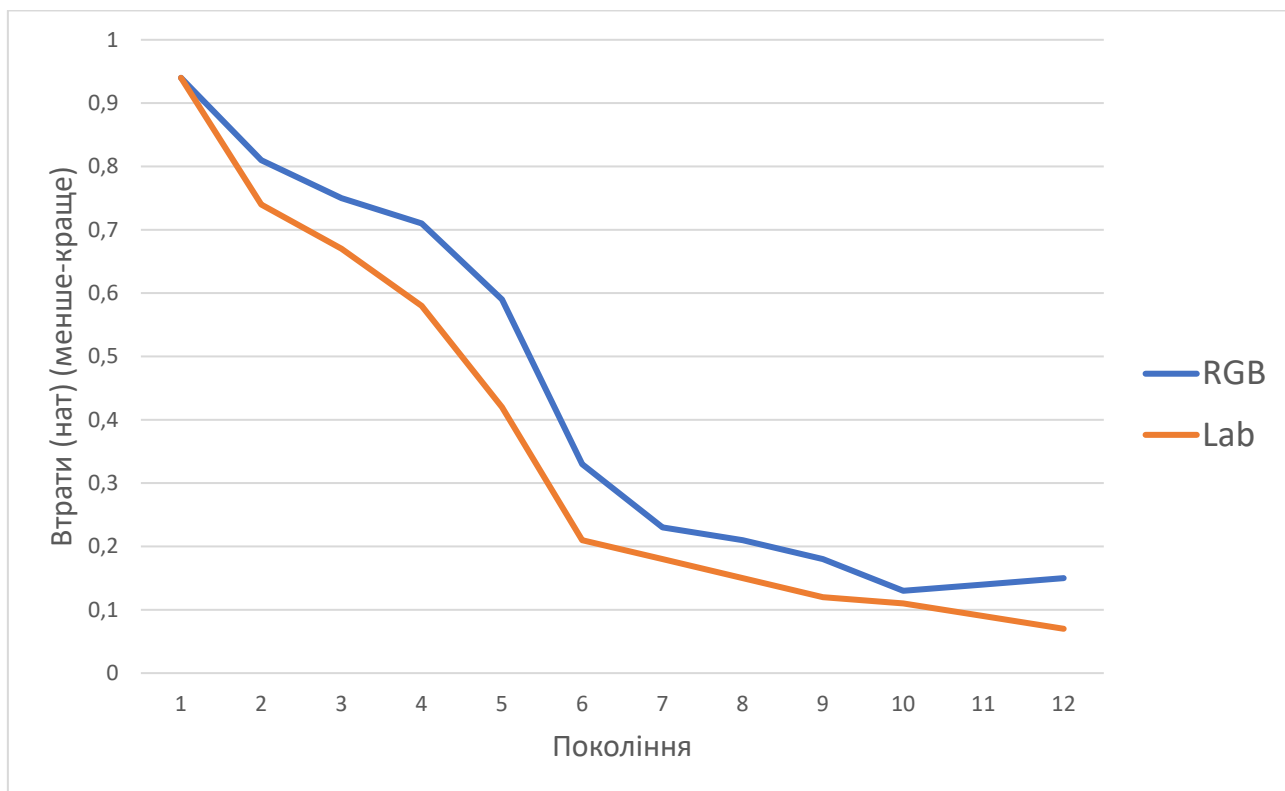


Рисунок Г.37 - Втрати моделей з різною колорифікаційною обробкою.

ДОДАТОК Д ДІАГРАМИ

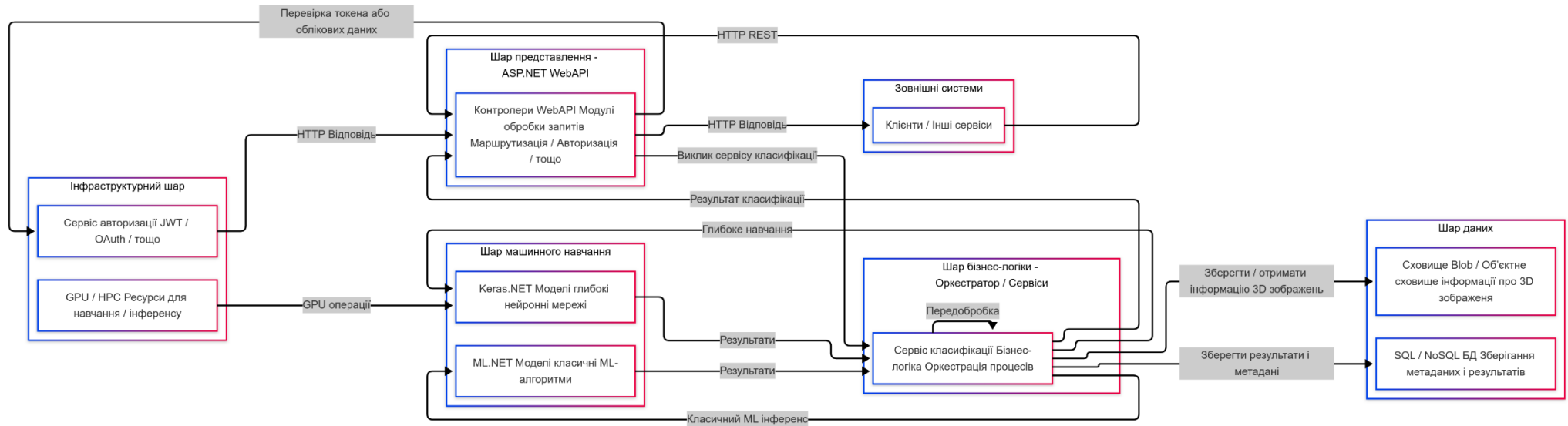


Рисунок Д.1 - Діаграма побудови програмних засобів.

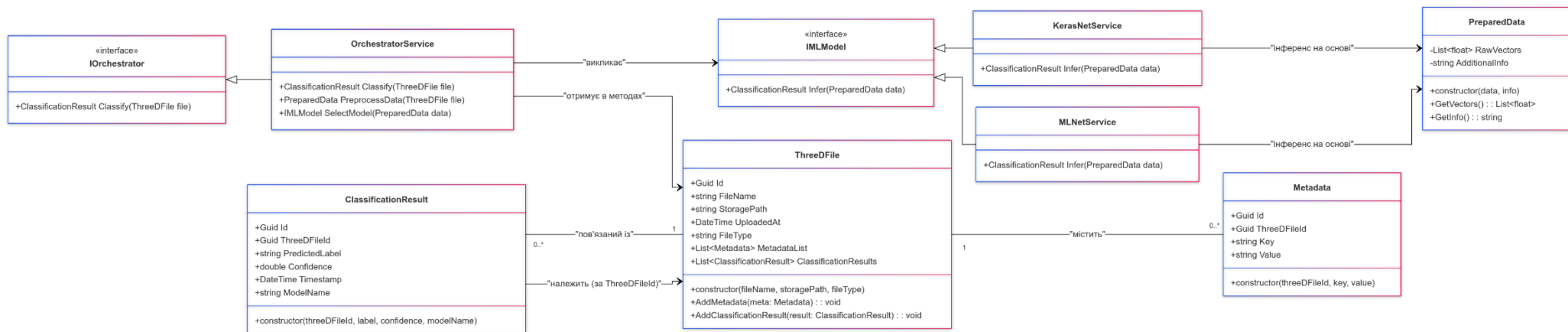


Рисунок Д.2 - Діаграма класів програмних засобів.

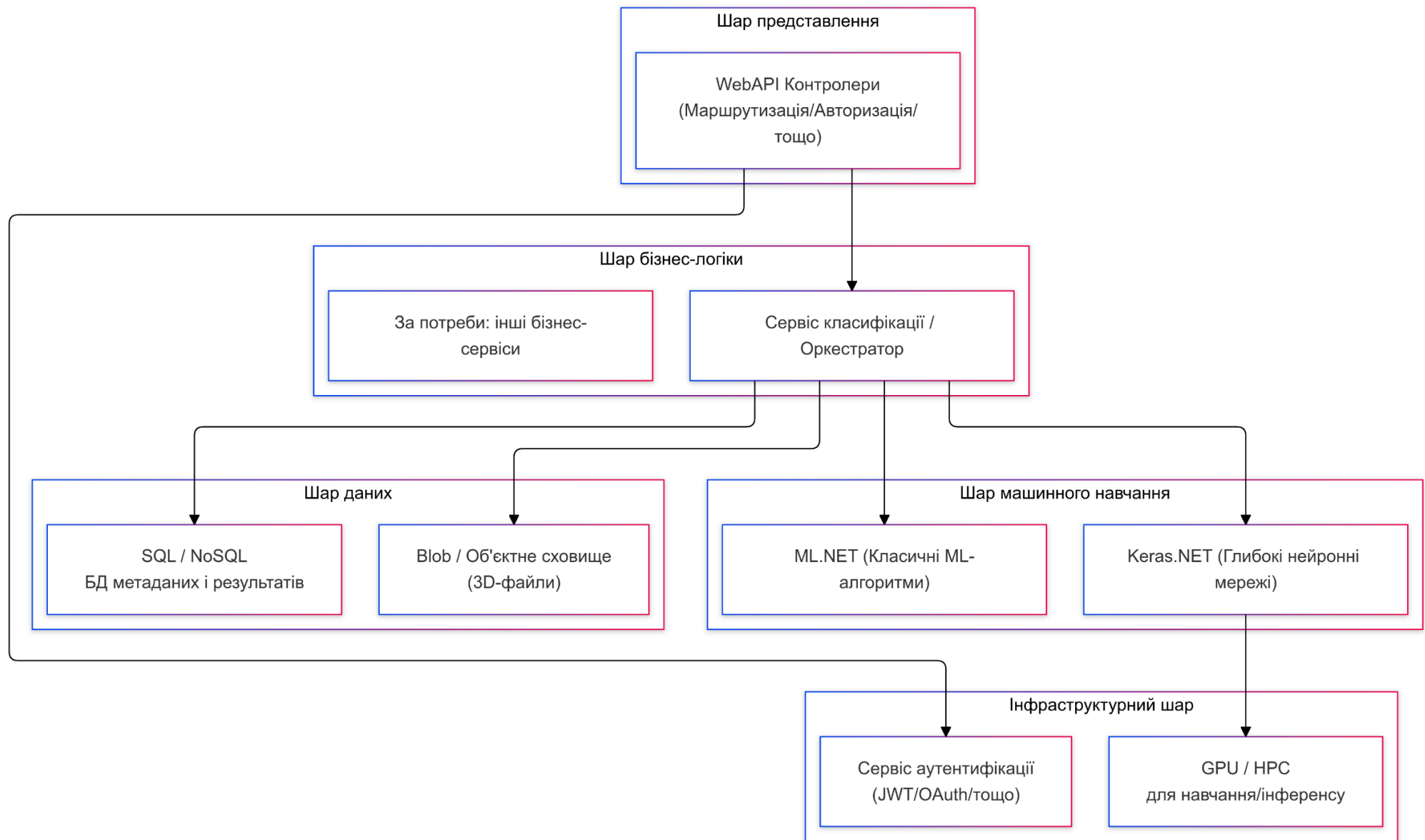


Рисунок Д.3 - Діаграма компонентів програмних засобів.

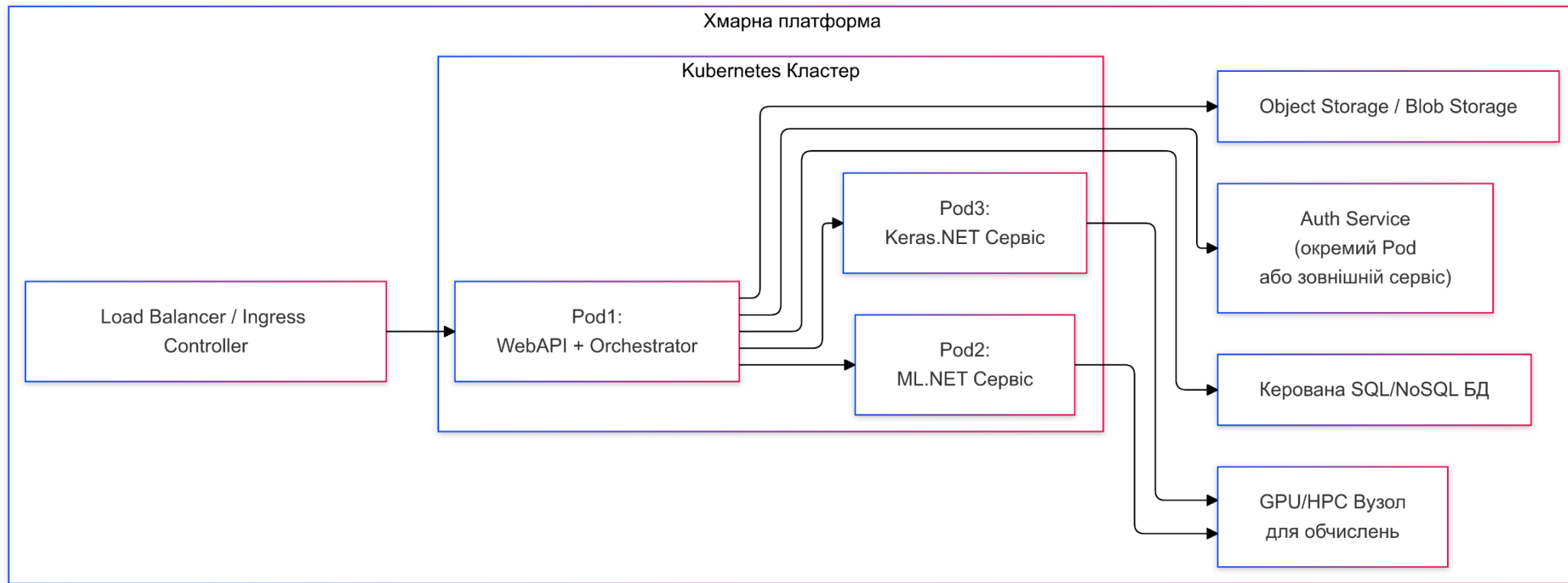


Рисунок Д.4 - Діаграма розгортання.

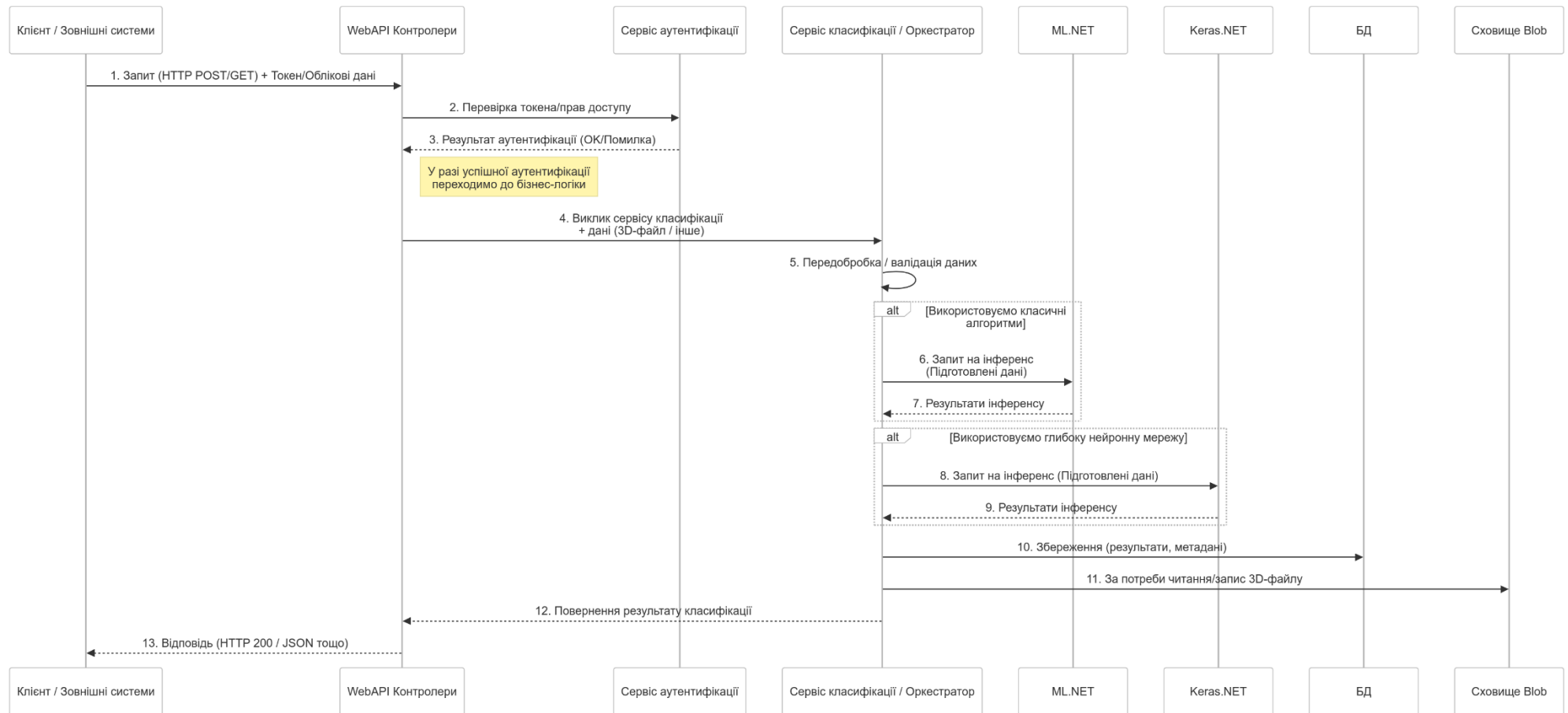


Рисунок Д.5 - Діаграма станів системи.

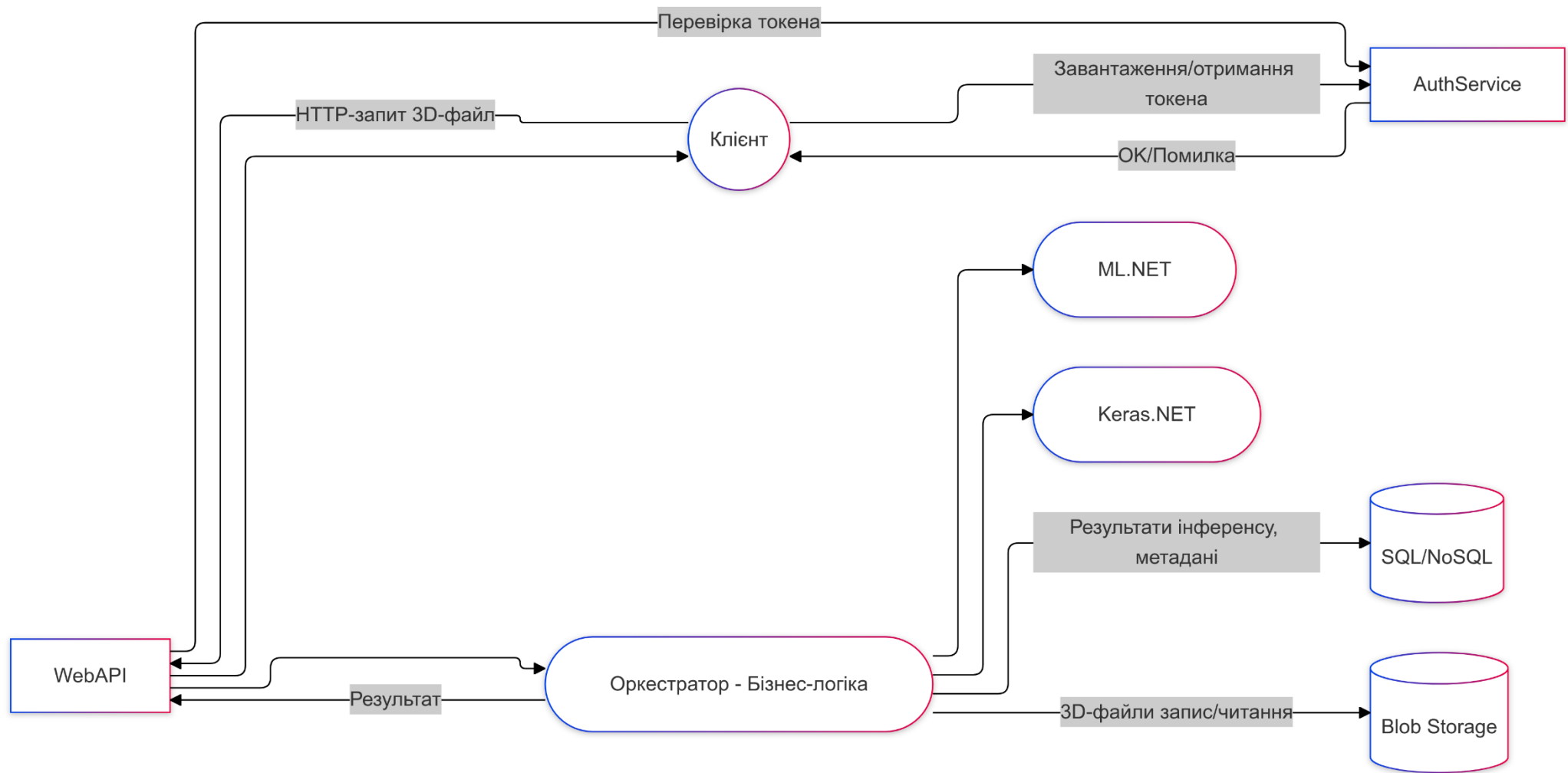


Рисунок Д.6 - Діаграма діяльності.

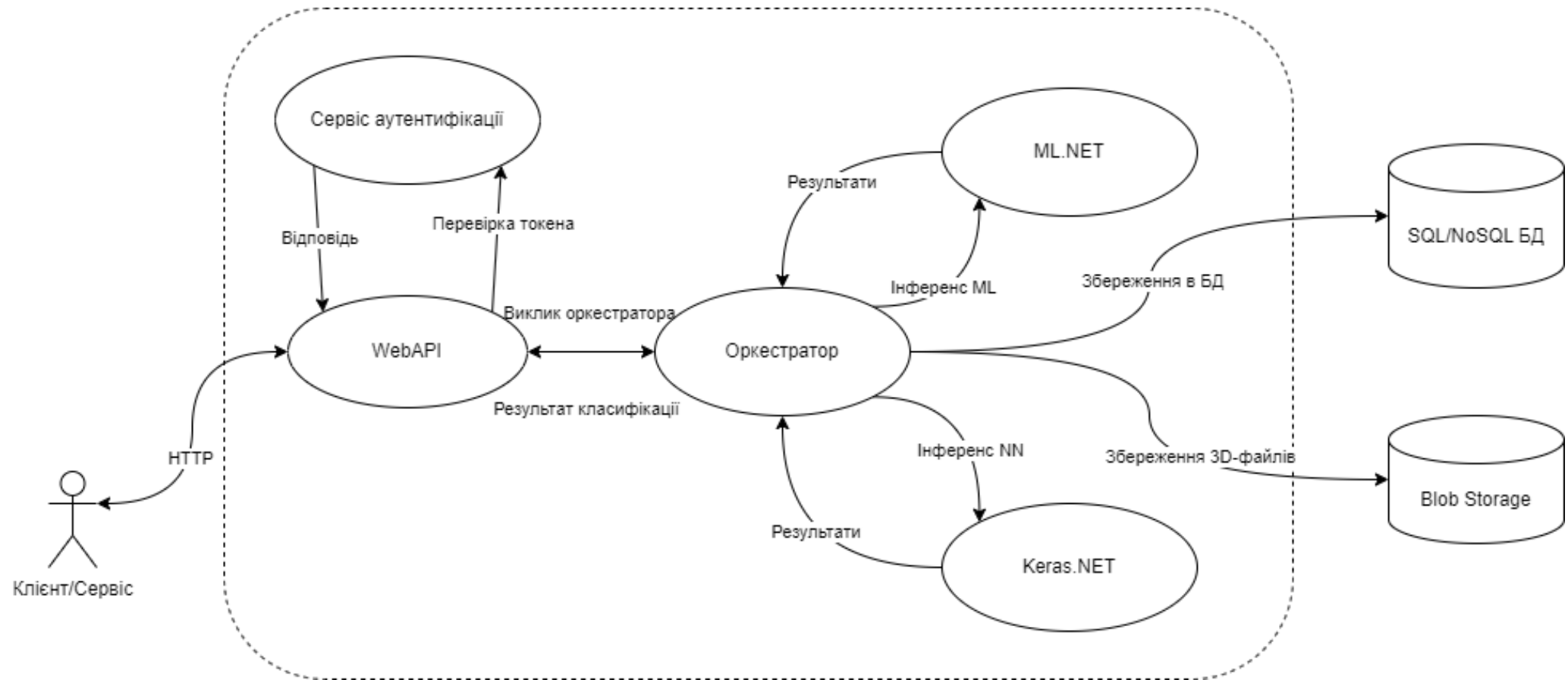


Рисунок Д.7 - Діаграма варіантів використання.

ДОДАТОК Е НАЯВНІ ПРОГРАМНІ ЗАСОБИ

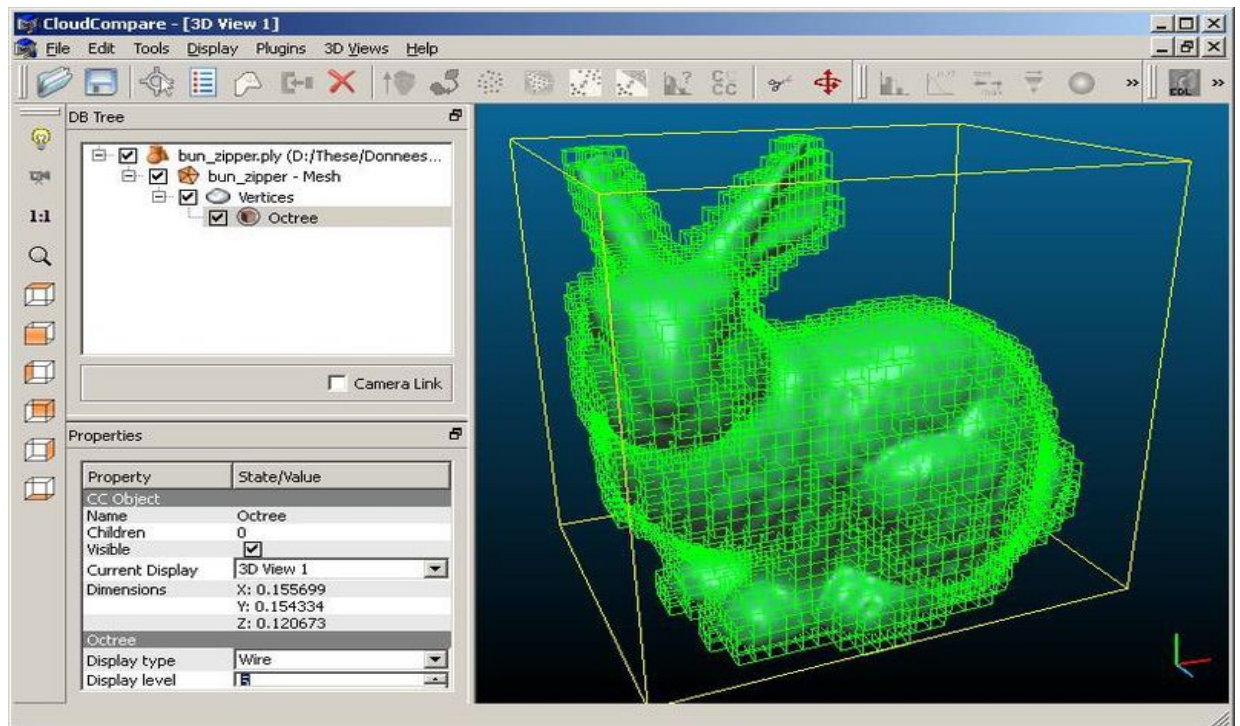


Рисунок Е.1 - Інтерфейс користувача CloudCompare.

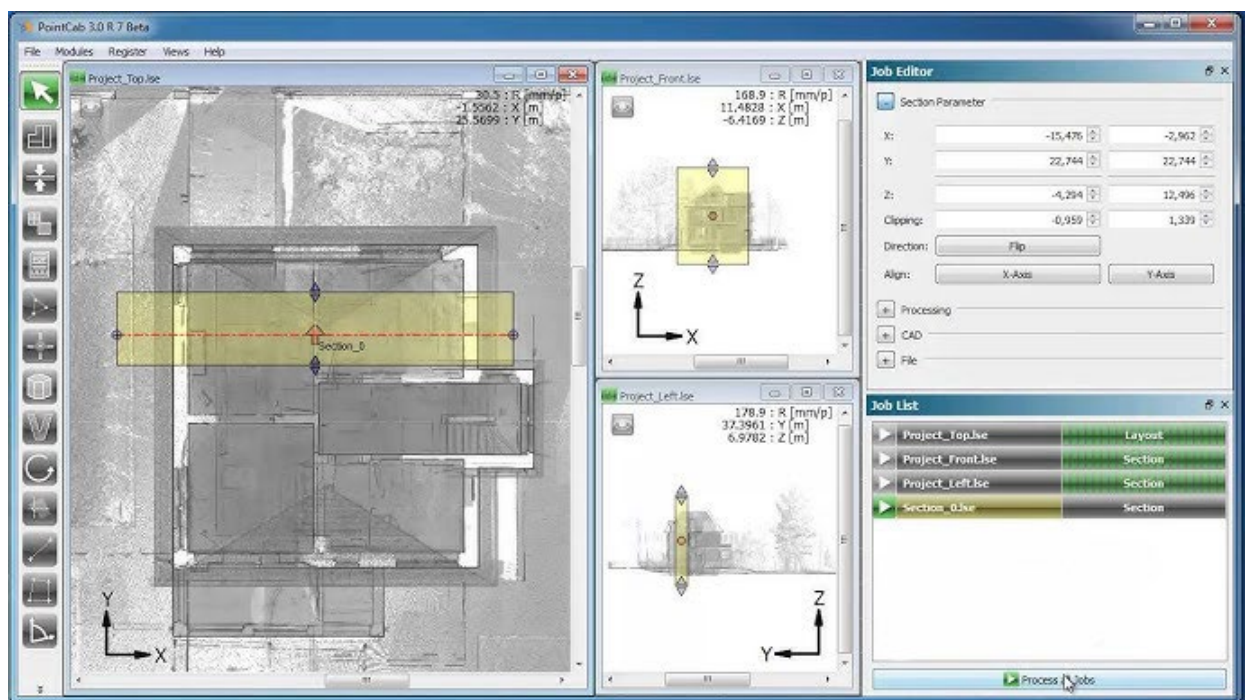


Рисунок Е.2 - Інтерфейс користувача PointCab.

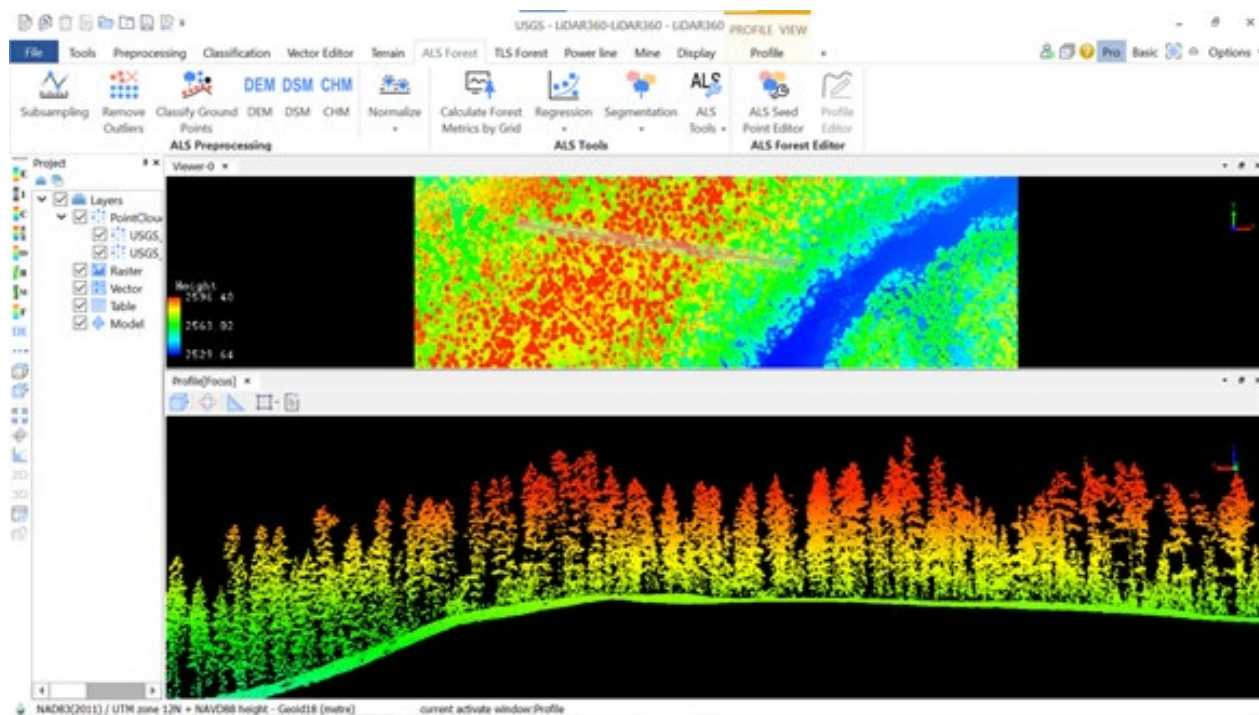


Рисунок Е.3 - Интерфейс користувача LiDAR360.